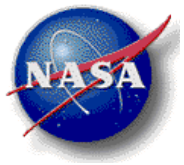
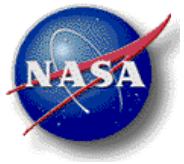


The Challenge of Verification and Validation of Automated Planning Systems

Jeremy Frank

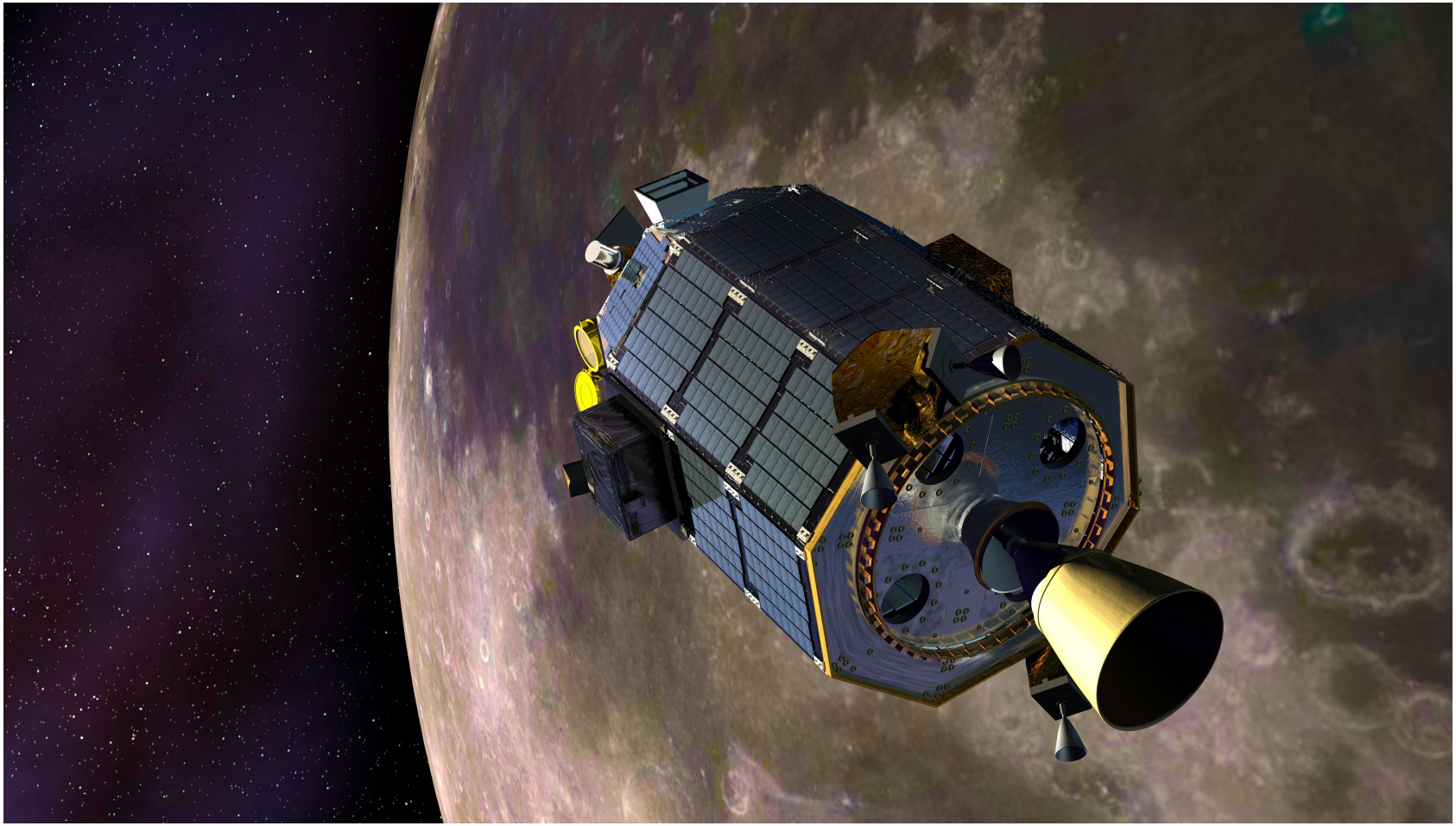
NASA Ames Research Center

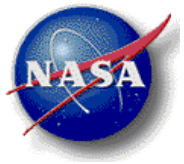


The Takeaway

- Verification and Validation of space mission planning systems is a challenging problem with several unique features:
  - Planning performed at a *high level of abstraction* (compared to spacecraft behavior).
  - V&V of *model*, not just code.
  - Availability of a simulation as an 'oracle'.
- V&V of mission planning can be *automated* by formalizing the relationship between the plan (actions, states) and the spacecraft behavior (commands and data).

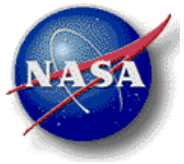
The Challenge





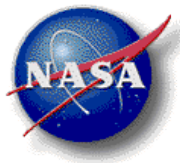
The Challenge

- Mission Planning Systems are a key part of the ground segment of every space mission.
  - Mission planners generate courses of action to satisfy mission goals while respecting constraints.
- Like other software, the Mission Planning System (henceforth, the *planner*) must be verified and validated.
  - Verify that plans generated by the planner satisfy constraints and achieve goals.
  - Validate the constraints the planner enforces.



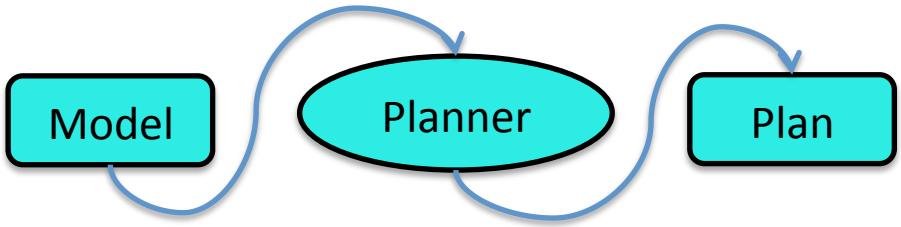
The Challenge

- Like (many) other applications, spacecraft mission planning places a premium on planning due to the high costs of defects and the complexity of the application.
  - Opportunities to test the planner prior to operations are limited to hardware simulations and ground-based testbeds.



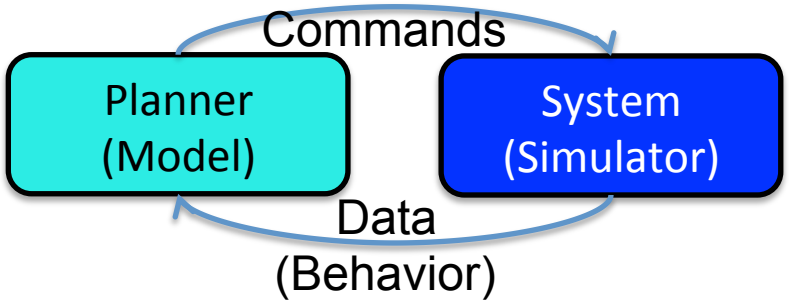
The Challenge

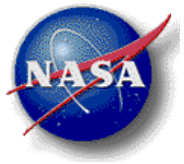
- Unlike other software, *model-based* planning systems change the nature of the software verification and validation problem.
  - The model is an input to the planning system.
  - Model verification and validation is different than software verification and validation!



The Challenge

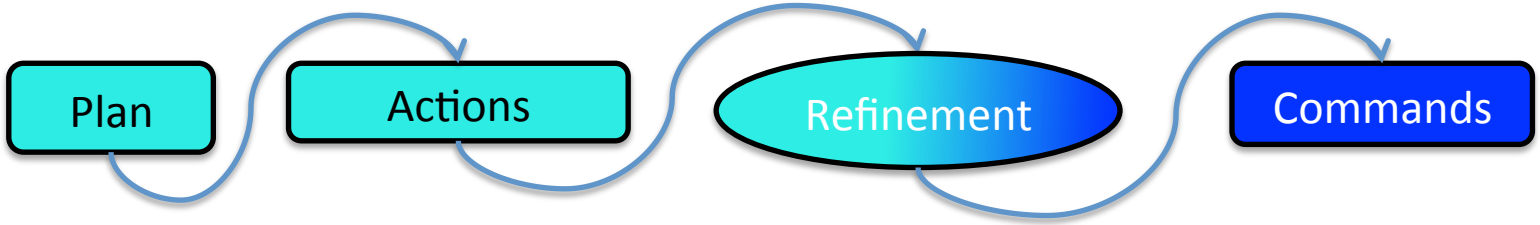
- Unlike other applications, the existence of a *simulation* offers an opportunity to validate the planner.

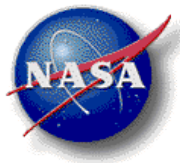




The Challenge

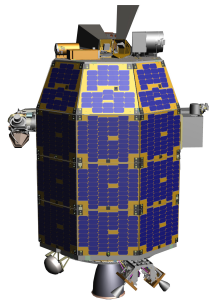
- Unlike other applications, *abstraction* is a key element of the V&V problem.
  - Planners approximate or abstract the actual spacecraft behavior.





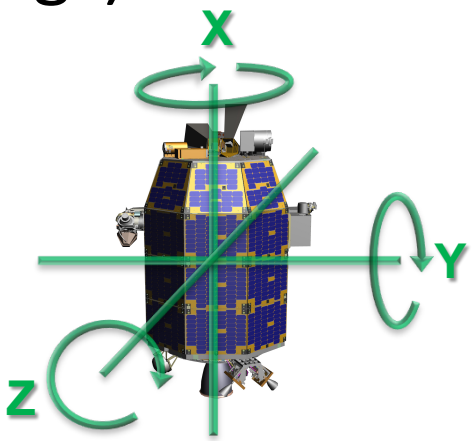
Example

- Suppose we are developing a mission planning system for a spacecraft.
- How does a spacecraft change the direction (attitude) it is pointing?
  - A change of pointing is called a slew.

