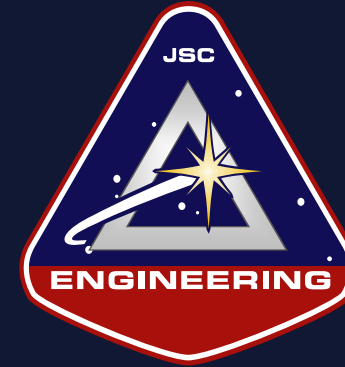




Trustworthy Autonomy for Gateway Vehicle Systems Manager

Jim Dabney, PhD; **Julia Badger, PhD**; Pavan Rajagopal
University of Houston – Clear Lake; NASA-Johnson Space Center;
CACI



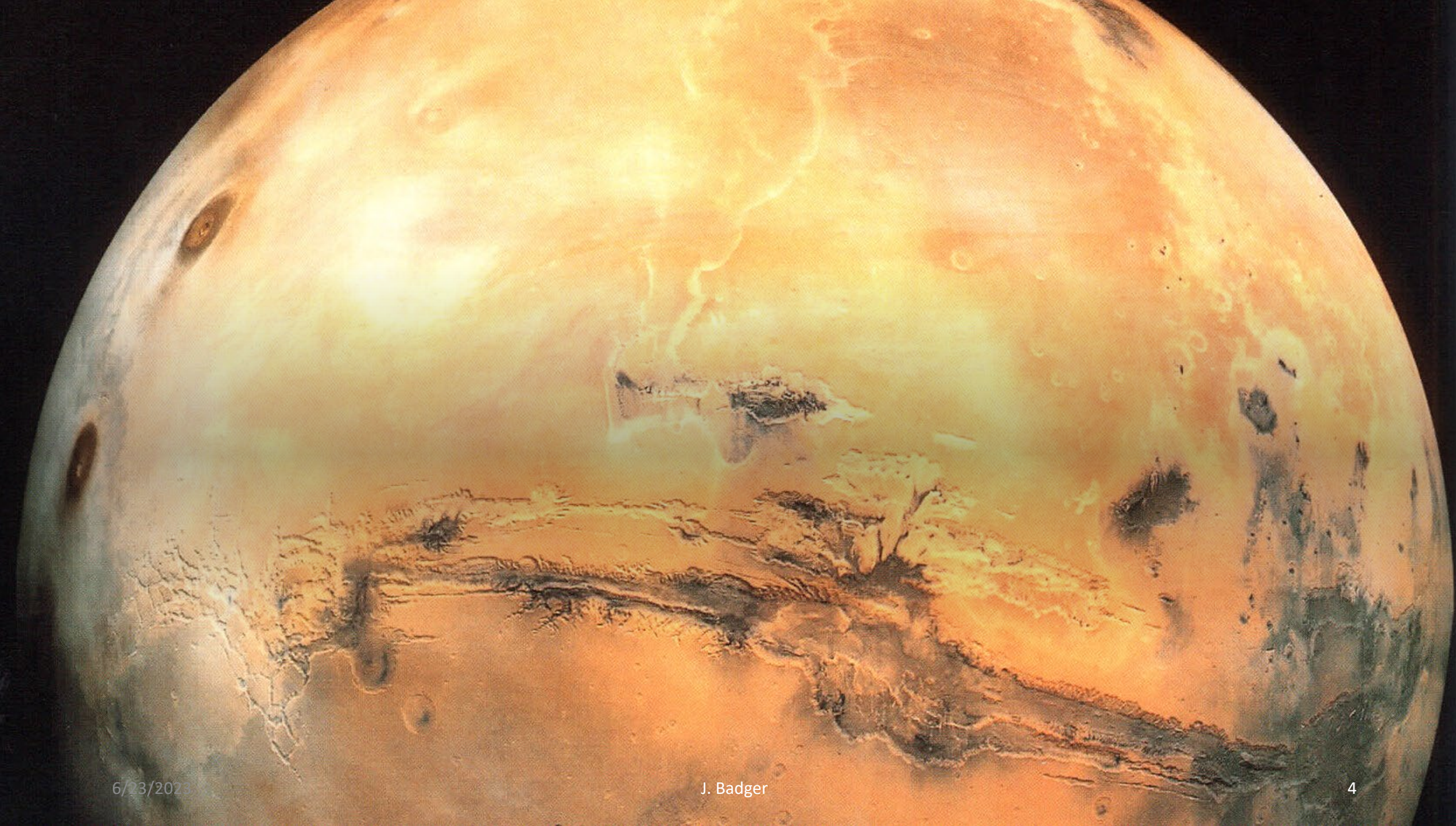
J. Badger

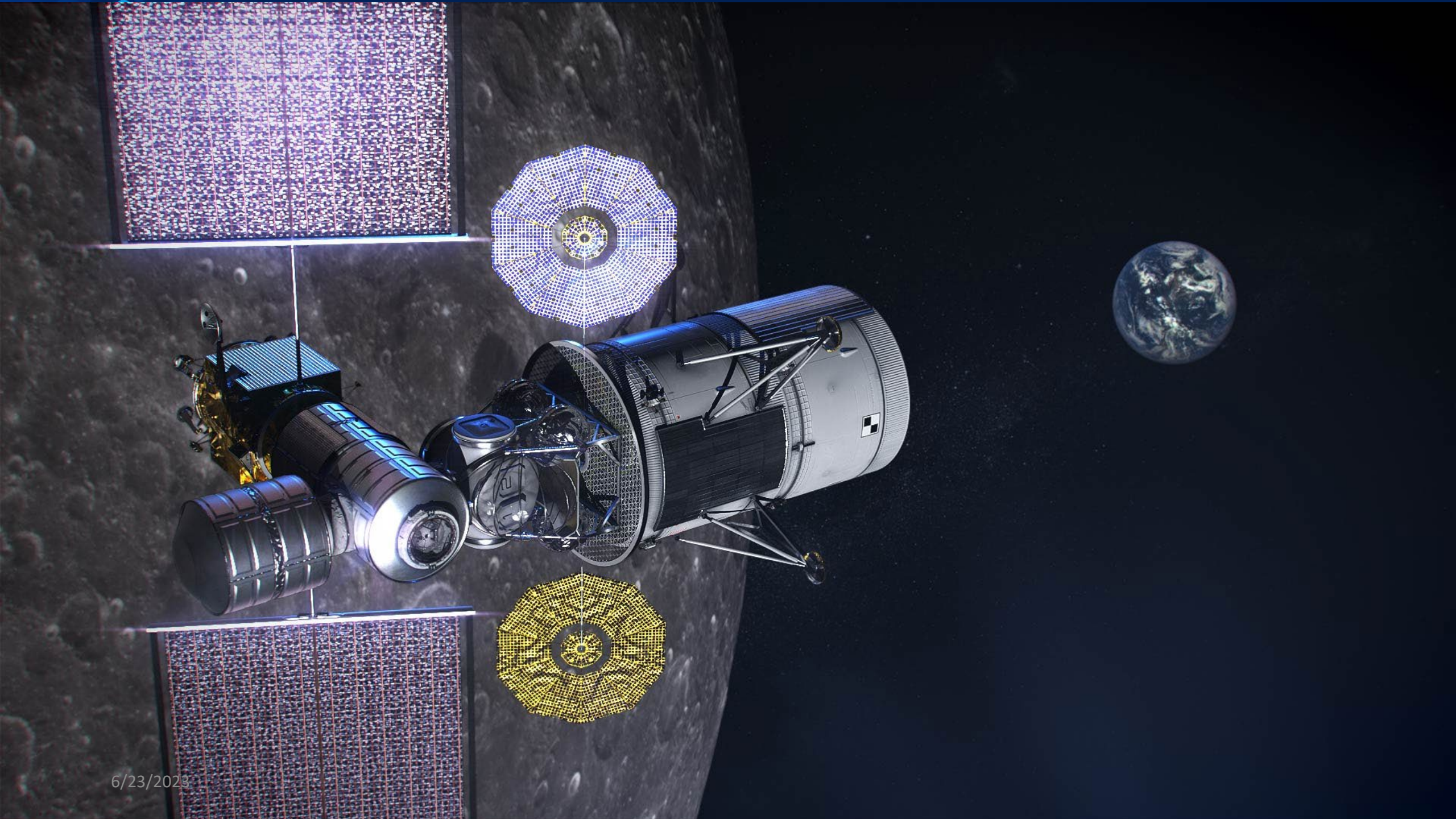




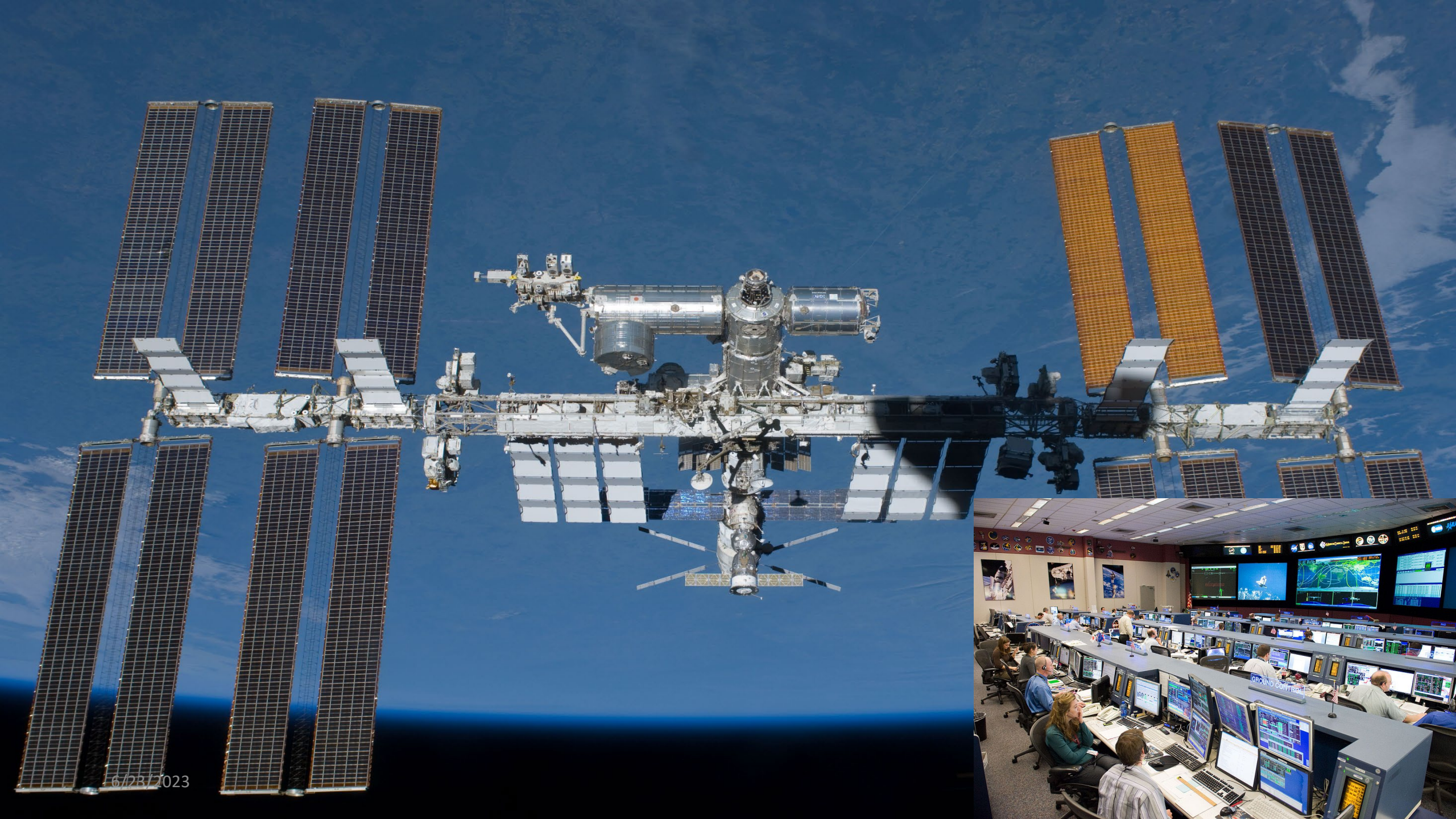
6/23/2023







