# Real-Time Hardware-in-the-Loop Simulation and Test Conductor Platforms

## Flight Software Workshop
## Dec 12-14, 2019

Presented By:
Ashley Lee (NASA)
& Pat Tobbe (DCI)

# ARTEMIS Introduction

- **ARTEMIS – Advanced Real Time Environment for Modeling, Integration, and Simulation**
  - Provides Real Time hardware-in-the-loop (HWIL) environment hosting system simulations (i.e. aerospace vehicles, robotic systems, etc) which stimulate hardware under test (avionics components, sensors, effectors, motion systems)
  - Reconfigurable for vehicle design, hardware / software interfaces, hardware under test, laboratory computer resources

# Why ARTEMIS?

- **Single source code tree supporting:**
  - Multiple model fidelities
    - Vehicle – flex body, rigid body
    - Selectable winds, atmosphere, TVC nozzle, etc.
  - Non real-time, all-digital
  - Real-time, distributed configurations
    - all-digital, partial HWIL, all HWIL
  - Multiple development and test labs
  - Software portability between Linux distributions
- **ARTEMIS operates in a HWIL environment such that a user can select between models of avionics components or interfaces to avionics hardware**
- **Simulation interacts with lab configuration and control software to support model selection and fault insertion**
- **Utilizes modern computing technology to achieve real-time performance of high-fidelity models**

# ARTEMIS Requirement Drivers

- **Model vehicles with high fidelity in real-time**
  - Low/High fidelity models of all avionics components
  - Low/High fidelity models of all subsystems that effect the vehicle or interact with avionics
  - Low/High-fidelity models of environment effects
  - Model dynamic effects (including flex)
- **Model all phases of mission (Prelaunch/Pad-ops through orbit insertion)**
- **Support of Multiple Labs & Configurations**
  - Software Development Facility (Flight SW Development)
  - System Integration Test Facility (Core Stage Avionics Test Lab)
  - System Integration Lab (Full-Scale SLS Avionics Test Lab)
  - Off-Site Emulators (KSC, MAF, SSC, etc.)
- **Support of Multiple Configurations & Scenarios for Each Lab**
  - HW under test can be swapped in and out and replaced with SW models without recompiling
  - Data files can be modified by "Scenario" files that contain specific items to be overwritten that initialize models to the proper state for each test

# ARTEMIS Requirements Drivers

- **Execute in multiple modes of operation**
  - Non-real-time, all simulated busses (i.e. run on a laptop)
  - Real-time, all simulated busses
  - Real-time, real busses/hardware
- **Interact with lab configuration and control software**
  - Model & configuration selection
  - Generation of scenarios (I.C. and fidelity options)
  - Real-time data viewing & fault insertion
- **Data Recording & Archiving**
  - Each HW bus must be recorded and archived
  - Simulation data must be recorded and archived
  - Metadata must be provided for all recorded data

# ARTEMIS Organization

- **ARTEMIS is organized into six functional components**
  - Simulation
    - Contains the executive framework
  - Synchronization & Timing
    - Scheduling, synchronization, global timing source, time stamps
  - Models
    - Three major categories: Core Simulation, Components, Subsystems
  - Input / Output
    - SCRAMNet, shared memory, discrete, analog, EIA-422, MIL-STD-1553B, Gigabit Ethernet
  - Data Recording
    - Global, local, meta data definition
  - Hardware
    - Computers, I/O cards, cables, racks

# ARTEMIS Organization

- **ARTEMIS is organized into six functional components**
  - **Simulation**
    - **Contains the executive framework**
  - Timing
    - Scheduling, synchronization, global timing source, time stamps
  - Models
    - Three major categories: Core Simulation, Components, Subsystems
  - Input / Output
    - SCRAMNet, shared memory, discrete, analog, EIA-422, MIL-STD-1553B, Gigabit Ethernet
  - Data Recording
    - Global, local, meta data definition
  - Hardware
    - Computers, I/O cards, cables, racks

# Simulation Overview

- **Executive Framework Consisting of:**
  - Input data processing of XML input files
  - Multi-phased initialization
  - Scheduled (run-time) loop
    - Derivative / Integration
  - Shutdown
  - Error handling
  - Monte Carlo
  - Fault Insertion

# ARTEMIS Organization

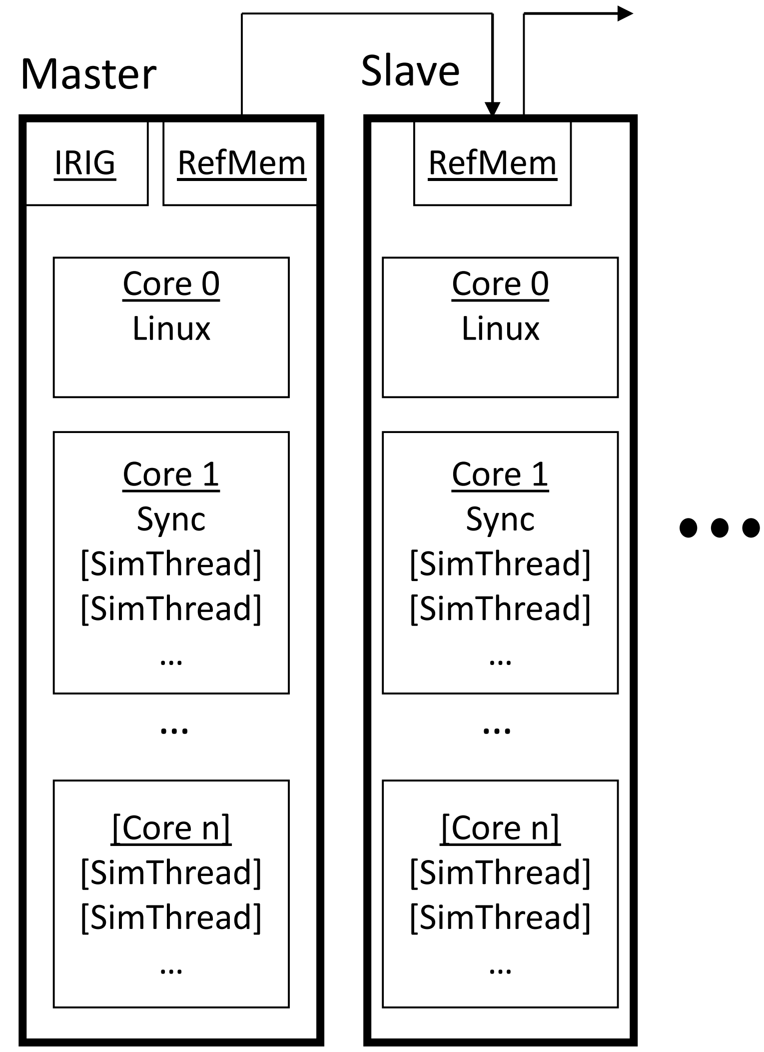
- **ARTEMIS is organized into six functional components**
  - Simulation
    - Contains the executive framework
  - **Timing**
    - **Scheduling, synchronization, global timing source, time stamps**
  - Models
    - Three major categories: Core Simulation, Components, Subsystems
  - Input / Output
    - SCRAMNet, shared memory, discrete, analog, EIA-422, MIL-STD-1553B, Gigabit Ethernet
  - Data Recording
    - Global, local, meta data definition
  - Hardware
    - Computers, I/O cards, cables, racks

# Timing Overview

- **Sync controls timing and scheduling of frames for each ARTEMIS executable**
  - ARTEMIS executables may run at different frame rates that are a multiple of Sync minor frame rate
- **Maintains hard real-time operation using a timing card such as IRIG-B or RCIM**
  - Can also run non-real-time
- **Creates and controls access to the shared/reflective memory region for ARTEMIS**
- **Receives and responds to commands from MAESTRO for both Master and Slave Sync**
  - MAESTRO passes test configuration and startup commands through Master Sync
  - MAESTRO issues Sync commands to control ARTEMIS execution
  - Sync responds to MAESTRO with status messages

# Sync Architecture

- **One Master Sync process runs on the Master Node**
- **Each additional simulation node runs the Slave Sync process that is controlled by Master Sync**
- **Master node controls real time synchronization via reflective memory**
  - Receives timer interrupt from timing card
- **Sync Data Coherence**
  - Data input at beginning of sim thread's start cycle
  - Data output at end of cycle prior to sim thread's next start cycle