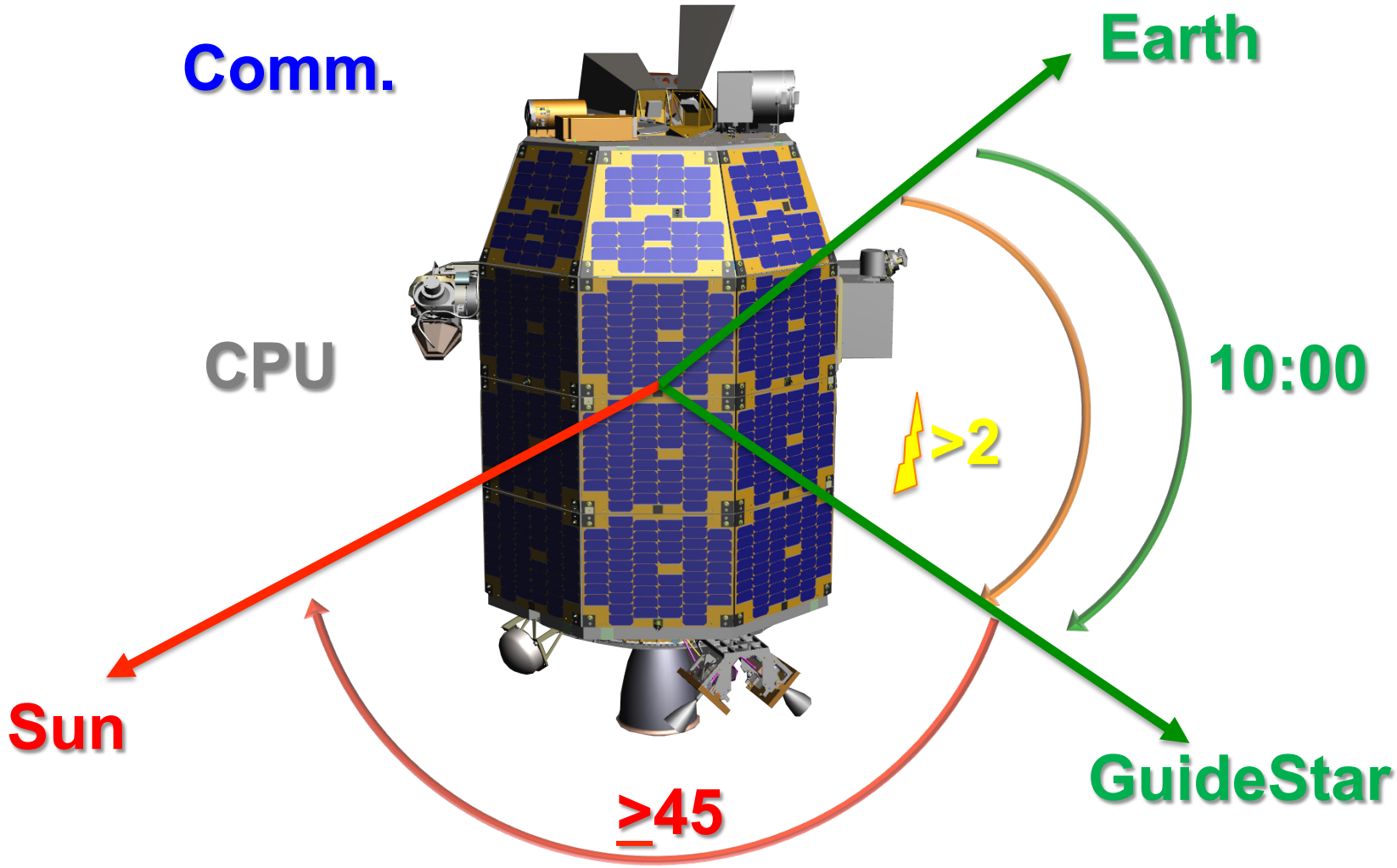


Example

- Attitude is angles <*x*, *y*, *z*> around the x,y, and z spacecraft body axes.
- Slews are constrained by solar panel power generation, thermal, communications to Earth, sensor and instrument performance and safety (among other things).