6/23/2023



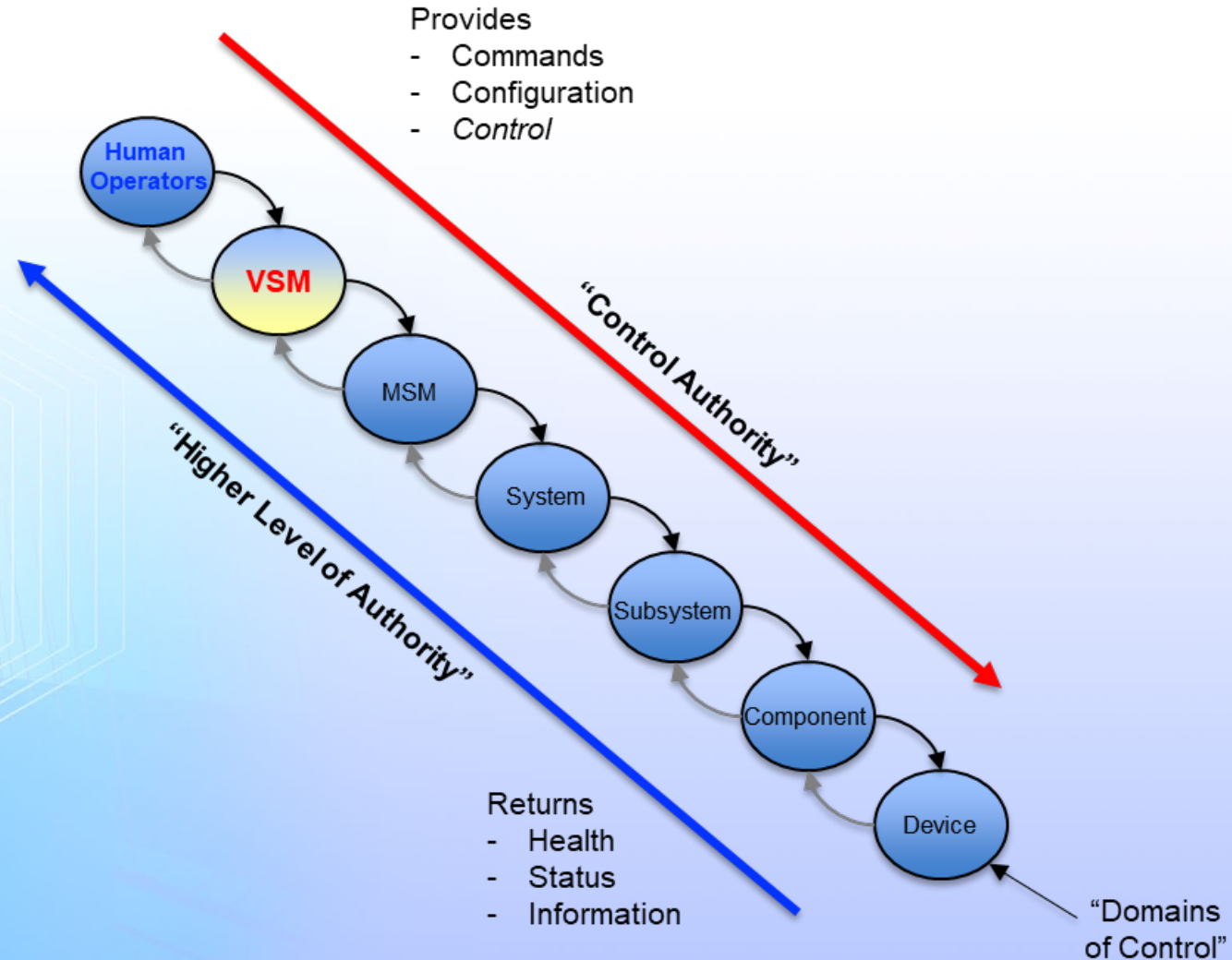
6/23/2023

Ways to Achieve Autonomy

- **System design** plays a major role
 - Early definition of subsystem interdependencies is key
 - Simplified interfaces, less complexity, and materials selection for more robust design
 - Robust, fail-operational designs for critical components
 - Make choices to increase the time to criticality