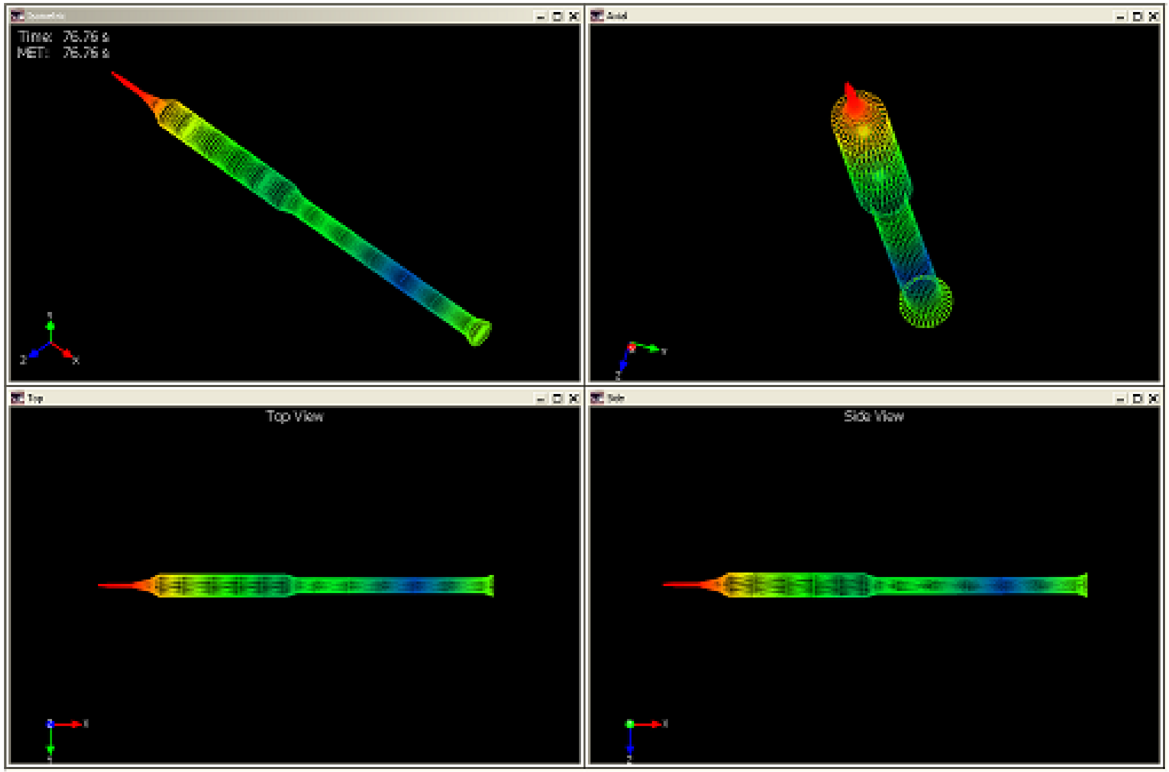


# ARTEMIS Organization

- **ARTEMIS is organized into six functional components**
  - Simulation
    - Contains the executive framework
  - Timing
    - Scheduling, synchronization, global timing source, time stamps
  - **Models**
    - **Three major categories: Core Simulation, Components, Subsystems**
  - Input / Output
    - SCRAMNet, shared memory, discrete, analog, EIA-422, MIL-STD-1553B, Gigabit Ethernet
  - Data Recording
    - Global, local, meta data definition
  - Hardware
    - Computers, I/O cards, cables, racks

# Models Overview

- **Core Simulation**
  - Flexible and rigid body equations of motion and environment models
- **Component**
  - Digital models representing the functionality of actual Ares avionics boxes




- Subsystem
  - Digital, physics-based models representing the vehicle's physical subsystems that are not typically tested in the lab.

# ARTEMIS Organization

- **ARTEMIS is organized into six functional components**
  - Simulation
    - Contains the executive framework
  - Timing
    - Scheduling, synchronization, global timing source, time stamps
  - Models
    - Three major categories: Core Simulation, Components, Subsystems
  - **Input / Output**
    - **SCRAMNet, shared memory, discrete, analog, EIA-422, MIL-STD-1553B, Gigabit Ethernet**
  - Data Recording
    - Global, local, meta data definition
  - Hardware
    - Computers, I/O cards, cables, racks

# I/O Layer Overview

- **Provides a transparent, consistent architecture for performing I/O for the ARTEMIS models**
- **Handles simulated device communication between the models via either shared memory or SCRAMNet reflective memory**
- **Transfers to real or simulated devices must be transparent to the models**
- **Handles the following real devices contained in the Ares I avionics architecture:**
  - MIL-STD-1553B, EIA-422, Discrete I/O, Analog Sensors, D/A and A/D, Gigabit Ethernet
- **Handles other real devices needed by the simulation system such as:**
  - GPIB, RCIM II / RCIM III, SCRAMNet GT, IRIG

# I/O Layer Overview

- **The I/O Layer consists of:**
  - A set of common library calls that the ARTEMIS models use for communication with the I/O Layer
  - The I/O Layer process which performs all the I/O with real or simulated devices
  - An XML file describing the configuration of the Ares I avionics rings, simulation computers, and I/O devices used during a simulation
  - A python based GUI that allows a user to build the XML configuration file
  - An I/O Layer library:
    - Contains the initialization, read, write and close calls for each device the models control
    - Communicates with the I/O Layer process via shared memory semaphores
    - Passes unique device information and data from the models to the I/O Layer process via device structures in shared memory
    - The read and write calls communicate directly with the device driver threads

# ARTEMIS Organization

- **ARTEMIS is organized into six functional components**
  - Simulation
    - Contains the executive framework
  - Timing
    - Scheduling, synchronization, global timing source, time stamps
  - Models
    - Three major categories: Core Simulation, Components, Subsystems
  - Input / Output
    - SCRAMNet, shared memory, discrete, analog, EIA-422, MIL-STD-1553B, Gigabit Ethernet
  - **Data Recording**
    - **Global, local, meta data definition**
  - Hardware
    - Computers, I/O cards, cables, racks

# Data Recorder Overview

- **Data Recorder supports generic data recording of multiple types of interfaces:**
  - SCRAMNet, MIL-STD-1553, Gigabit Ethernet, EIA-422, Discrete I/O, Cross Channel Data Link (CCDL)
- **Configured via an XML file**
- **Data is recorded in its raw format**
  - Each packet/message is recorded with a timestamp
- **Each interface is recorded in a separate file**
  - Filenames contain the beginning and ending timestamp for its corresponding data
- **Interfaces with the local MAESTRO daemon**
- **Provides periodic archiving capability for early analysis during long tests**

# ARTEMIS Organization

- **ARTEMIS is organized into six functional components**
  - Simulation
    - Contains the executive framework
  - Timing
    - Scheduling, synchronization, global timing source, time stamps
  - Models
    - Three major categories: Core Simulation, Components, Subsystems
  - Input / Output
    - SCRAMNet, shared memory, discrete, analog, EIA-422, MIL-STD-1553B, Gigabit Ethernet
  - Data Recording
    - Global, local, meta data definition
  - **Hardware**
    - **Computers, I/O cards, cables, racks**

# Hardware Overview

- **Concurrent RedHawk real-time operating system**
  - Devices verified by vendor to meet real-time requirements
- **I/O Cards**
  - SCRAMNet, MIL-STD-1553, Gigabit Ethernet, EIA-422, Discrete I/O, Analog Sensors, D/A and A/D boards, RCIM, IRIG
- **The SIL will have flight-like cables**
- **All simulated components will be positioned in computer racks near avionics boxes in each ring**

# MAESTRO Introduction

- **What is MAESTRO?**
  - **MAESTRO** stands for **M**anaged **A**utomation **E**nvironment for **S**imulation, **T**est, and **R**eal-time **O**perations.
  - In a nutshell, MAESTRO is an automation, configuration, and orchestration software framework.
- **Who developed it?**
  - Developed by ES53 Avionics and Software Ground Systems Test Branch.
  - Class D Software.
- **Why?**
  - To serve as the lab automation and configuration software for the Integrated Avionics Test Facilities.
  - Allows users to configure real and simulated, execute faults and events, monitor, and analyze integrated avionics tests.

# What MAESTRO does:

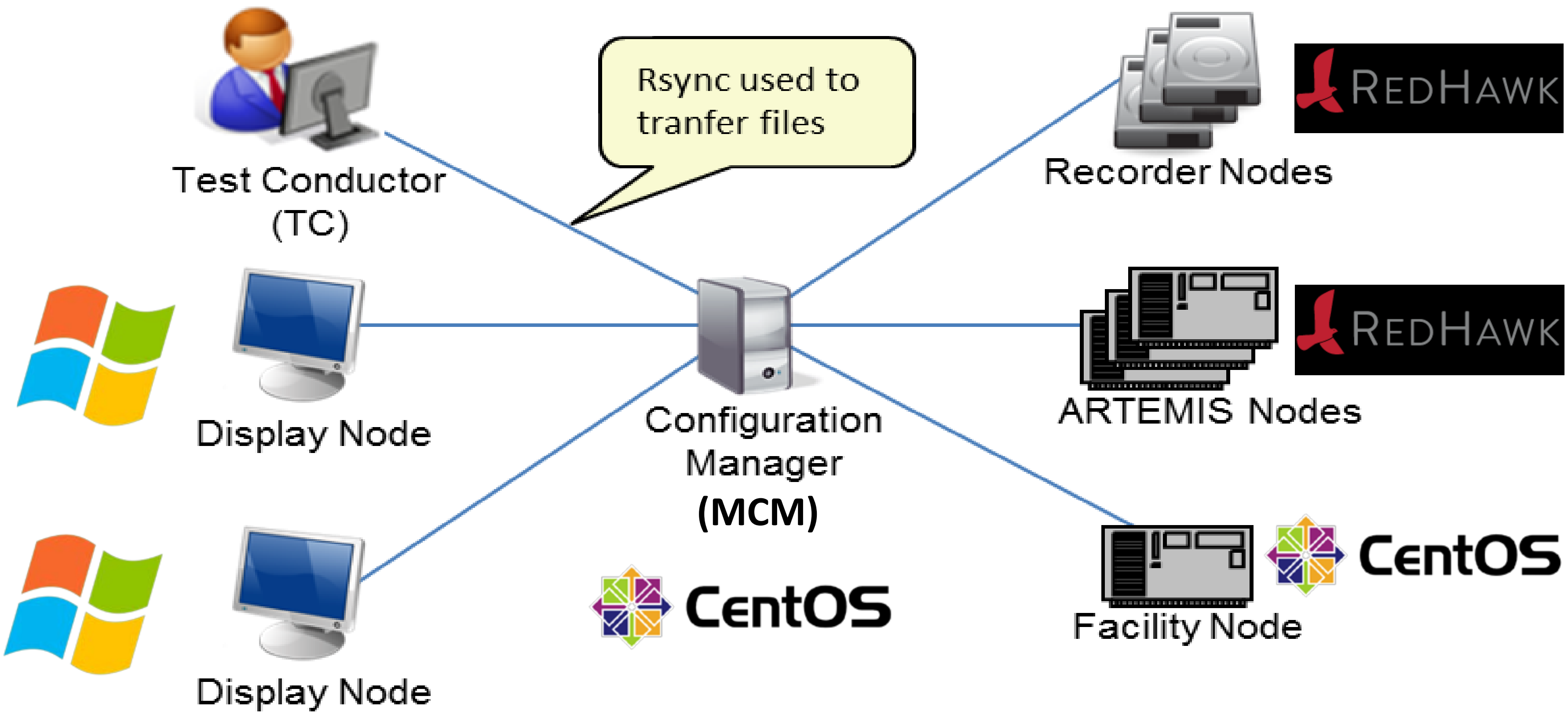
- Orchestrate the test (configure, start, stop, clean up).
- Configure, launch, interface to, and monitor ARTEMIS
- Configure lab-specific equipment based on test configuration.
- Display run-time test and facility data
- Archive test artifacts
- Perform run-time and post-test data analysis