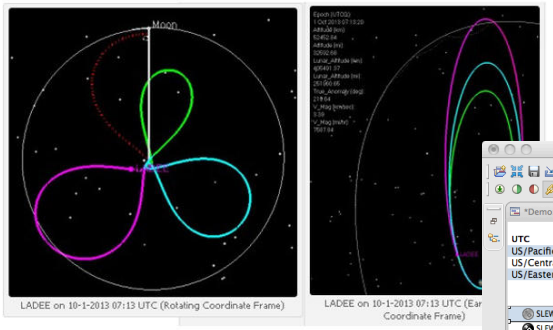


Example



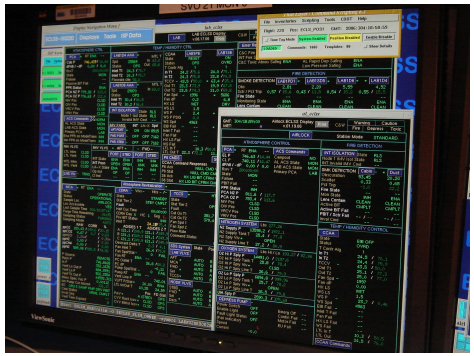
Mission Planning System

Flight Dynamics

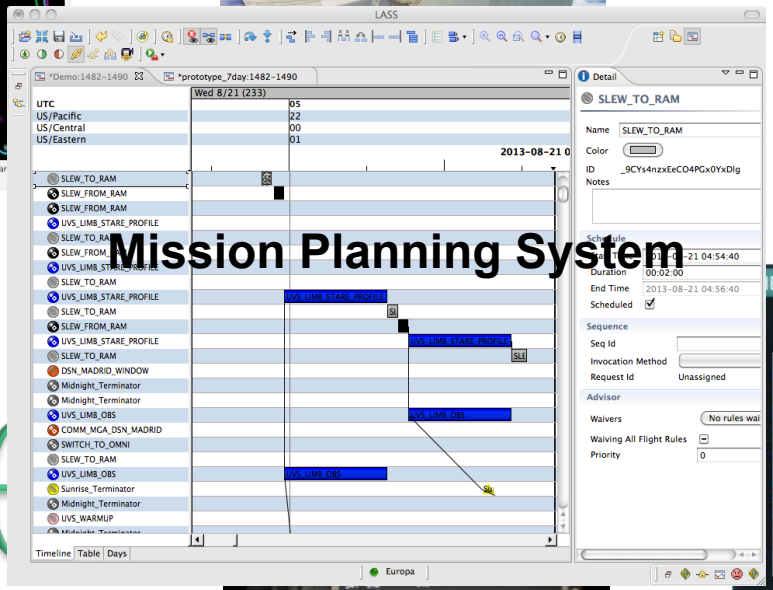


Ground Network Scheduling

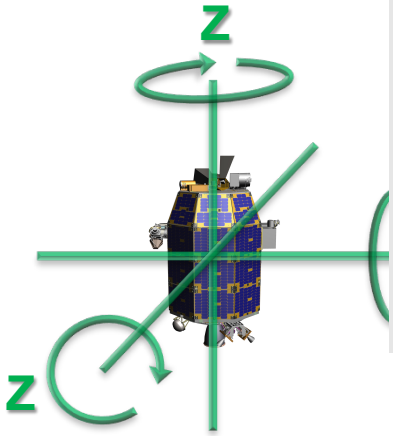
Command and Telemetry System



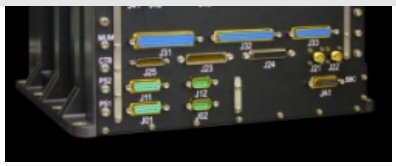
Mission Planning System



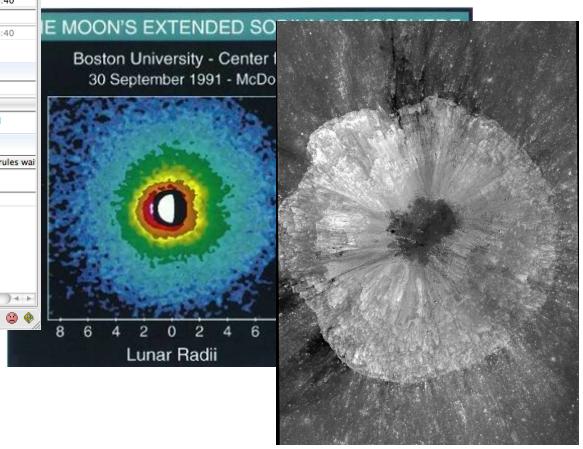
Attitude Determination

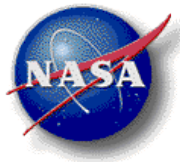


Flight Software and Hardware



Science Operations

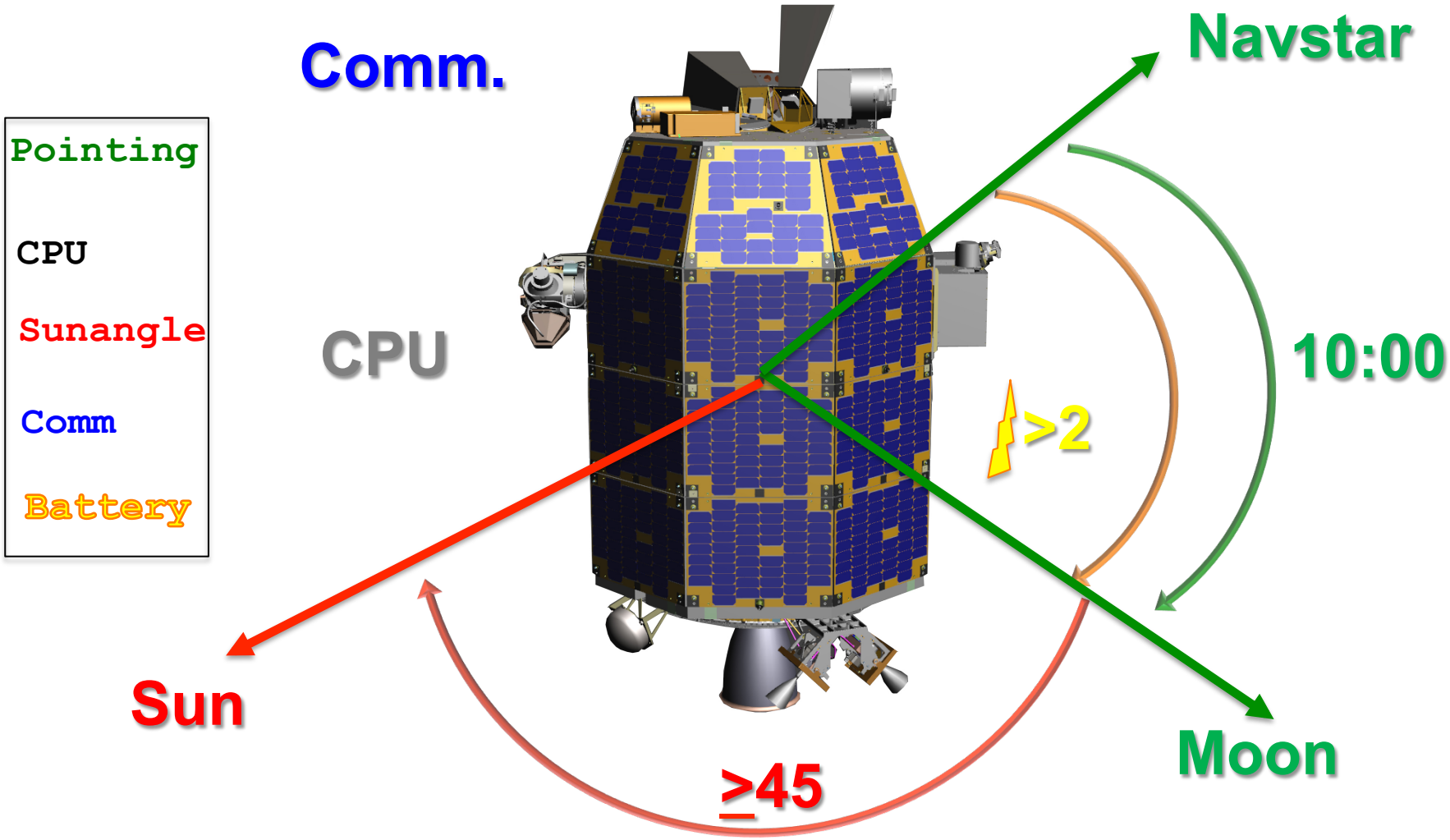


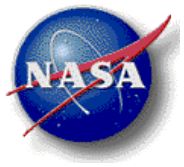


Mission Planning System

- A planning *model* consists of:
  - Objects – things in the world.
  - States – properties of things.
  - Actions – ways of changing the properties of things.
- A planning problem consists of:
  - A model.
  - An initial state description.
  - A set of goal states.
- The planner reads the model, initial states, and goals, and produces a plan.
- There are myriad planning algorithms with many different properties.
  - Run-time, types of plans produced, etc.

Example


Pointing
CPU
Sunangle
Comm
Battery
Comm.
CPU
Navstar
10:00
>2
Sun
≥45
Moon



Model-Based Planning

```
(:durative-action slew
 :parameters (?from - attitude
              ?to - attitude)
 :duration (= ?duration 10)
 :condition (and
     (at start (pointing ?from))
     (at start (cpu on))
     (over all (cpu on))
     (at start (>= (sunangle) 45.0))
     (over all (>= (sunangle) 45.0))
     (at start (communicating))
     (over all (communicating))
     (at start (>= (batterycharge) 2.0)))
 :effect
  (and
     (at start (decrease (batterycharge)
 2.0))
     (at start (not (pointing ?from)))
     (at end (pointing ?to))))
```