 - Design for robotic maintenance and inspection
- **Vehicle systems management** software provides in situ operational autonomy
 - Distributed, hierarchical architecture
 - ***Clear definitions of interfaces and interdependencies***
 - Careful design of locus of authority and associated transitions
 - Redundancies for telemetry data collected in case of failure or degradation

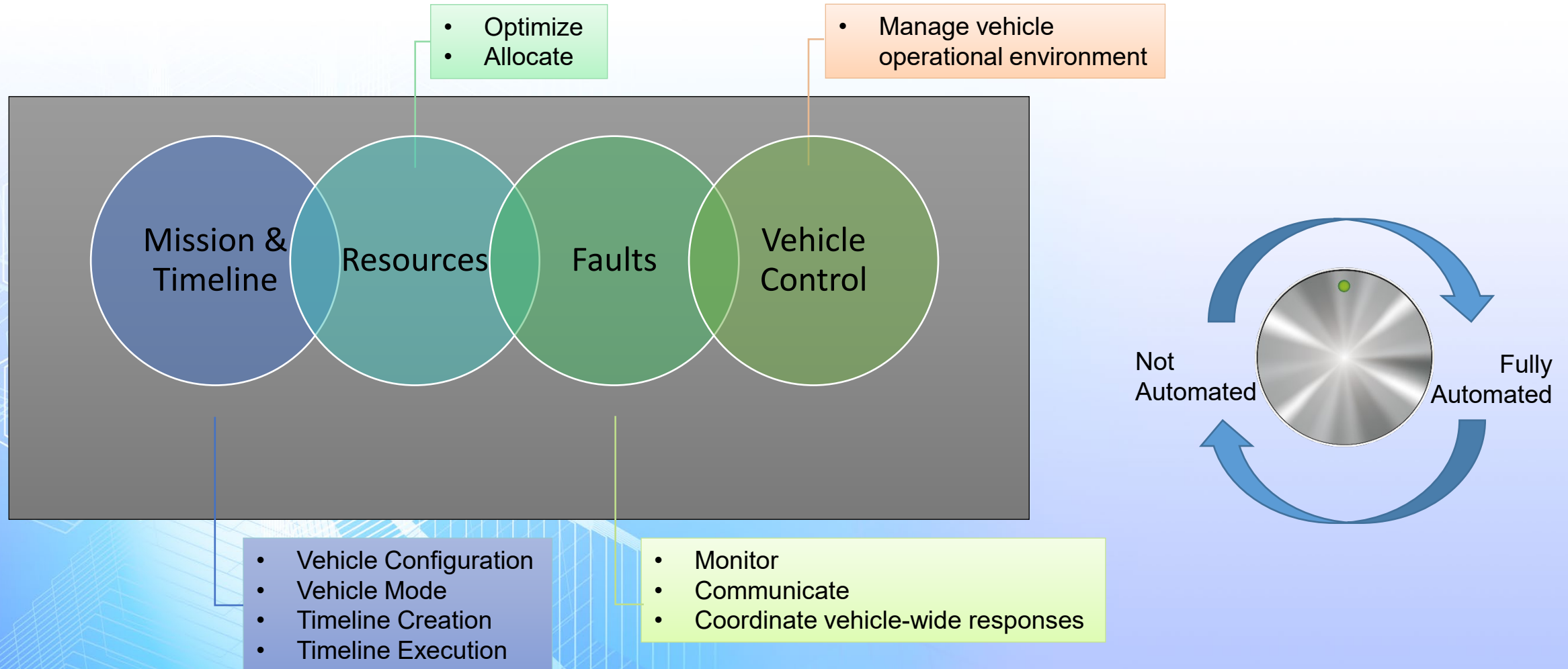
Gateway's Vehicle Systems Manager

- **Gateway's VSM** is designed around a distributed, hierarchical autonomous control concept
 - Domain of control should stay at lowest possible level
 - Control authority rises up only to appropriate level of visibility
 - Data/model abstraction allows each level of the control architecture to maintain a limited scope