# Key Features

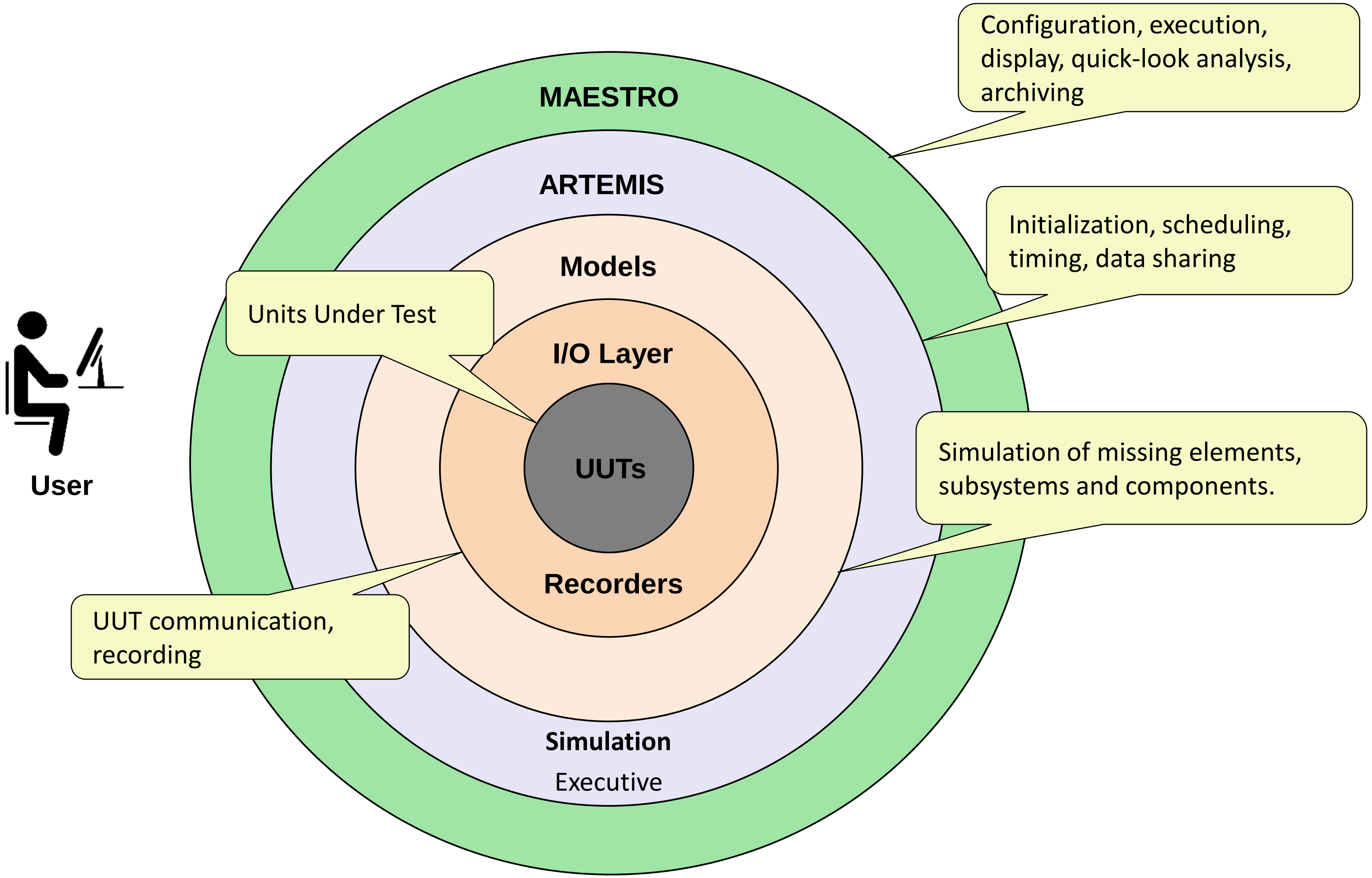
- **Distributed, asynchronous test control**
  - Distributed : The MAESTRO application runs distributed across the multiple computers in the lab, allowing it to scale to support multiple lab configurations
- **Allows for custom user-definable configuration for UUTs**
- **Highly reconfigurable lab supported by XML based configuration management and user definable node types**
- **Publish/subscribe data distribution including runtime distribution of metadata**
- **Supports Linux and Windows operating systems**
- **Script driven run-time and post test data analysis capabilities**

# MAESTRO typical operating environment

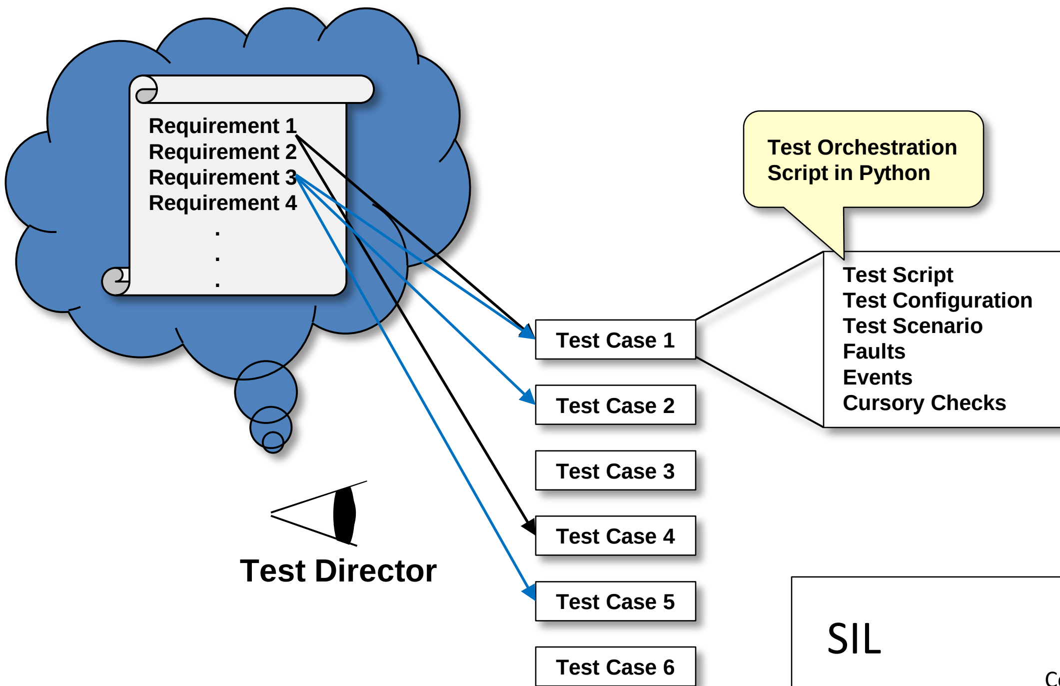
- **MAESTRO synchronizes the computers involved, configures what is real or simulated hardware in the loop, runs the test, archives the data, and shuts down necessary systems.**