Pointing

CPU

Sunangle

Comm

Battery

Action

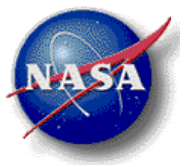
Object

Condition

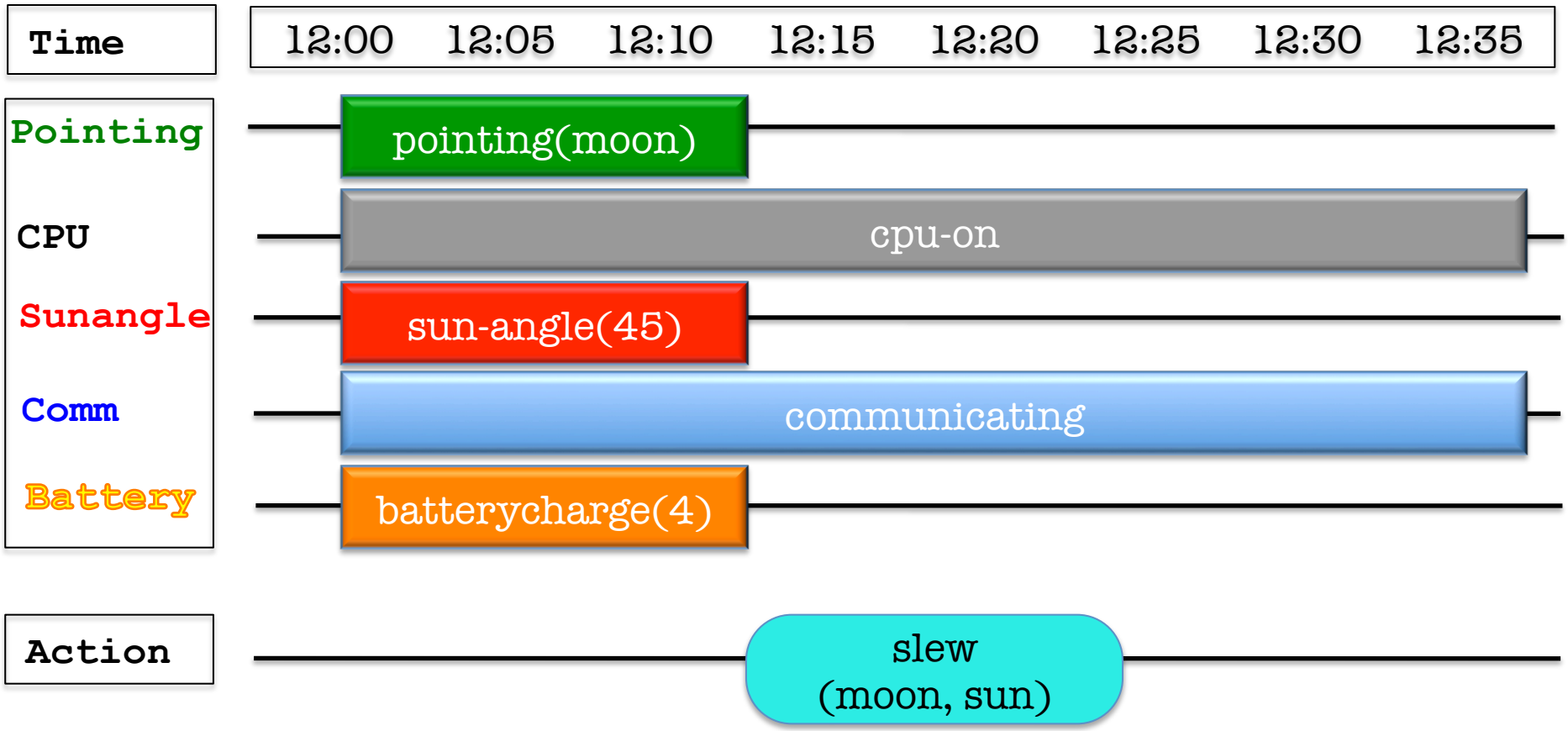
States

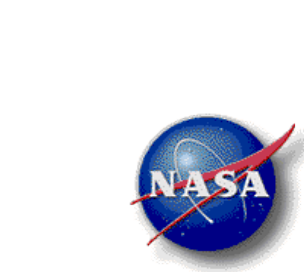
Effect

State

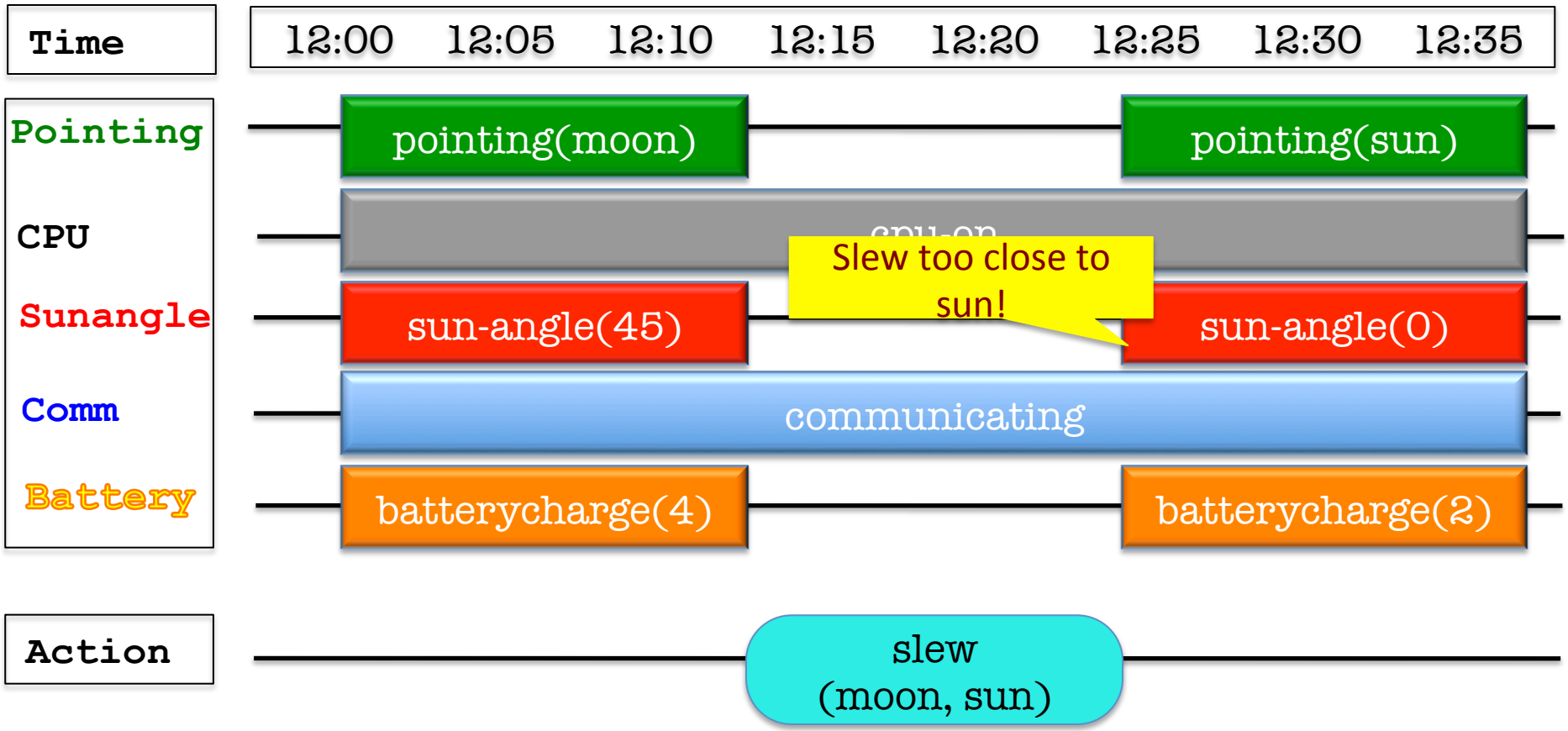


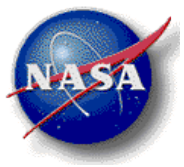
Model-Based Planning



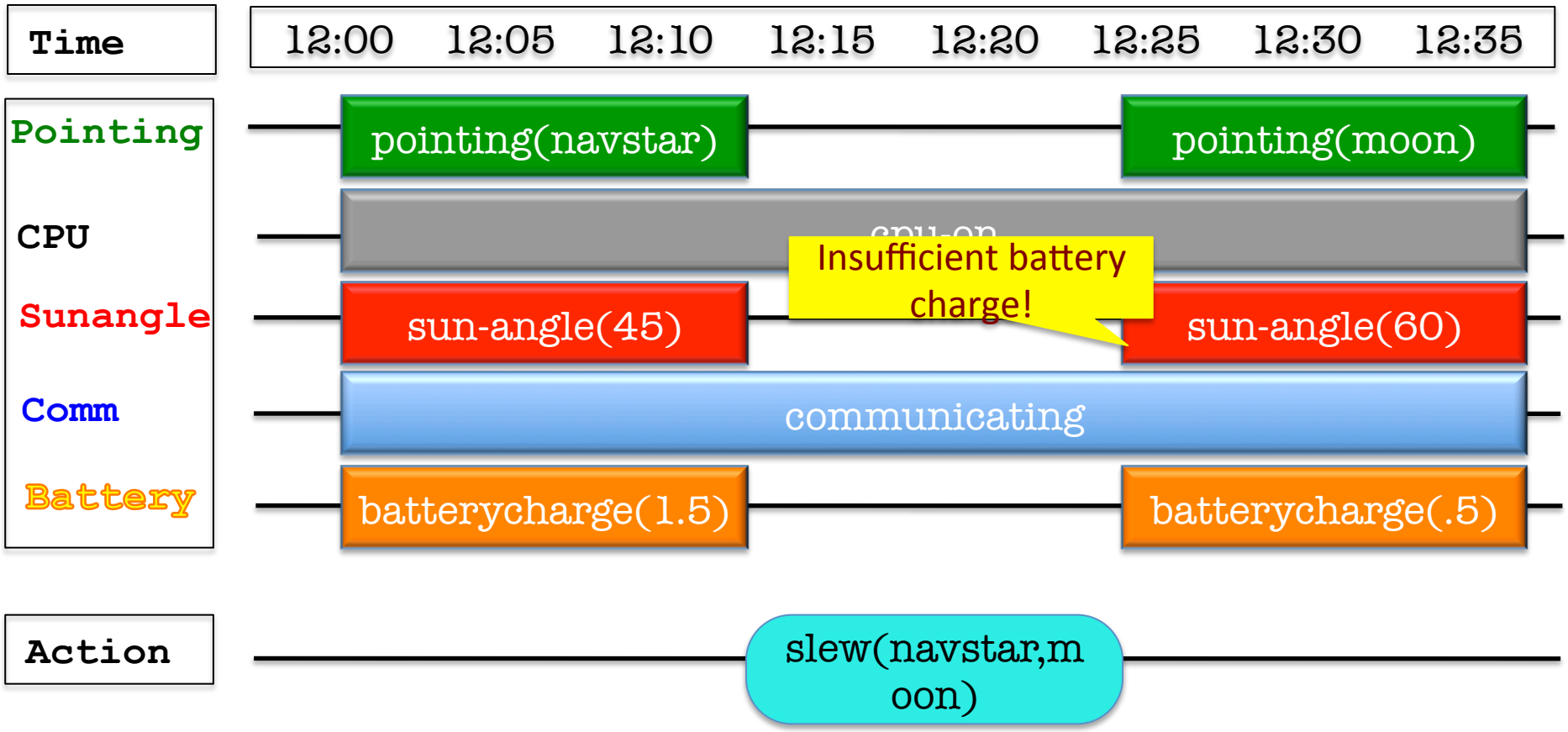


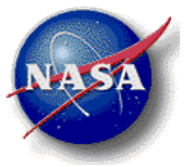
Model-Based Planning



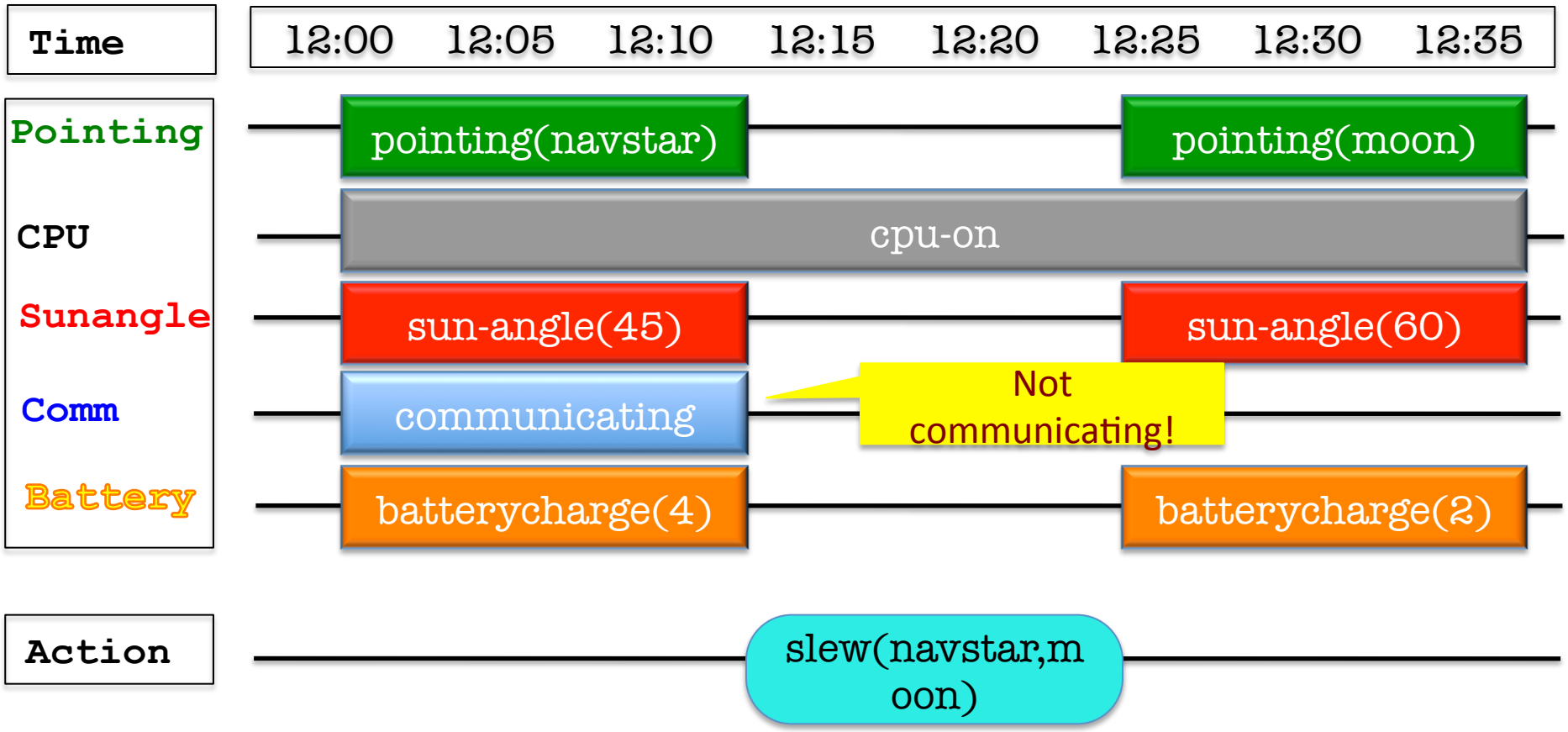


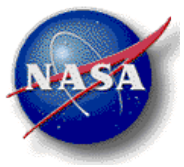