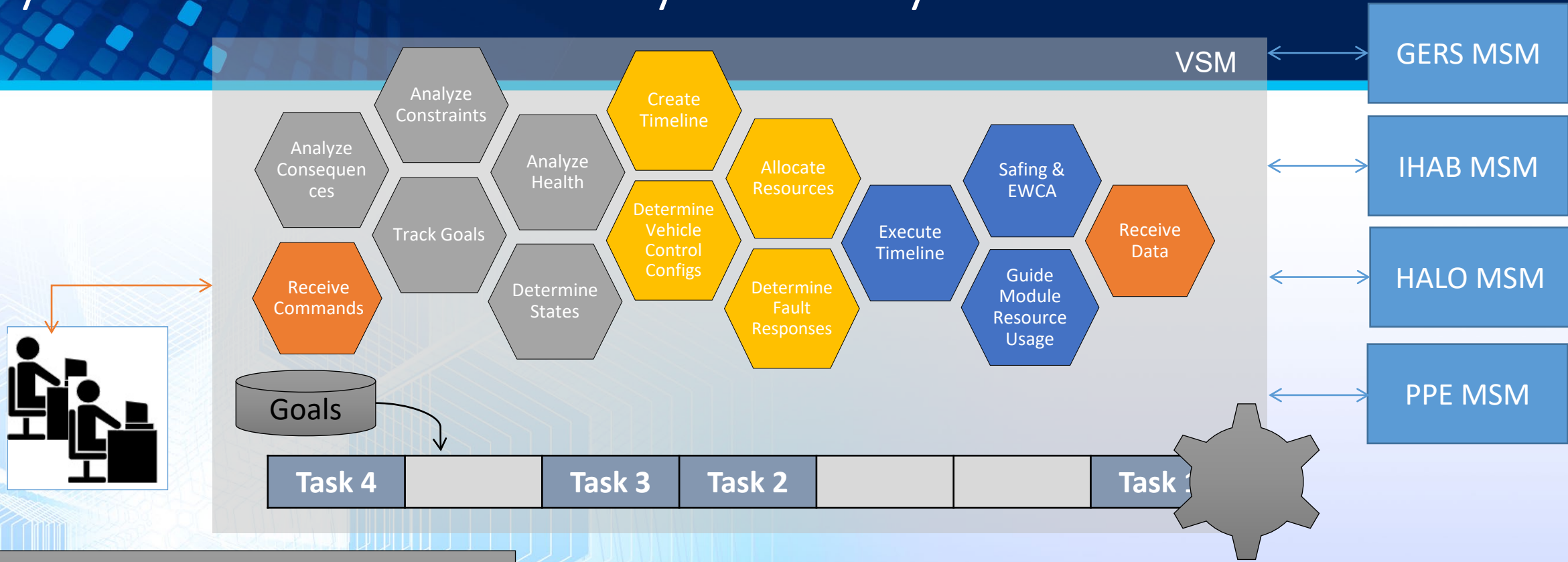


Vehicle Systems Manager's Main Functions

VSM has 4 management functions that can be operated at various levels of automation



Day in the Life of Gateway: 21-Day Automation



- All aspects of VSM automation are fully utilized
- Human operators
 - Define *mission goals* as inputs to VSM
 - Receive data and reports