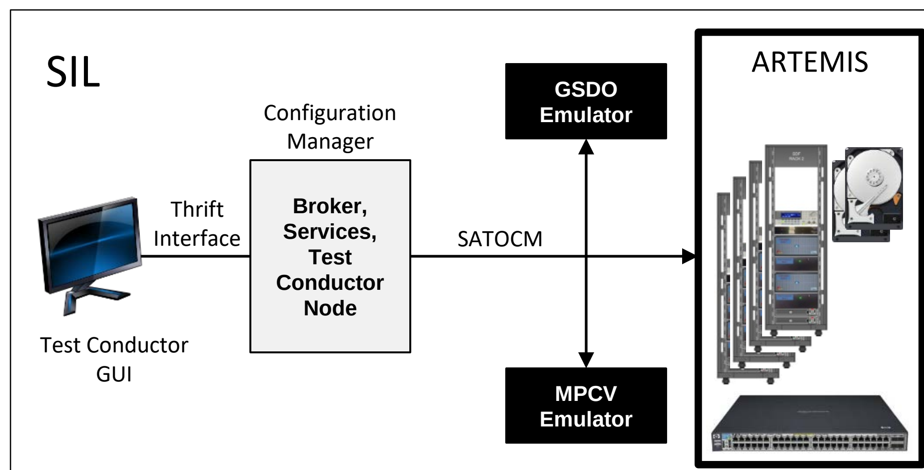
# The Big Picture – Layer View



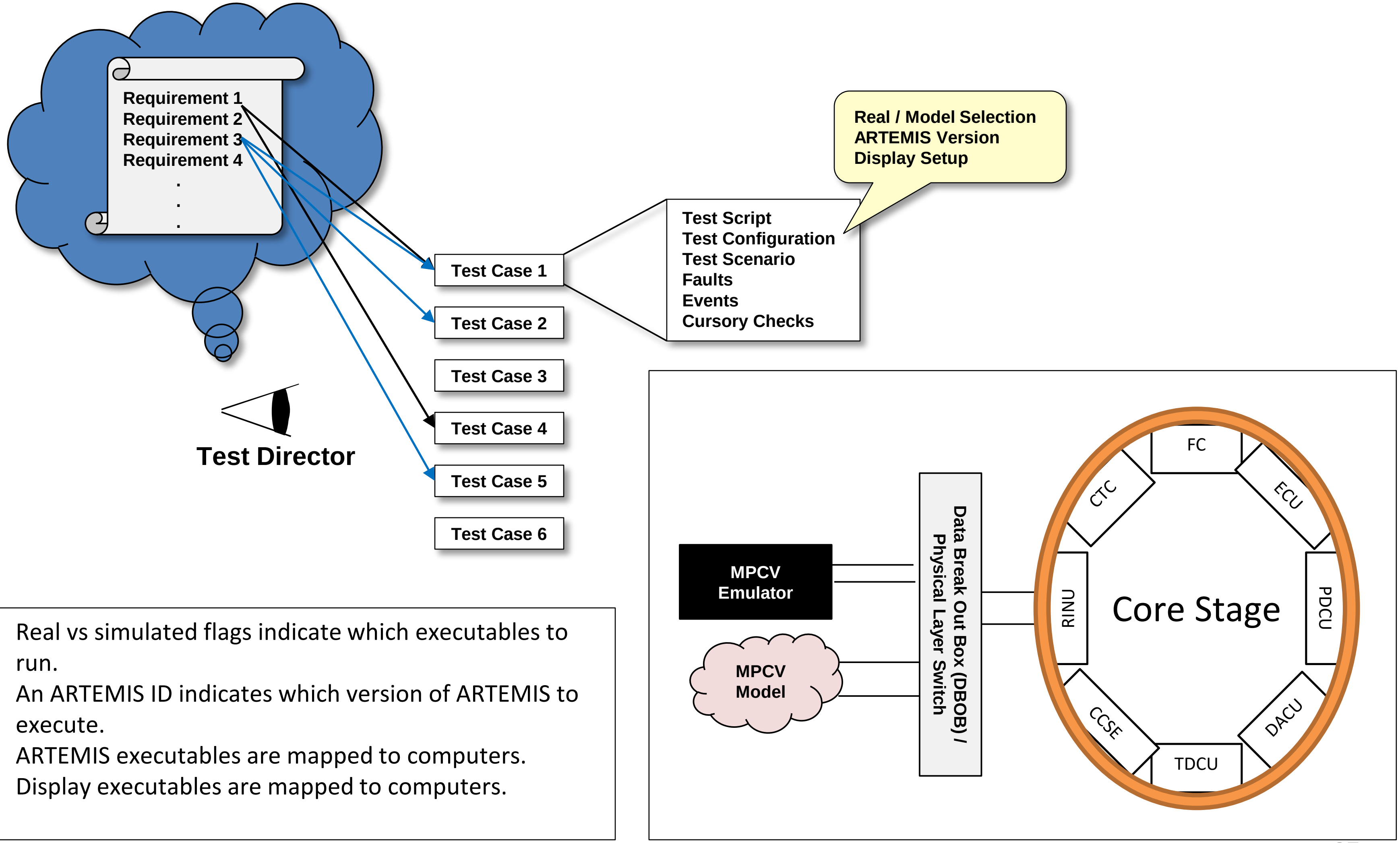
# Test Case Example




- A single test script will cover the majority of test cases
- The test script runs during test execution beginning with the initialization command through the stop test command.
- Lab configuration, archiving, and post test analysis are handled outside of the test script.
- MAESTRO implements the SATOCM spec for communication with emulators.



# Test Case – Test Configuration




- Real vs simulated flags indicate which executables to run.
- An ARTEMIS ID indicates which version of ARTEMIS to execute.
- ARTEMIS executables are mapped to computers.
- Display executables are mapped to computers.

# Backup Slides

# ARTEMIS Customers

- **SDF – Software Development Facilities (4487, 4436)**
  - SDF1 – Early integration test lab with FSW
  - SDF2 – Lab for development and test of unreleased FSW
  - SDF3 – Lab used for formal testing of released FSW
  - SDF4 – Lab used for formal testing of released FSW
- **SITF – Software Integration Test Facility (4205)**
  - SITF-Q - Core stage avionics test – formal qualification
- **SIL – System Integration Lab (4205)**
  - Core stage & booster avionics test
- **TDL – Tester Development Lines (4476)**
  - SDF & SITF configurations used to develop test cases and support V&V of ARTEMIS
- **RINU 6DOF Test (4663)**
  - HW Testing of RINU avionics using 6DOF platform driven by ARTEMIS

# ARTEMIS Customers - Emulators

- **GSDO – Ground Systems Development and Operations (KSC)**
  - SHADE – SLS High Fidelity emulator for Ground Systems development and testing
- **MPCV – Multi-purpose Crew Vehicle (JSC, LMCO)**
  - VTB –MPCV avionics test & development
  - ITL - MPCV/ICPS test and development
  - MS – Mission Systems
- **Green Run (SSC)**
  - SLS & Test Stand models used to support hot-fire testing of SLS & FSW
- **Booster Hardware-in-the-Loop (HIL) (4205)**
  - Booster avionics interfaces to the Booster subsystems and Core Stage

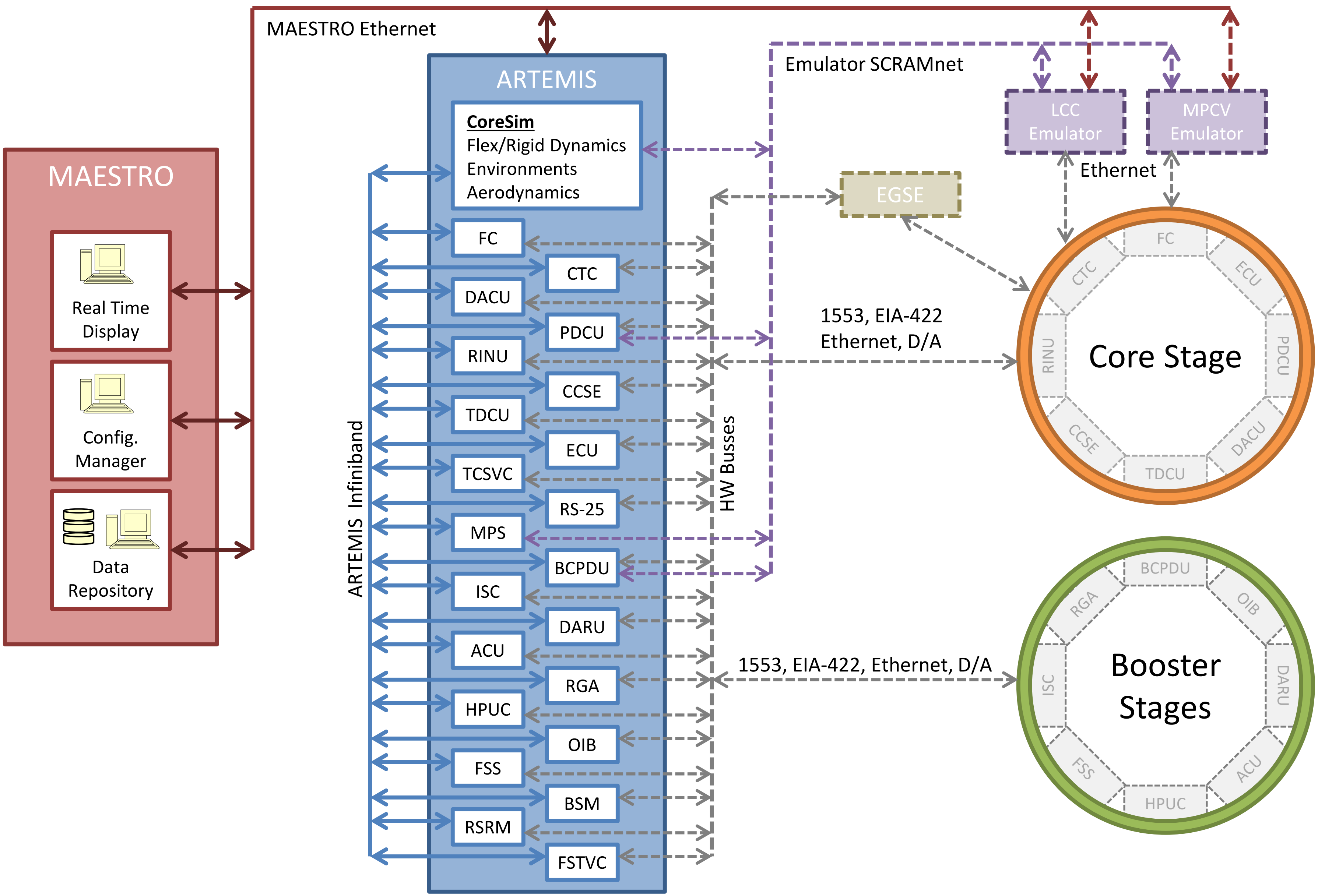
# ARTEMIS Development Lines

- **Software Development Lab (4476/110A)**
  - SDF Configuration
  - SITF Configuration
  - ADSB – Single box configuration
- **“A” Nodes (4476/200)**
  - SITF Configuration – Internal use only to test new configurations, HW, OS updates & Software
  - SDF Configuration in development (“S” Nodes, Location TBD)
- **Emulator Development Lines (4476/113)**
  - Separate lines for each emulator used to test new updates and configurations
- **Emulator Test Lines (4476/113)**
  - Separate lines for each emulator used for debugging delivered configurations – Used in lieu of on-site debugging
- **Test Nodes (4476/114)**
  - Used for testing long runs, misc. debug
- **Control Room (4476/100)**
  - Multi-monitor consoles for running all development/test lines to be installed soon