Model-Based Planning



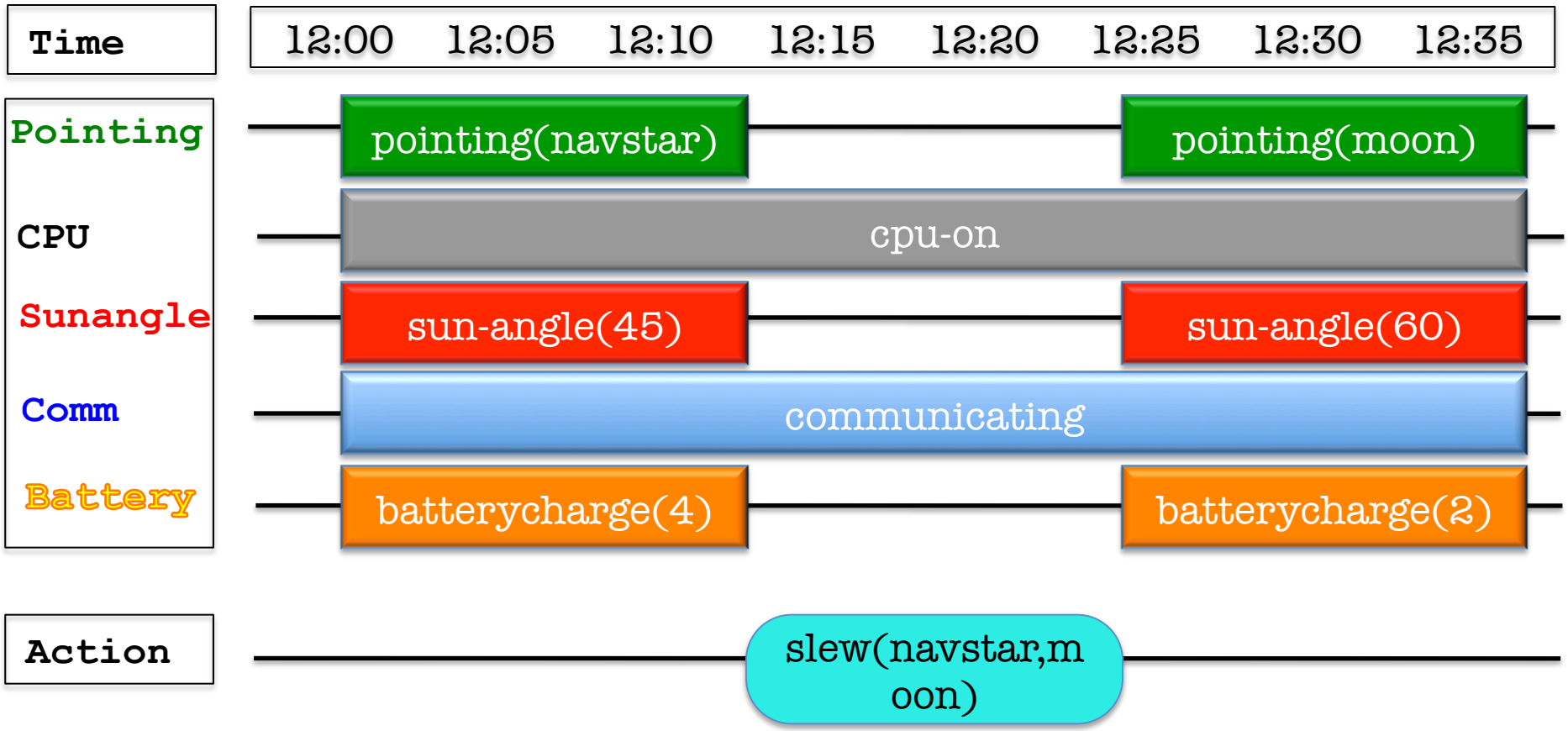


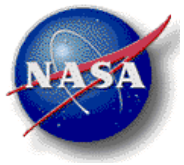
Model-Based Planning





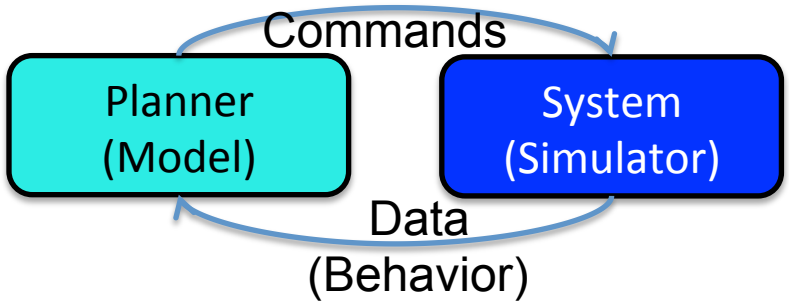
Model-Based Planning



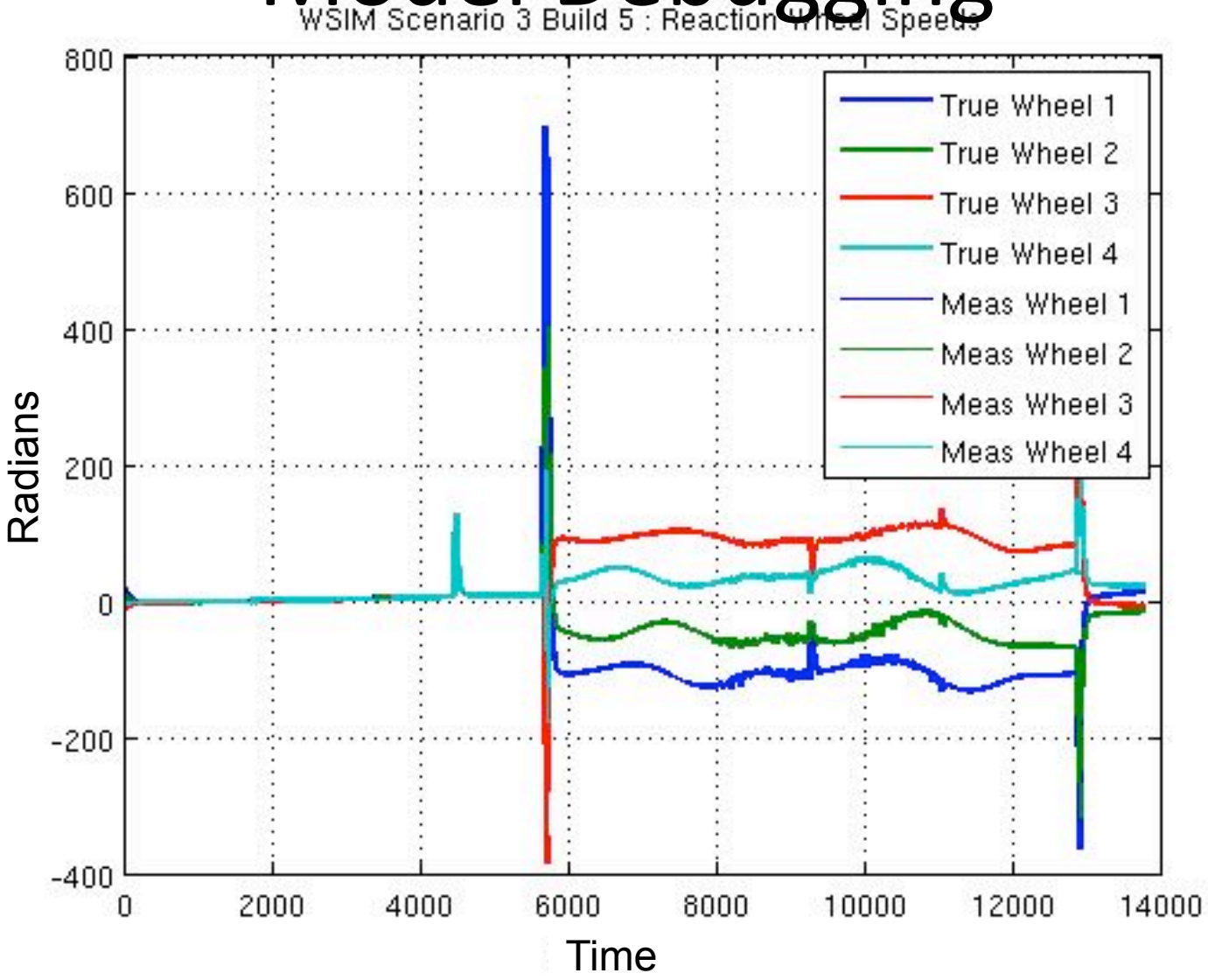


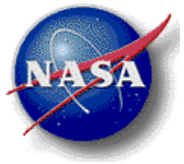
Model Debugging

- Spacecraft simulation:
  - Accepts a sequence of commands
  - Coupled with an initial spacecraft state
  - Produces output corresponding to spacecraft behavior
  - Using software, hardware, or both to simulate the spacecraft