Mission & Timeline

Resources

Faults

Vehicle Control

VSM Verification Plan

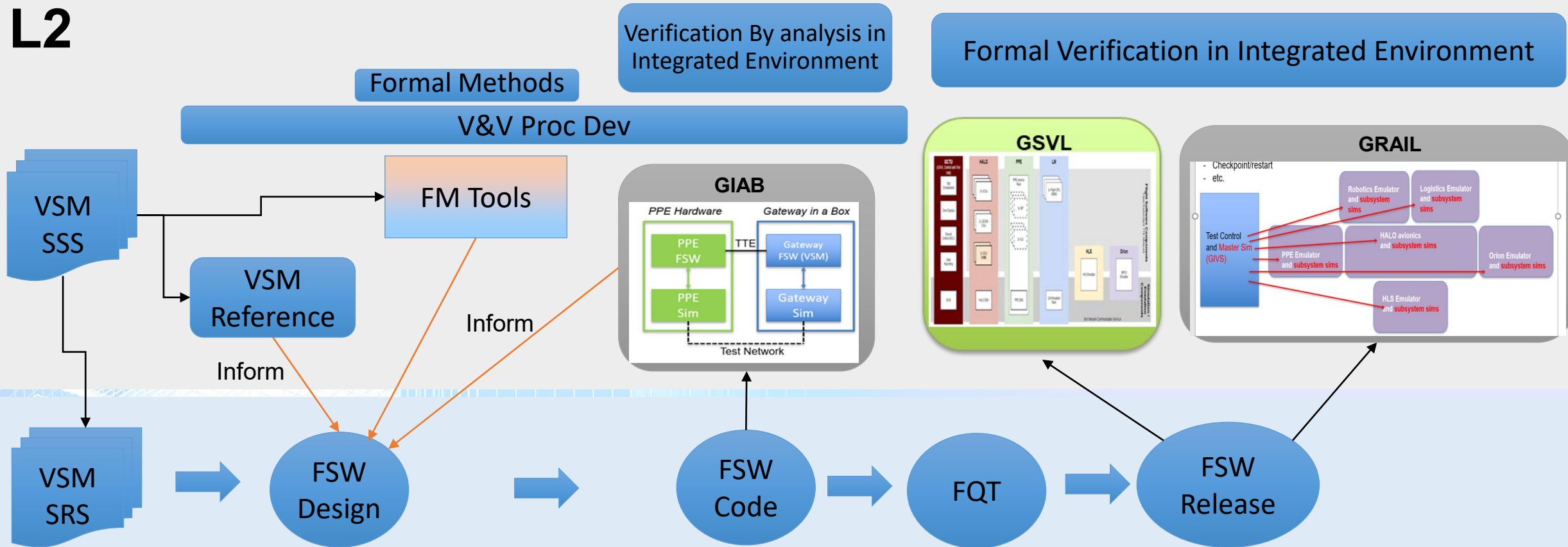


- Verification challenges
 - VSM will encounter environmental conditions and operational scenarios not currently envisioned
 - VSM will assess event consequences for human operators and make complex decisions
 - VSM is largely data-driven resulting in a large set of state trajectories
 - VSM will interact with subsystems developed in parallel or in the future by others
 - All the above increases the likelihood of defects and emergent behavior
- Verification goals
 - Identify defects in specification and implementation early in the development lifecycle
 - Exercise system across diverse state space and trajectories to ensure robustness and uncover emergent behavior
- Strategies
 - Comprehensive and layered verification using progressively higher fidelity test environments
 - Enhance existing verification methods to address additional risk associated with autonomy
 - Development and Operational assume-guarantee contracts
 - Data Model Semantic Verification

Iterative Development, Layered Verification



L2



L3

Verification Methods (Tailored for VSM)

Inspection

Analysis

Demonstration

Audit

Test

Model
Checking

Operational
Verification

Primary verification methods are Analysis and Test

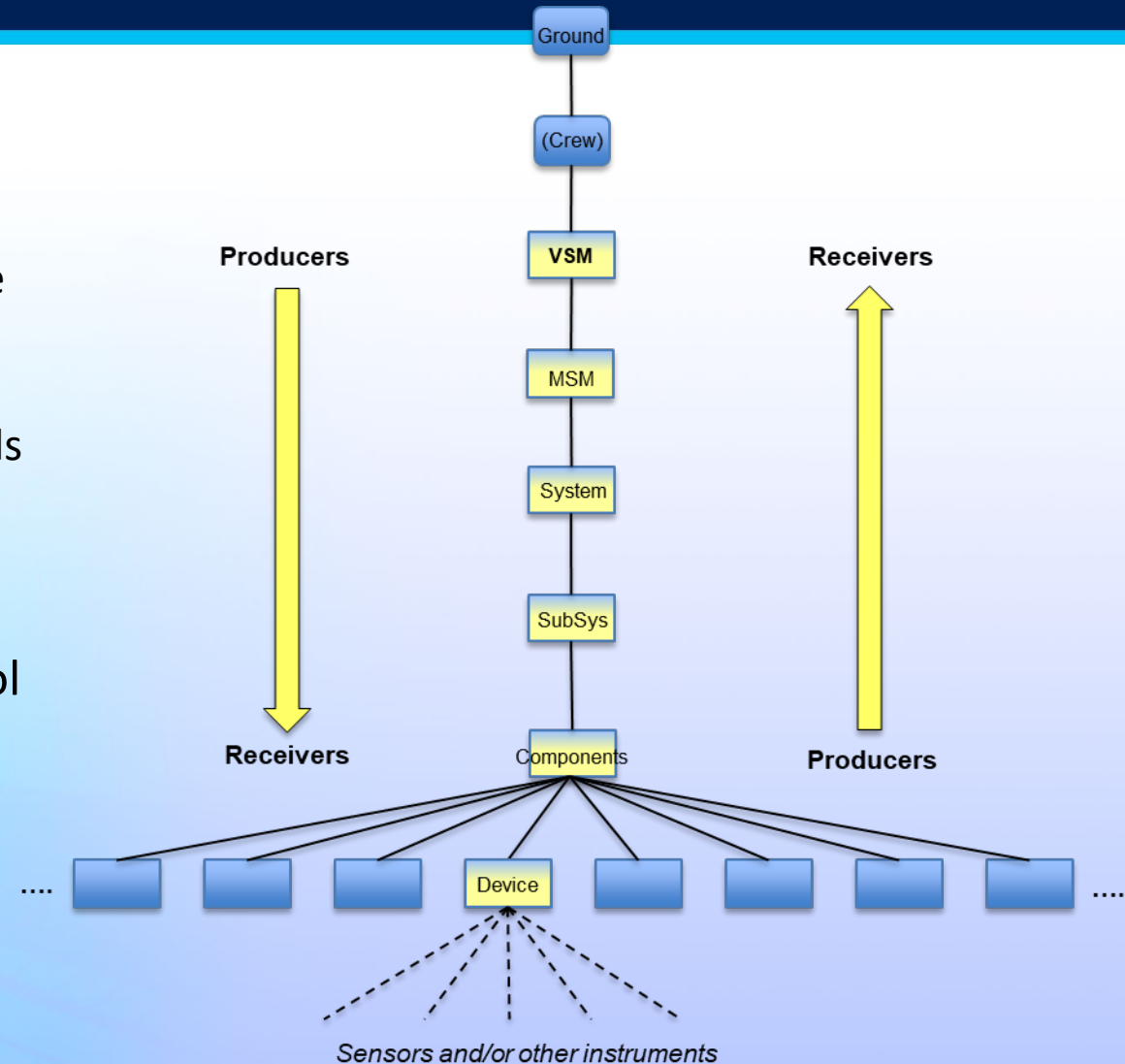
- ***Analysis*** is execution of ASMA (VSM, MSMs) in a simulated software environment using stochastic selection of initial conditions, injected inputs, and dispersions. Objective is to ensure robustness of VSM in complying with requirements.
- ***Test*** is execution of ASMA (VSM, MSMs) in a HW environment (flight equivalent processors, TTE network) using deterministic selection of initial conditions and injected inputs without dispersions. Objective is to verify functionality and performance of VSM in a TLYF environment.