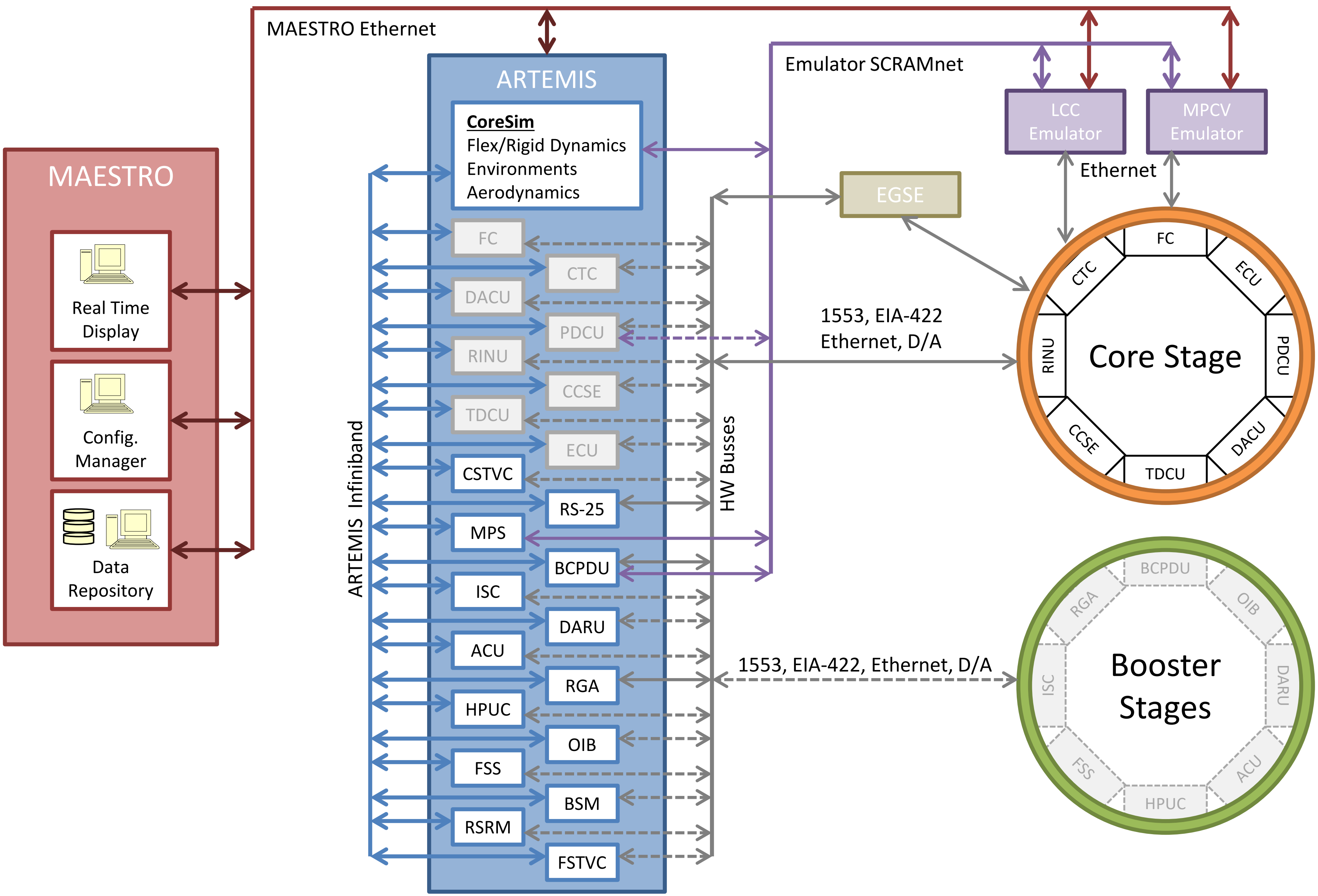
# ARTEMIS Simulation Design

- **Separate executables for each model**
  - Allows "plug-in operation" for switching model with LRU hardware
  - Allows models to be distributed for faster real-time operation
  - Allows models to be hosted on a simulation node located in correct physical location to use correct cable lengths
- **Single copy of source code for all simulation configurations**
  - Hardware interfaces defined by configuration files
  - Models (executables) are enabled based on configuration generated by MAESTRO
  - Models utilize XML input files to load initial conditions and configuration options
- **All models utilize the same libraries to allow for ease of incorporation & maintenance**
  - Real-time control
  - Inter-model communication
  - Hardware communication
  - Input file parsing, data recording and utility functions
- **Vehicle configuration logic and dynamics engine are generic to support changes in vehicle architecture**
- **Shared Memory region used to transfer data between models**
  - Blackboard structure defined to include all data shared between models
  - Each model "owns" a particular contiguous section of blackboard
  - Infiniband is used to copy blackboard to each machine at start of each frame
  - Models update blackboard section at end of each frame
- **Fault insertion engine can override real-time inputs to models**
  - C code-based