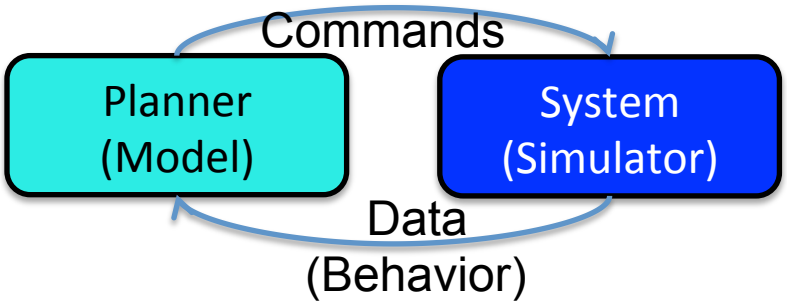
Model Debugging

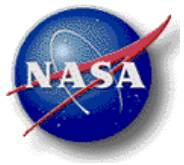




Model Debugging

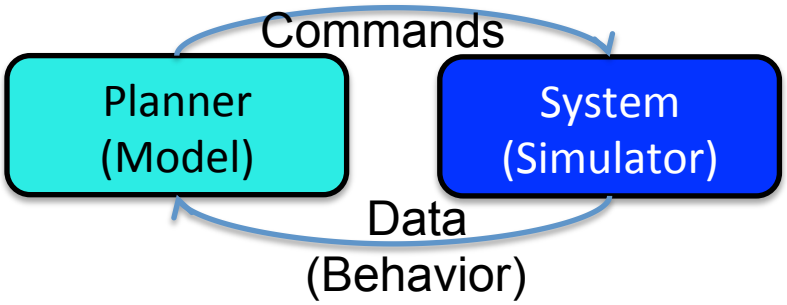
- What does it mean to validate the planner model with respect to the simulation?
  - Valid plans should execute on the simulator.
  - States predicted by planner should match (up to specified tolerances) the states produced by the simulator

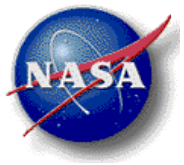




Model Debugging

- What does it mean to validate the planner model with respect to the simulation?
  - Does the slew take the expected amount of time?
  - Does the planned slew take place?
  - Does spacecraft point to proper place?



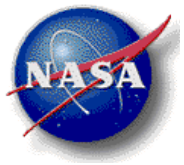


Model Debugging

12:00	12:05	12:10

slew
(navstar,moon)

Select RCS Pulsing	Enable RCS Thruster	Fire RCS Thruster	X Body Rate < cutoff	Pointing Attitude
			Y Body Rate < cutoff	Moon
			Z Body Rate < cutoff	



Model Debugging

12:00 | 12:05 | 12:10

slew
(navstar,moon)

Select RCS Pulsing

Enable RCS Thruster

Fi RCS Thrus r

X Body Rate < cutoff

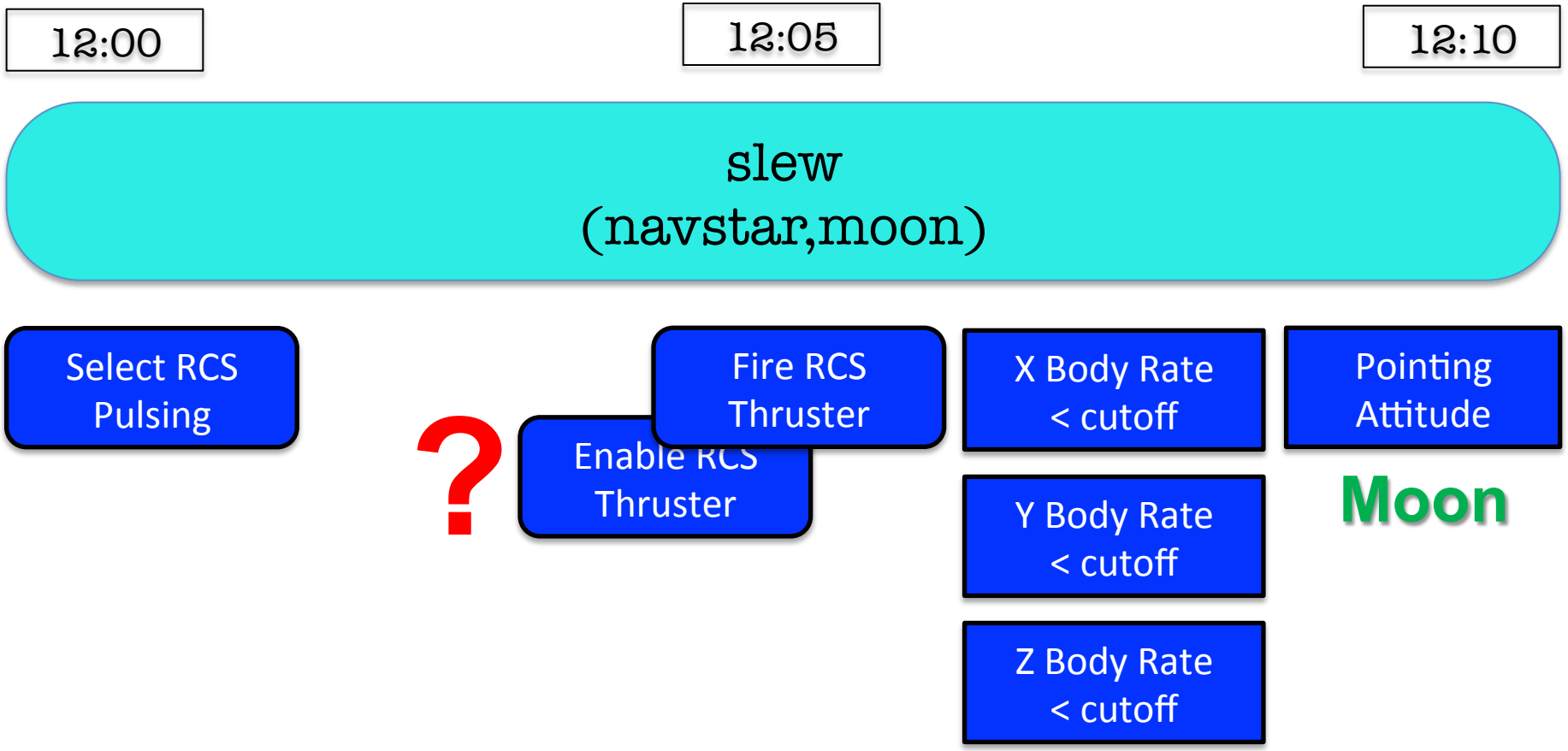
Y Body Rate < cutoff

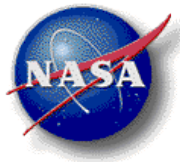
Z Body Rate < cutoff

Pointing Attitude

Moon

Model Debugging





Model Debugging

12:00 | 12:05 | 12:10

slew
(navstar,moon)

Select RCS Pulsing | Enable RCS Thruster | Fire RCS Thruster | X Body Rate < cutoff | Pointing Attitude | ?

Y Body Rate < cutoff | Moon

Z Body Rate < cutoff

Model Debugging

12:00 | 12:05 | 12:10

slew
(navstar,moon)

Select RCS Pulsing

Enable RCS Thruster

Fire RCS Thruster

X Body Rate < cutoff

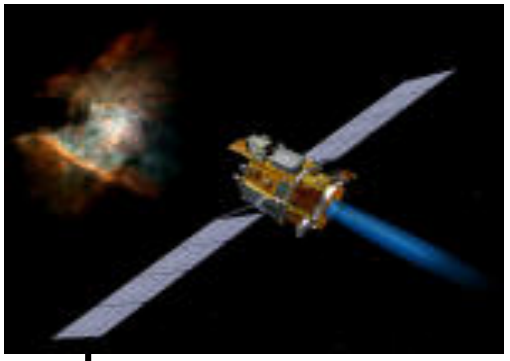
Y Body Rate < cutoff

Z Body Rate < cutoff

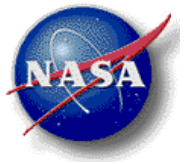
Pointing

Sun

Previous Work (Applications)



- Remote Agent [1], EO-1 [2]
  - Extensive model reviews.
  - Safety reviews to elicit potential hazards.
  - Automated tests stochastically generated by perturbations of nominal scenarios.
  - Executed on simulation platforms of varying fidelity where spacecraft, operations, and safety constraints were checked.

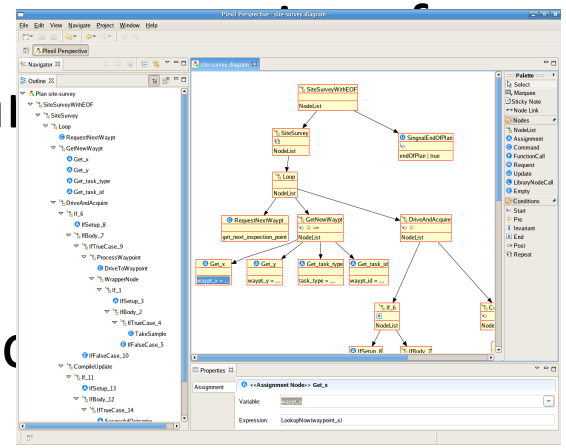


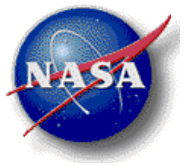
Previous Work (Academia)

- itSimple [3]
  - Allows some domain behavior modeling using UML object diagrams.
  - Generated plans can be checked against the UML.
- KEEN [4]
  - Similar to itSimple, but uses Timed Game Automata (TGA) instead of UML as domain model.
  - Emphasis on temporal planning domains and temporally flexible plans.
- PDVer [5]
  - Plan domain properties specified in LTL (Linear Temporal Logic).
  - Specification of test cases (goals) automatically from LTL.