Model checking and runtime verification added to address risk of emergent behavior resulting from autonomous operation

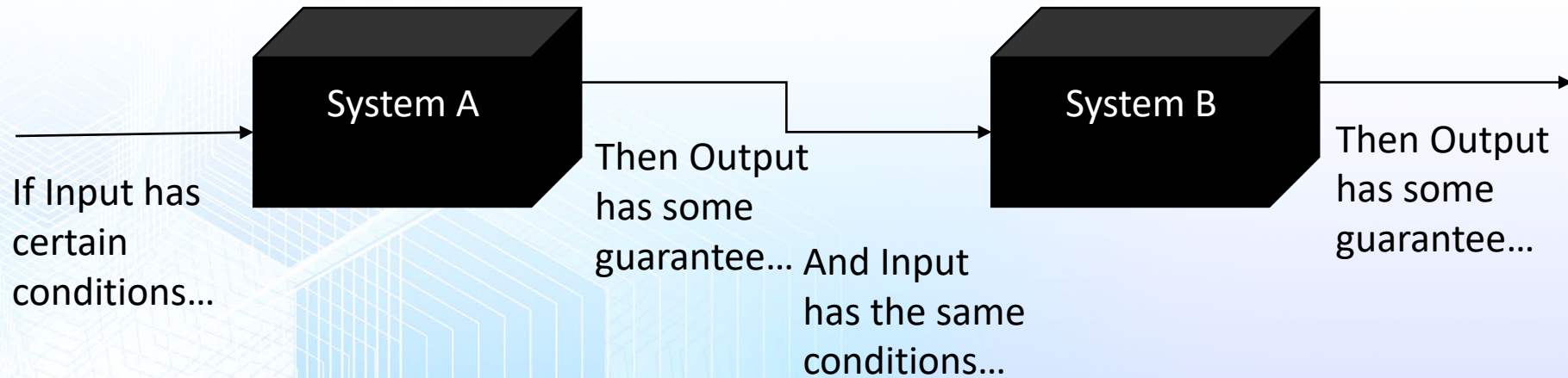
- ***Model checking*** – used during development on formal specifications that include system behavior and desired safety properties
- ***Operational verification*** – used during operations by employing a runtime monitor that executes in parallel with FSW

Solving Complex Problems through Abstraction

- Data abstraction key to making autonomy problem tractable
 - Static data (data stores) and computable data (runtime models) will be needed for VSM to perform its functions
 - Data and commands will have various abstraction levels to ensure that proper data and computation management occurs
- Module System Managers (and below) will contain low-level information about the systems under control (including immediate FDIR actions)
- VSM will take abstracted data (e.g., component and function health states) and derive vehicle level impacts and actions from them

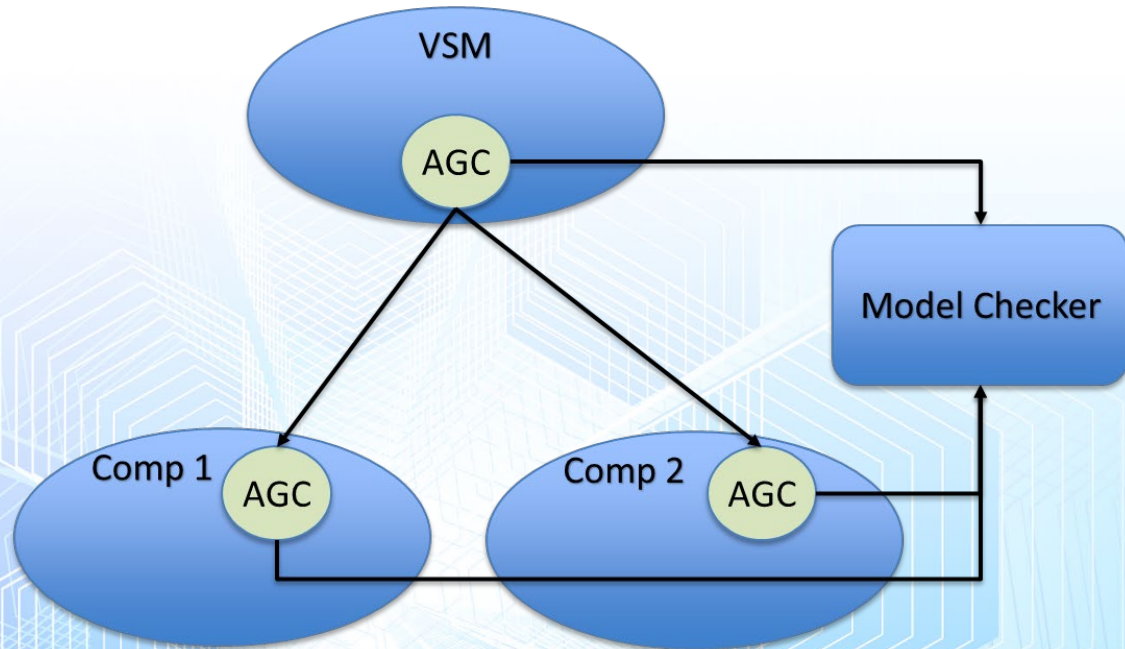


Assume-Guarantee Contracts



- Assume-Guarantee contracts are formal statements that allow for a logical separation of verification of interdependent systems
- Provides a framework for verification of distributed systems
 - Follows philosophy of well-defined interfaces and interdependencies