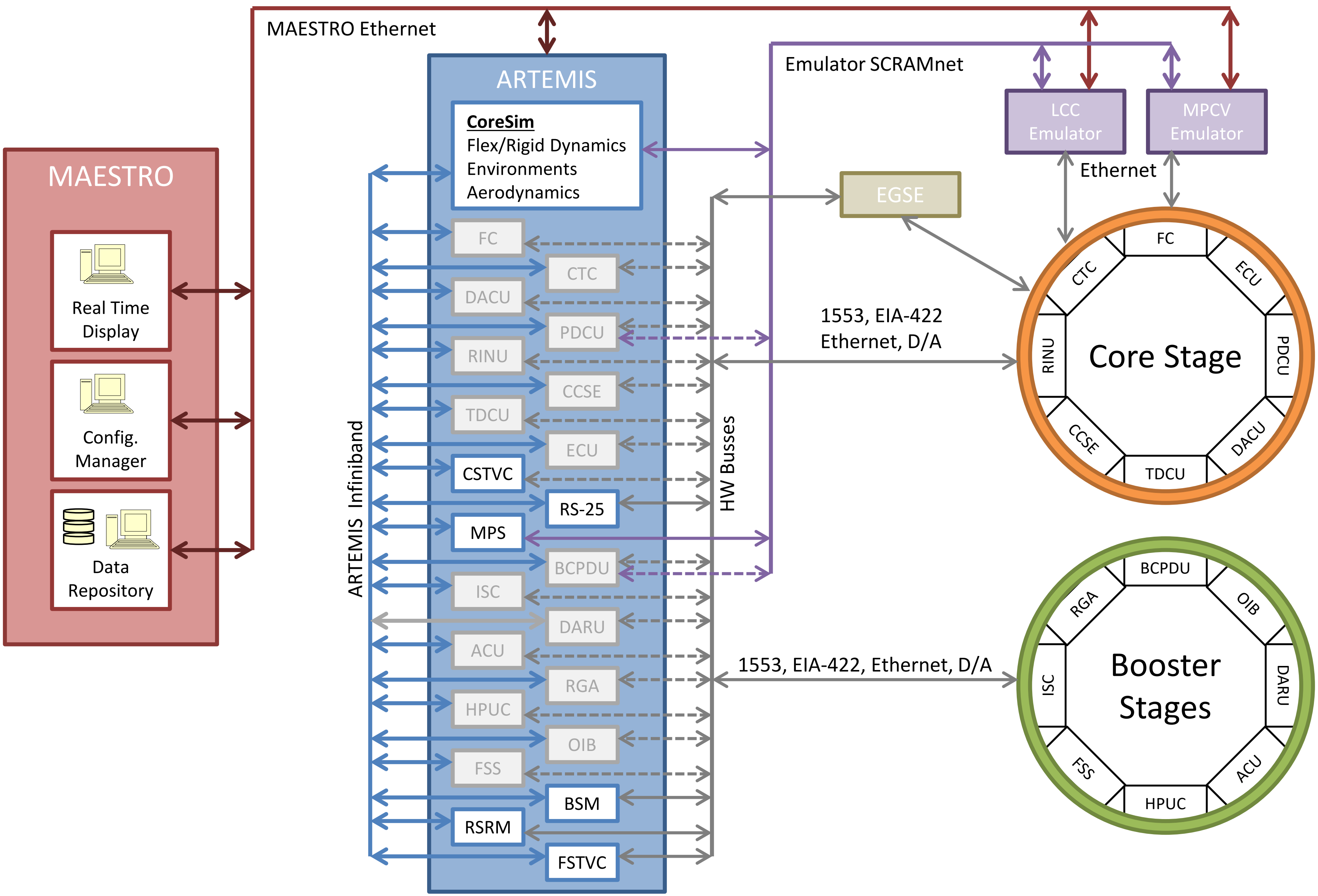
# All-Digital Lab Configuration



# SITF Lab Configuration



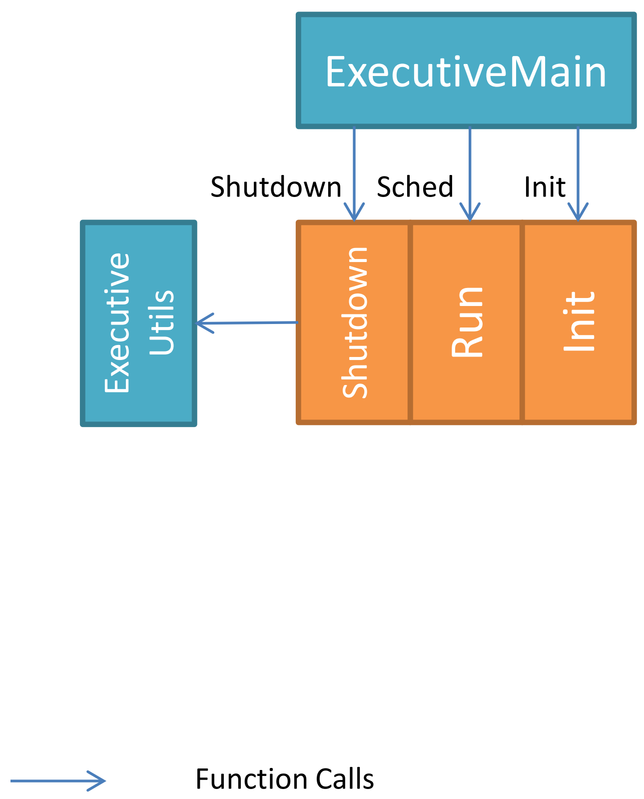
# SIL Lab Configuration



# ARTEMIS Software Design

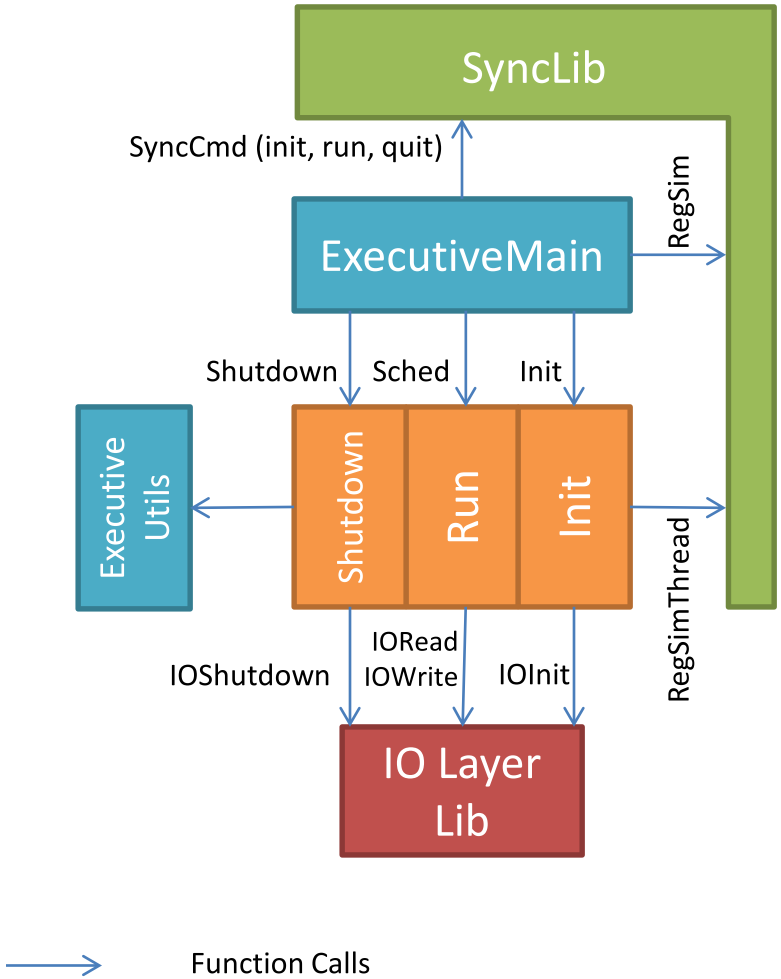
- **ARTEMIS Composed of 5 Modules**
  - Simulation
    - Base framework – Provides 'main' for each model executable
    - Input file processing
    - Data recording
  - Timing
    - Interacts with RCIM timing card
    - Keeps all executables on each node in lock-step
    - Manages Infiniband and blackboard shared memory on each machine
  - Input/Output Layer
    - Provides common, user-level interface to hardware
  - Models
    - CoreSim – Dynamics, environments & vehicle configurations
    - Subsystems – Models effectors and sensors, interfaces between CoreSim and Components
    - Components – Models avionic/electronic systems
  - Data Recorder
    - Records all HW bus data, sends data to remote archive
  - Hardware
    - Computers, I/O cards, cables, racks

# Architecture of a Single Model Executable



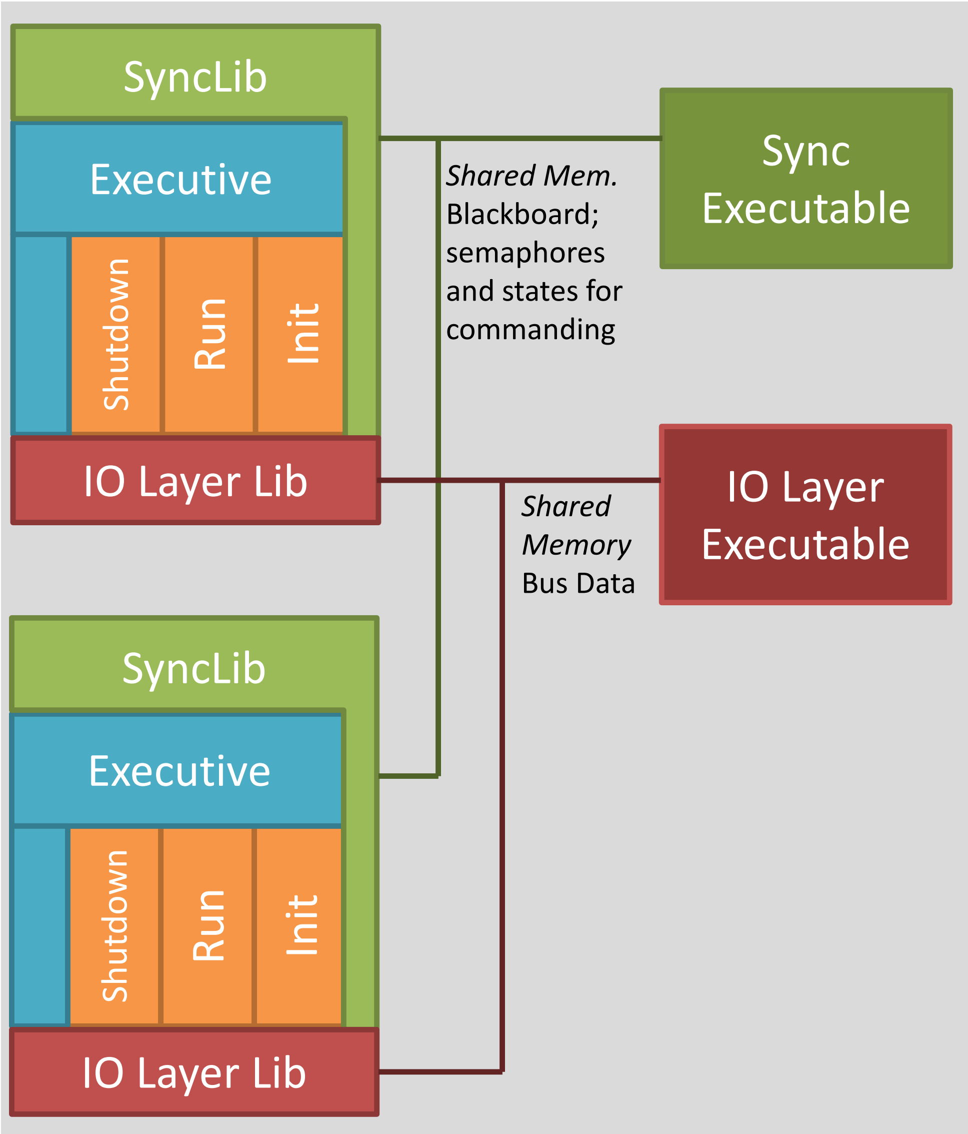

- **Each instance of a subsystem or avionics component is considered a separate model**
  - Each model runs in a separate executable to meet requirements for re-configurability (i.e. plug-and-play behavior)
- **Models contain code to simulate behavior of sensors, effectors, avionics components & dynamics**
  - Code for initialization
  - Code performed each time step of the simulation run
  - Code to be performed at shutdown
- **The Executive Main provides standard set of wrapper function prototypes for each of these simulation phases**
  - Models are responsible with populating these functions with code relevant to the simulation phase
  - Executive calls the wrapper functions, which then call model code
- **Executive also provides common set of utilities for input processing, data recording, mathematics and threading**