Previous Work (Academia)

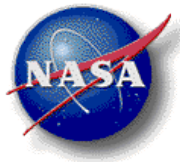
- VAL [6]
  - Given a plan and a model, determines whether the plan satisfies the constraints in the domain.
  - Limited ability to automatically fix plans.
- Model checking as plan verification [7]
  - Employs Java Pathfinder to check PLEXIL, a language and plan system.
  - Requires a system model (or and set of properties to





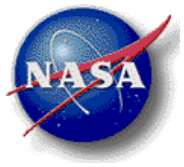
Previous Work (Summary)

- There are tools to assist in verification of plans against planning models.
- There are tools to assist in test case generation and model verification.
- *Few to no tools* to assist in validation of models.
- *No tools* to assist in validation against simulations.

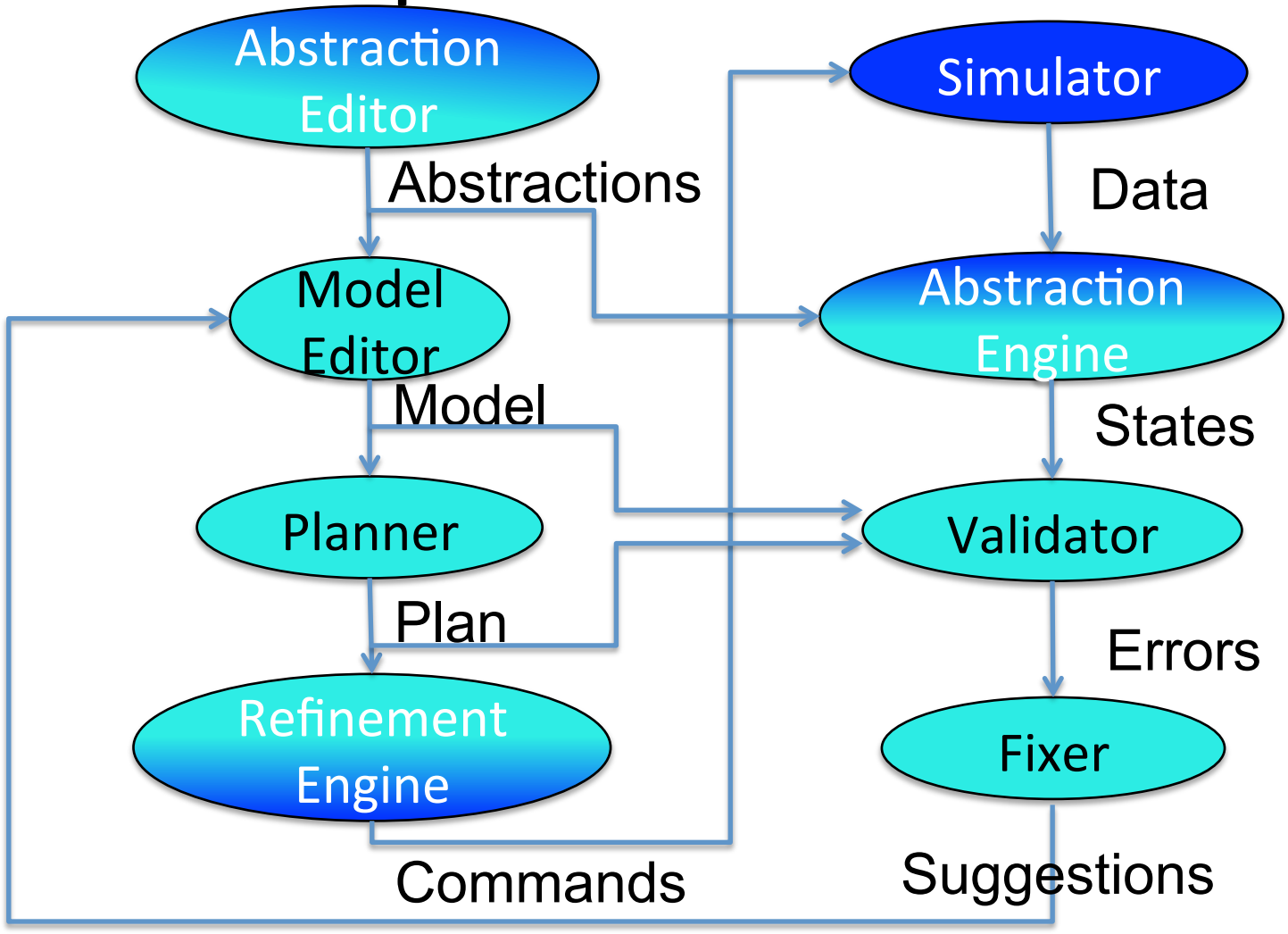


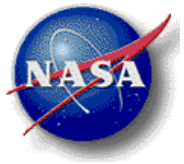
Integrated Model Development Environment

- The Integrated Model Development Envrionment (IMDE) bridges the gap between planning and simulation:
  - Modeler documents translation of plans to command sequences.
  - *Facilitates automatic generation of simulations.*
  - Modeler documents translation of spacecraft data to plan states.
  - *Facilitates automatic translation of simulation output into planning model states.*
  - *Enables comparison of planned and simulated spacecraft behavior.*
- See [8,9] for a more complete description.



Integrated Model Development Environment



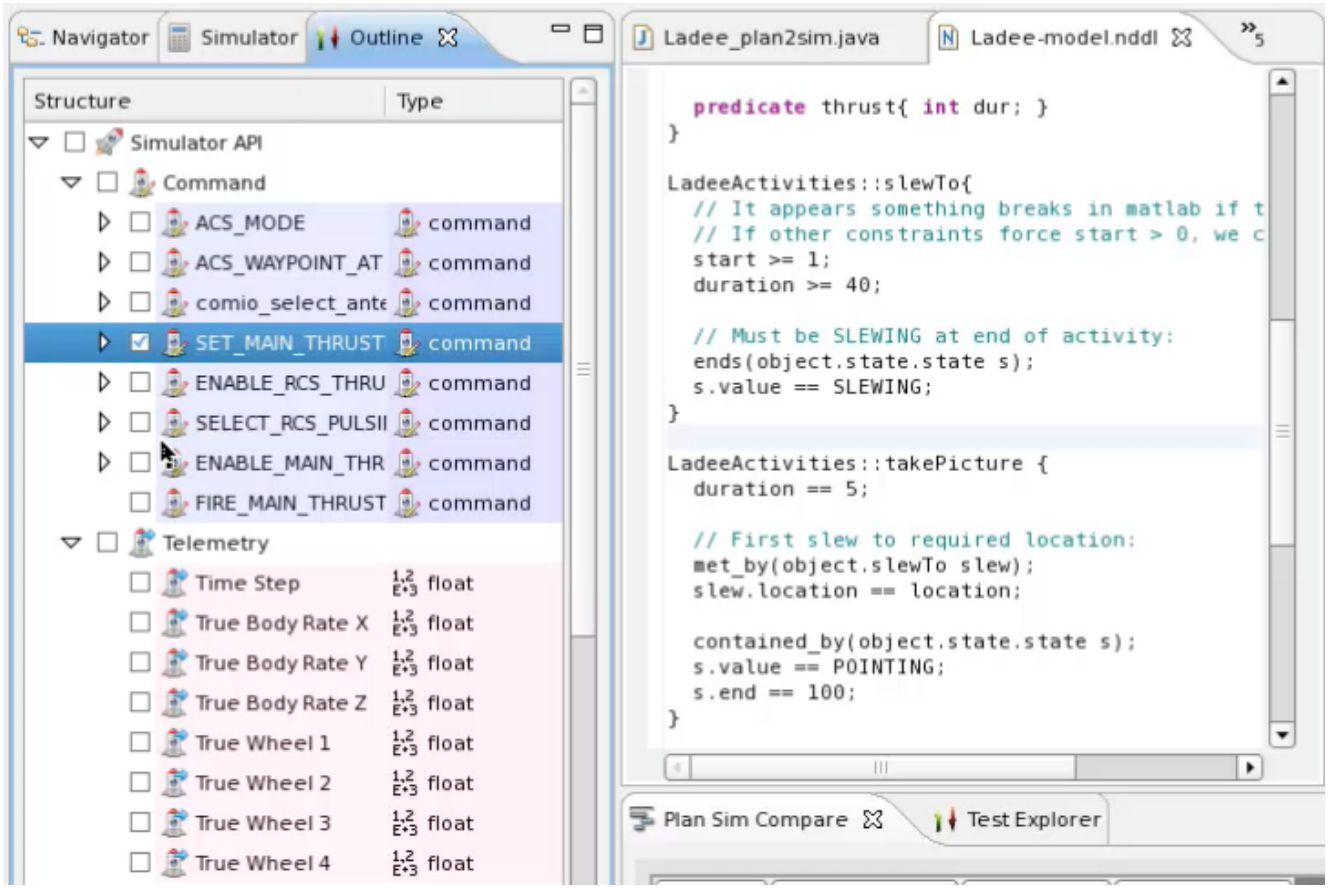


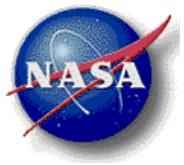
IMDE Architecture: Abstraction Editor

- Document relationship between model (actions, states, parameters, values) and spacecraft (simulation) command and data specification.
- Provides human-readable traceability between systems.
- Formal representation of abstractions is input to Refinement/Abstraction engine.