Contract Levels and Flow Down



- Assume-guarantee contracts can be expressed within systems between applications
- More commonly, AGC specify interfaces between hierarchical levels of control
- These contracts can simplify verification by focusing on one component at a time
- Decomposition can be checked using assertions in model checker
- Model checker determines if a finite-state model satisfies a given specification

Development and Operational Verification

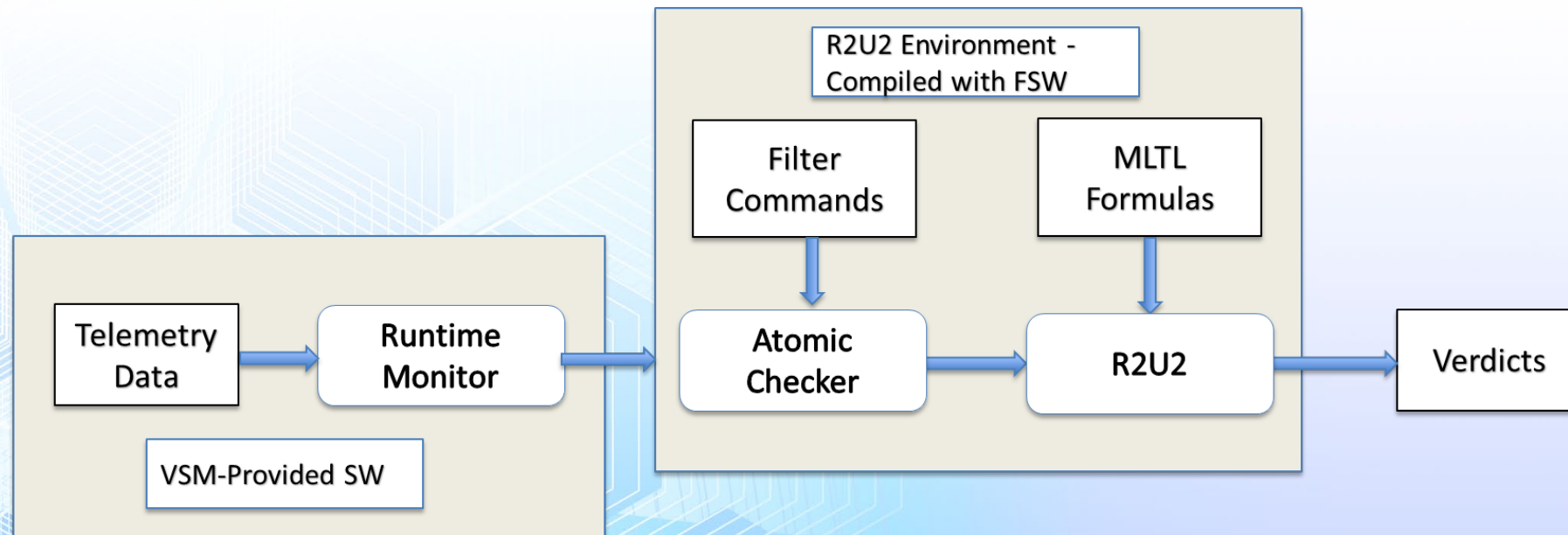
- Developmental verification
 - Performed at design time
 - Functionality represented as state machine using modeling language such as TLA+ or Promela
 - Contracts implemented as temporal formulas and compliance evaluated using model checker such as TLC or Spin
- Operational verification
 - Performed at runtime
 - Functionality is operational system
 - Contracts implemented as inline code or using runtime verification engine such as R2U2¹

1- Schumann, J., Moosbrugger, P., Rozier, K.Y. R2U2: *Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems*. In: Proc. RV2015: Runtime Verification 23 6th International Conference, Vienna, Austria, September 22-25, 2015 Proceedings. pp. 233-249 (2015), http://dx.doi.org/10.1007/978-3-319-23820-3_15

Specifying Operational Contracts

- AGCs use two types of logic
 - Propositional: $A \wedge B$ (*A and B are both true now*)
 - Temporal: $\diamond(A \wedge B)$ (*Eventually, A and B will both be true*)
- Both types can be coded in C, but
 - Coding temporal formulas is difficult
 - Coded temporal formulas are notoriously error-prone
 - Difficult to test – huge number of test cases required
 - Frequent source of verification and operational errors
- R2U2 provides a more reliable approach using mission-time linear temporal logic
 - Adds time bounds to temporal relations

Operational Contract Implementation



- Runtime monitor (RuM) is interface between data sources and R2U2 system
- Atomic checker produces Booleans by filtering data (float, int) from RuM
- Atomic checker and R2U2 command file processed at compile time, generate executable code

Conclusions

- Gateway is espousing autonomous systems technology to enable human exploration beyond Low Earth Orbit
- Assume-guarantee contracts provide a mechanism to allow a 'divide-and-conquer' verification approach that is formally correct
- Continued use of formal methods expected as capabilities take more and more of a non-deterministic feel (e.g., uncertainty-based planning algorithms)