# Architecture of a Single Model Executable



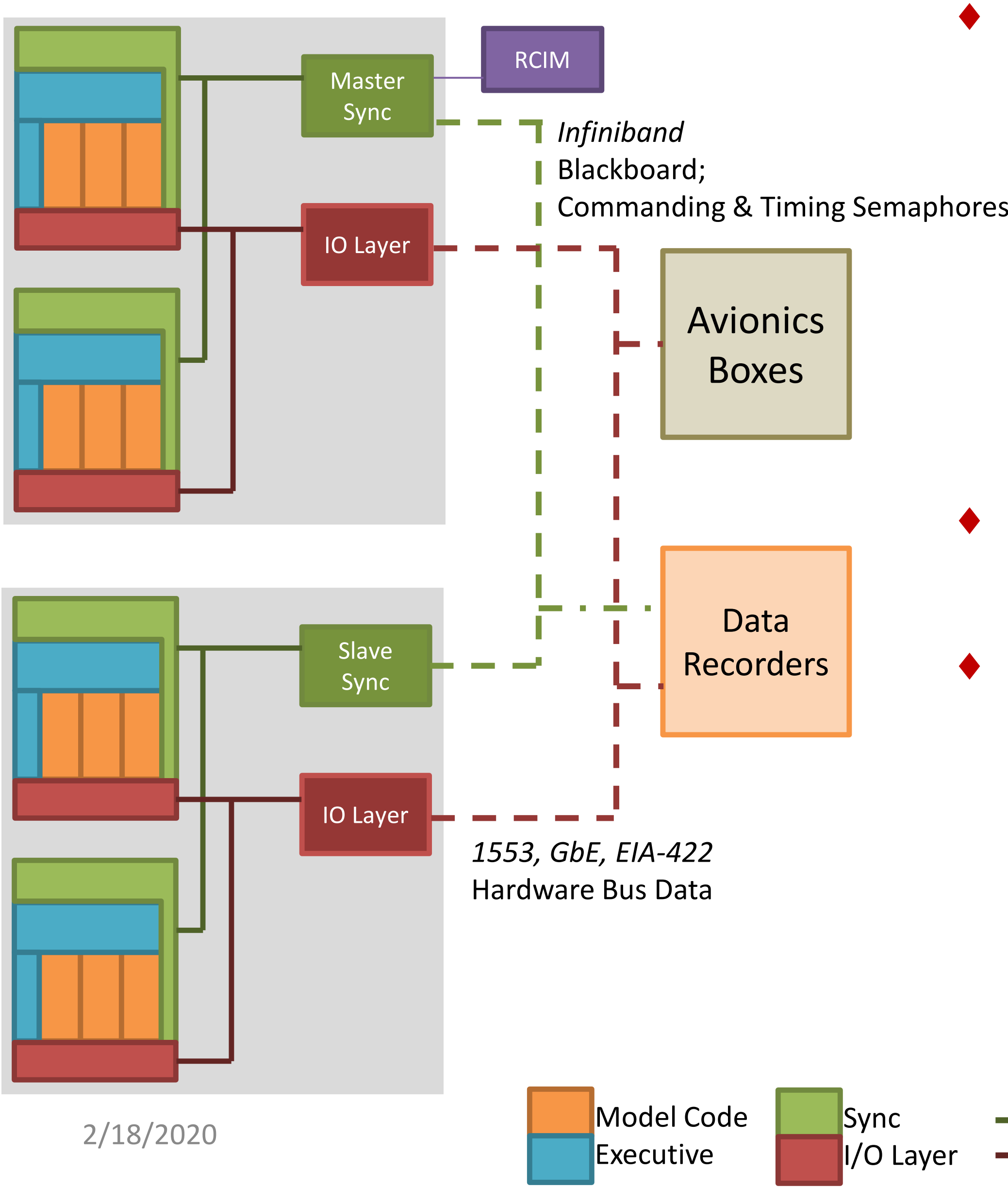

- **SyncLib is responsible for commanding the executive to call the init, run and shutdown functions based on clock timing and user input (SyncCmd)**
  - Executive makes function call to Sync to get current command
- **Executive registers itself with Sync to gain access to Sync shared memory area (RegSim)**
- **Models register with Sync a structure containing all the data to output to other models, as well as the data requested to be read in from other models (RegSimThread)**
- **IOLayerLib provides functions to models to read and write to simulated or real hardware devices**
- **IOLayerLib also provides functions to initialize and safely shutdown devices**

# Single Computer Configuration



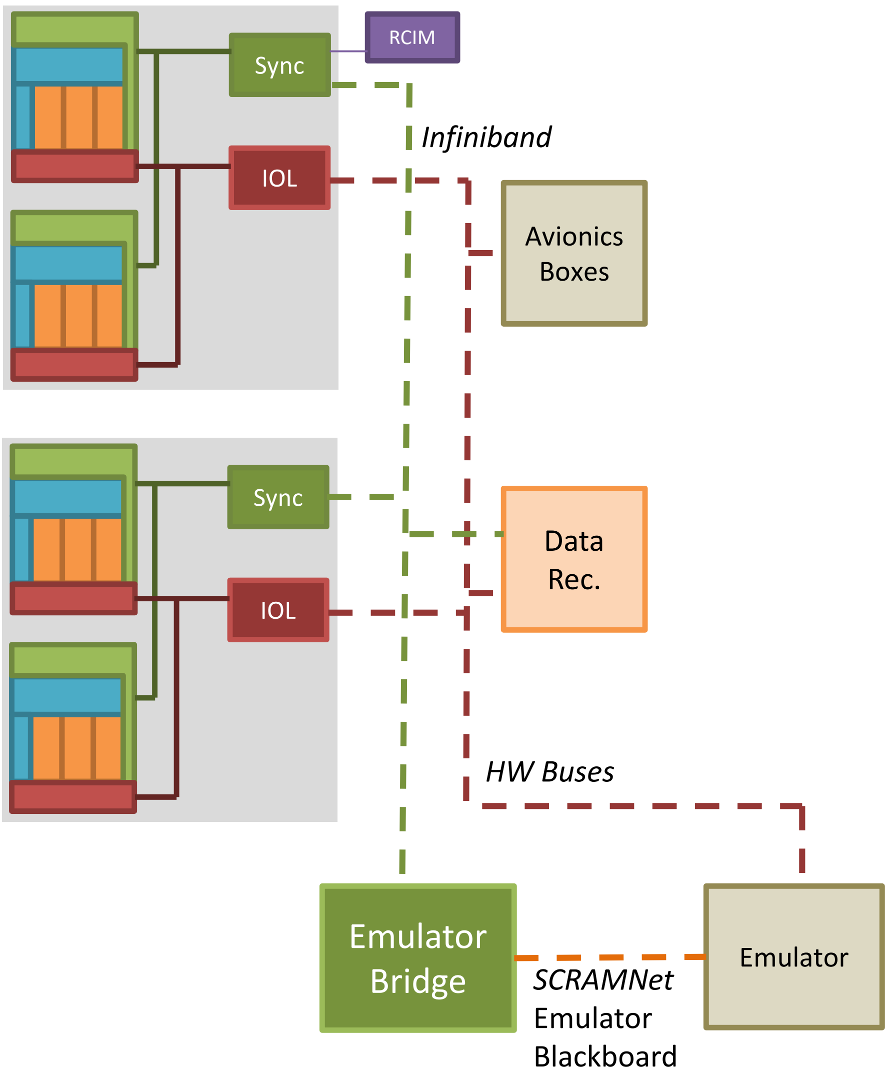

- **Sync Executable is responsible for timing (real-time) and getting current commands from user interface, as well as opening up shared memory region**
- **I/O Layer Executable provides communication with device drivers**
- **On each machine, multiple models may run, but there will only be one Sync Executable and one I/O Layer Executable**
- **The Sync & I/O Layer Executables communicate with the model executables using POSIX shared memory regions**
- **Sync shared memory region**
  - Blackboard
    - Memory region defined by a group of structures (one per model).
    - Each executable writes output data to an assigned area
    - Other model's area can be input using Synclib
  - Semaphores and data used for commanding
- **I/O Layer shared memory region**
  - Buffers populated by model with bus data intended to be sent to device driver
  - Buffers populated by IO Layer executable with bus data received from device drivers

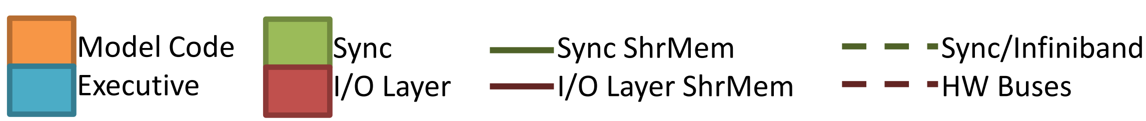
# Multi-Computer Configuration




- **Each machine has a separate Sync and I/O Layer Executables.**
  - Only one machine runs Master Sync
    - Interfaces with External Timing card in addition to functions performed by slave sync
  - All other machines hosting ARTEMIS executables run slave sync
  - Synchronizes shared memory of computer with all other computers via Infiniband
  - Communicates with user interface for commanding
- **Hardware avionics boxes may or may not be present, and can communicate directly with 1553, Ethernet, 422, etc.**
- **Data Recorders tap off of each bus, as well as off the Infiniband network (only for Blackboard data) to capture and archive data each frame**

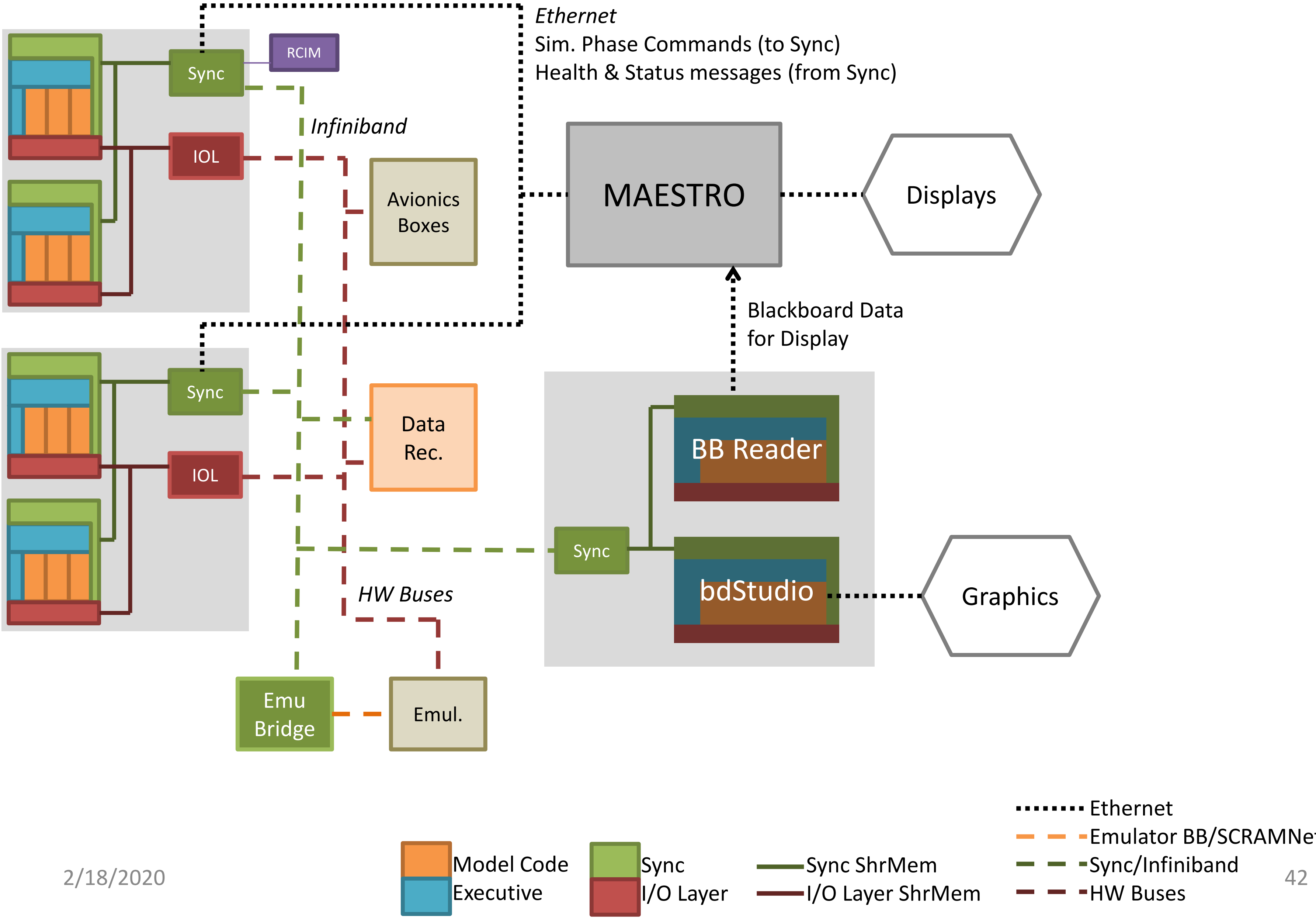
# Multi-Computer Configuration w/ Emulators




- **Emulators provide ability to communicate with external systems not modeled within ARTEMIS**
- **Emulator Bridge is a standardized interface used to communicate simulation data between ARTEMIS & Emulators**
  - Emulator Bridge is considered a component of Sync, but runs as separate executables
  - Emulator bridge communicates among Emulators and ARTEMIS using SCRAMNet interface with a separate Blackboard
  - ARTEMIS Client Maps Emulator Blackboard data to ARTEMIS Blackboard Data

# Multi-Computer Configuration w/ MAESTRO



# Multi-Computer Configuration w/ MAESTRO

- MAESTRO Communicates with the Sync executables on each node to command the simulation state
- Sync reports health & status and error messages back to MAESTRO
- In order to drive displays, the BBReader executable is used alongside other ARTEMIS executables to read the blackboard and send data to MAESTRO via Ethernet sockets
- For the animation, a bdStudio executable is used to read from the blackboard and send data to the animation tool
- BBReader and bdStudio both utilize the ARTEMIS Executive and must run on a computer with Sync
- MAESTRO generates configuration files specifying run details for use by ARTEMIS and pushes these files to a pre-determined location
- MAESTRO reads console output from each model and filters out messages intended to be relayed to user

# Fault Insertion

## Two Types of Faults

- Overwrite exposed simulation variables in SCRAMNet
  - Least expensive to implement
  - Limited to exposed simulation variables
  - Won’t cover all fault requirements
- Execute an embedded fault in the simulation
  - Require additional software development and V&V in models to simulate fault
  - Initiated by tripping fault flag in SCRAMNet
  - Need to streamline number of embedded faults

## Fault Insertion Mechanism

- Peek and Poke via MAESTRO
  - User can trip embedded faults via MAESTRO interface to sync
  - Non-Deterministic
- Separate fault insertion executable
  - Controlled by sync master
  - Provides logical conditions to determine when fault inserted
  - Input file driven
  - Deterministic

# Core Simulation Models

## Stack Dynamics

- Coupled rigid body and flexible body dynamics formulation which properly accounts for variable mass effects and force following terms
- Supports all nominal and abort configurations
- Input data developed from EV30 LA2 structural models
- Multithreaded partitioned equations to achieve real-time performance with a frame time under 2ms

## Stage Dynamics

- 6 DOF rigid body formulation with vehicle states defined with respect to Constellation structural frame (fixed point off nose of LAS)
- Supports all nominal and abort configurations

## Mass Properties

- Propellant mass computed using mass flow rate defined by engine model
- Propellant mass properties  computed from structural model mass matrices
- Compute mass properties of each stage from sum of dry structure and propellant
- Mass properties of stack (or combined stages) computed from sum of stages for current configuration defined by flight phase

# Core Simulation Models

## Structural Properties

- Stage mass and stiffness matrices defined by NASTRAN models
- Family of propellant mass matrices based on stage mass
- Assemble stack mass and stiffness matrices from stage and propellant matrices based on vehicle configuration
- Update generalized vehicle mass and stiffness matrices each time step for coupled flex body EOM
- All vehicle node geometry extracted from integrated NASTRAN model

## Nozzle Dynamics

- Rigid body formulation uses discrete nozzle EOM driven by vehicle dynamics, TVC actuator forces, aerodynamic forces, and flex bearing stiffness
- Rigid body formulation also includes Tail-Wag-Dog effects
- Flex body formulation utilizes coupled nozzle dynamics embedded in system Ritz vectors or modes

## Slosh Dynamics

- Rigid body formulation uses discrete slosh masses per tank modeled by spring-mass-damper systems, Lookup tables for slosh parameters
- Flex body formulation utilizes slosh modes developed from additional effects superimposed on propellant mass and stiffness matrices

# Core Simulation Models

## ♦ Atmosphere and Winds

- US76 standard atmosphere model
- 2007 Global Reference Atmospheric Model (GRAM2007)
- 1800 Measured Day-of-Launch Winds
- Ground winds to support pre-launch

## ♦ Lumped Aerodynamics

- Linear 1-D table lookup and Nonlinear 2-D table lookup for aerodynamic coefficients for stack and stages (SRB, LAS, etc)

## ♦ Distributed Aerodynamics

- Aerodynamic data mapped to NASTRAN mesh for loads applied to the stack (primary driver of flex)

## ♦ Gravity

- 3 model options:
  - Kepler
  - J-2, J-3, & J-4
  - Gravity Recovery and Climate Experiment (GRACE)

# Component Models

## ♦ Flight Computer (FC)

- Controller algorithm
  - Exact representation of DAC2 Ares Controller algorithms (Gain-scheduled Flex Mitigation Filters + PID)
- Navigation algorithm
  - Fundamental Navigation Equations for multiple sensors and rate gyros
- Guidance algorithm
  - Exact representation of DAC2 Ares Guidance algorithms (Open-Loop Profile for 1st Stage; Closed-Loop Algorithm for US)
- Mission Manager and Event Controller
  - Event handler to control flight and vehicle phasing based on flight time and mission events

## ♦ Booster Control & Power Distribution Unit (BCPDU)

- Passes commands from the flight computer to downstream avionics boxes
- Prototype MIL-STD-1553 interface from FC with TVC commanded rock and tilt current message

## ♦ Sensors

- Medium-fidelity RINU model with gyroscope and accelerometer error terms (bias, noise, scale factor, misalignments, initial condition errors)

# Component Models

## Recovery Control Unit (RCU)

- Commands the BTM, aeroshell jettison, and forward skirt extension jettison on the first stage during recovery operations

## Ignition & Staging Controller (ISC)

- Commands the firing of the first stage, BDM, USM, and first stage separation pyros based on commands from the BCPDU

## Altitude Sensor Assembly (ASA)

- Pressure sensor that activates first stage recovery system once the SRB falls below a given altitude

## Command and Telemetry Computer (CTC)

- Currently relays ground commands to the FC during pre-launch and ascent

## Rate Gyro Assembly Electronics (RGAE)

- Buffers the RGA outputs for use in the FC for both the first stage and upper stage RGAs

## Redundant Inertial Navigation Unit Electronics (RINUE)

- Uses $\Delta V$ & $\Delta\theta$ from the RINU to estimate vehicle states & other data needed by the flight software

# Component Models

## Combined Control System Electronics (CCSE)

- Partial CCSE model outputs valve commands to support tanking ground ops. Incorporates previous ReCSE model as well.

## Roll Control System Electronics (RoCSE)

- Relays the fire commands for the first stage roll control system from the FC to the RoCS thrusters

## Upper Stage Engine Control Unit (US ECU)

- Controls the J-2X firing, mixture ratio, and throttle

## Upper Stage TVC Data & Control Unit (US TVC DCU)

- Converts a commanded set of gimbal angles from the FC into a current value used by the upper stage TVC

# Subsystem Models

- **Reaction Control System (RCS)**
  - Ideal thrust, general valve dynamics developed but not activated
  - Lookup tables for thrust & valve dynamics
- **Booster Separation Motors (BDM, BTM, Ullage)**
  - Uses lookup table for thrust, supports delayed firing
- **Engines**
  - Lookup table driven, supports separate tables for nominal, startup and shutdown operations
- **Thrust Vector Control (US & FS TVC)**
  - High-fidelity simplex algorithm with models of servo valves, power spool, and actuator
- **Main Propulsion System (MPS)**
  - Simple tanking model
  - High fidelity model incorporated using existing ROCETS code

# Current Subsystem Models

- **Hold-Down Post (HDP)**
  - Uses stiffness and damping matrix to model flexibility of launch platform
  - Spring can only provide force while in compression
- **Linear Shaped Charges (LSC)**
  - Model does not provide forces, but sets flags indication whether stage separation has occurred
- **Redundant Inertial Navigation Unit (RINU)**
  - Converts sensed vehicle motion signals into $\Delta V$ & $\Delta\theta$ values needed by the flight software
- **Rate Gyro Assembly (RGA)**
  - Senses the vehicle motion and converts to $\Delta\theta$ signals used by the FC controller

# MAESTRO - Components

- **Core Services** – Communications framework, XML Readers, transfer protocols.
- **Test Control** – Scripting, command implementation.
- **Test Monitoring** – Run-time data collection, distribution, and processing.
- **Data Analysis** – Run time and Quick-look tools for recorded data.
- **GUIs** – User interfaces.