IMDE Architecture: Abstraction Editor

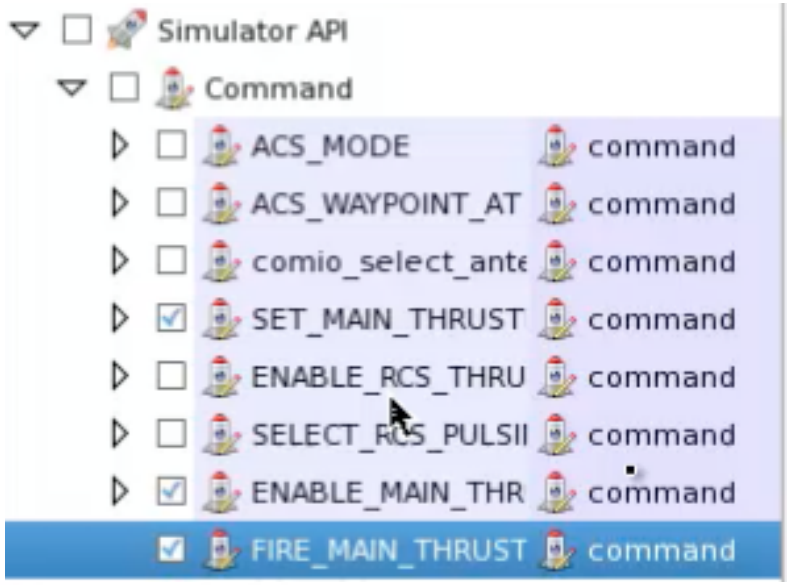
- Commands and Telemetry Browsing:



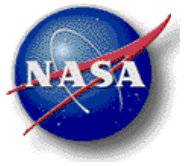


IMDE Architecture: Abstraction Editor

- Refining Planner actions into commands:



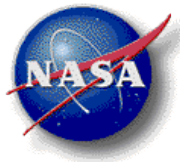

```
public void refinethrust(
        @Input("LadeeActivities.thrust") PlanActivity planActivity,
        @Output("SET_MAIN_THRUSTER_DURATION") SimActivity simActivity(
        @Output("ENABLE_MAIN_THRUSTER") SimActivity simActivity1,
        @Output("FIRE_MAIN_THRUSTER") SimActivity simActivity2)
{
```



IMDE Architecture: Abstraction Editor

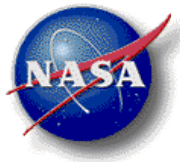
- Abstracting spacecraft telemetry into states:

```
public boolean abstractBodyRate(
        @Input("True Body Rate X") String val1,
        @Input("True Body Rate Y") String val2,
        @Input("True Body Rate Z") String val3,
        List <DSAParameter> params)
{
    double d1 = Math.abs(Double.parseDouble(val1));
    double d2 = Math.abs(Double.parseDouble(val2));
    double d3 = Math.abs(Double.parseDouble(val3));

    double cutoff = 0.001;

    // d3 settles to -0.001, not 0
    boolean moving =  (d1 > cutoff || d2 > cutoff); // || d3 > cutoff)
    boolean val =  (d1 > cutoff || d2 > cutoff || d3 > cutoff);
    params.add(new DSAParameter("value", "" + (moving ? "SLEWING" : "P

    return true;
}
```



IMDE Architecture: Abstraction Refinement Engine

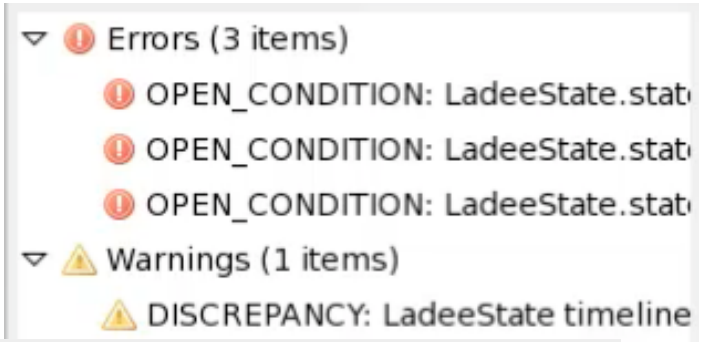
- Refinement:
  - Transforms plan and initial state into simulation input and command sequence.
  - Transformation defined by 'inverting' abstractions created in Abstraction Editor.
- Abstraction:
  - Transforms simulation output into 'predicted' plan states.
  - Transformation uses abstractions created in Abstraction Editor (for data to states only).

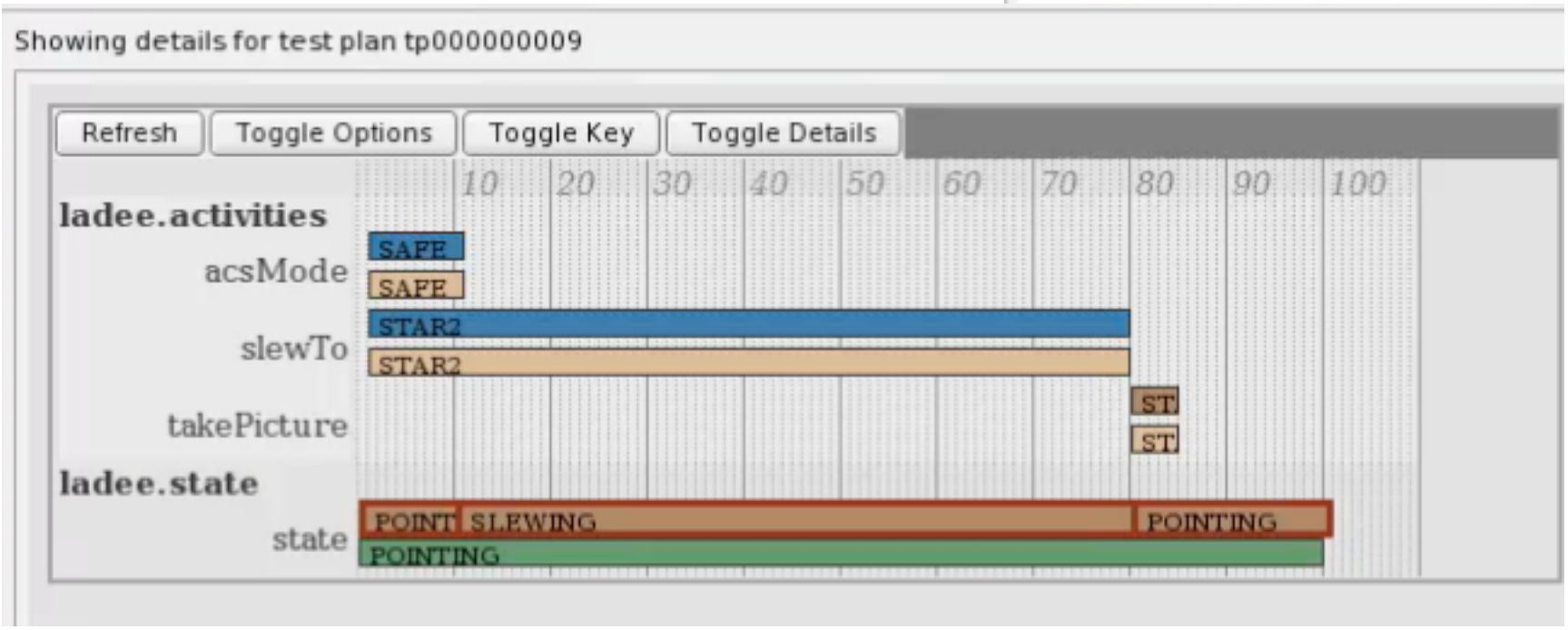


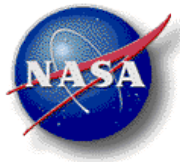
IMDE Architecture: Validator

- Compare states as generated by planner to those abstracted from simulation.
- Differences indicate a discrepancy between plan model and simulation behavior.
- Can also report simulator errors, constraint violations not caught by simulator, etc.

IMDE Architecture: Validator

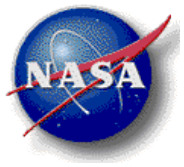






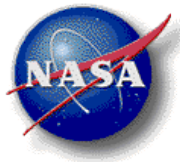
IMDE Architecture: Validator

✕	tp000000003	NA	NA	Yes	40	LadeeActivities. location	STAR2	10	LadeeActivit mode	FINE_POINTING
✕	tp000000007	NA	NA	Yes	40	LadeeActivities. location	STAR2	1	LadeeActivit mode	SAFE
✕	tp000000011	NA	NA	Yes	40	LadeeActivities. location	STAR2	10	LadeeActivit mode	SAFE
✕	tp000000015	NA	NA	Yes	40	LadeeActivities. location	STAR2	1	LadeeActivit mode	FINE_POINTING
✖	tp000000000	NA	NA	Yes	60	LadeeActivities. location	STAR2	1	LadeeActivit mode	FINE_POINTING
✖	tp000000004	NA	NA	Yes	60	LadeeActivities. location	STAR2	10	LadeeActivit mode	FINE_POINTING
!	tp000000008	3	0	Yes	60	LadeeActivities. location	STAR2	1	LadeeActivit mode	SAFE
!	tp000000012	3	0	Yes	60	LadeeActivities. location	STAR2	10	LadeeActivit mode	SAFE
✖	tp000000001	NA	NA	Yes	80	LadeeActivities. location	STAR2	1	LadeeActivit mode	FINE_POINTING
✖	tp000000005	NA	NA	Yes	80	LadeeActivities. location	STAR2	10	LadeeActivit mode	FINE_POINTING
✓	tp000000009	0	0	Yes	80	LadeeActivities. location	STAR2	1	LadeeActivit mode	SAFE
!	tp000000013	3	0	Yes	80	LadeeActivities. location	STAR2	10	LadeeActivit mode	SAFE
✖	tp000000002	NA	NA	Yes	90	LadeeActivities. location	STAR2	1	LadeeActivit mode	FINE_POINTING



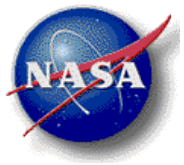
IMDE Architecture: Fixer

- If a command in the slew fails to execute:
  - Planner model could be missing one or more conditions the simulator enforces (e.g. thermal).
  - Planner model conditions could be wrong.
  - Abstraction of commands to action could be wrong (e.g. sequence isn't right, duration wrong).



IMDE Architecture: Fixer

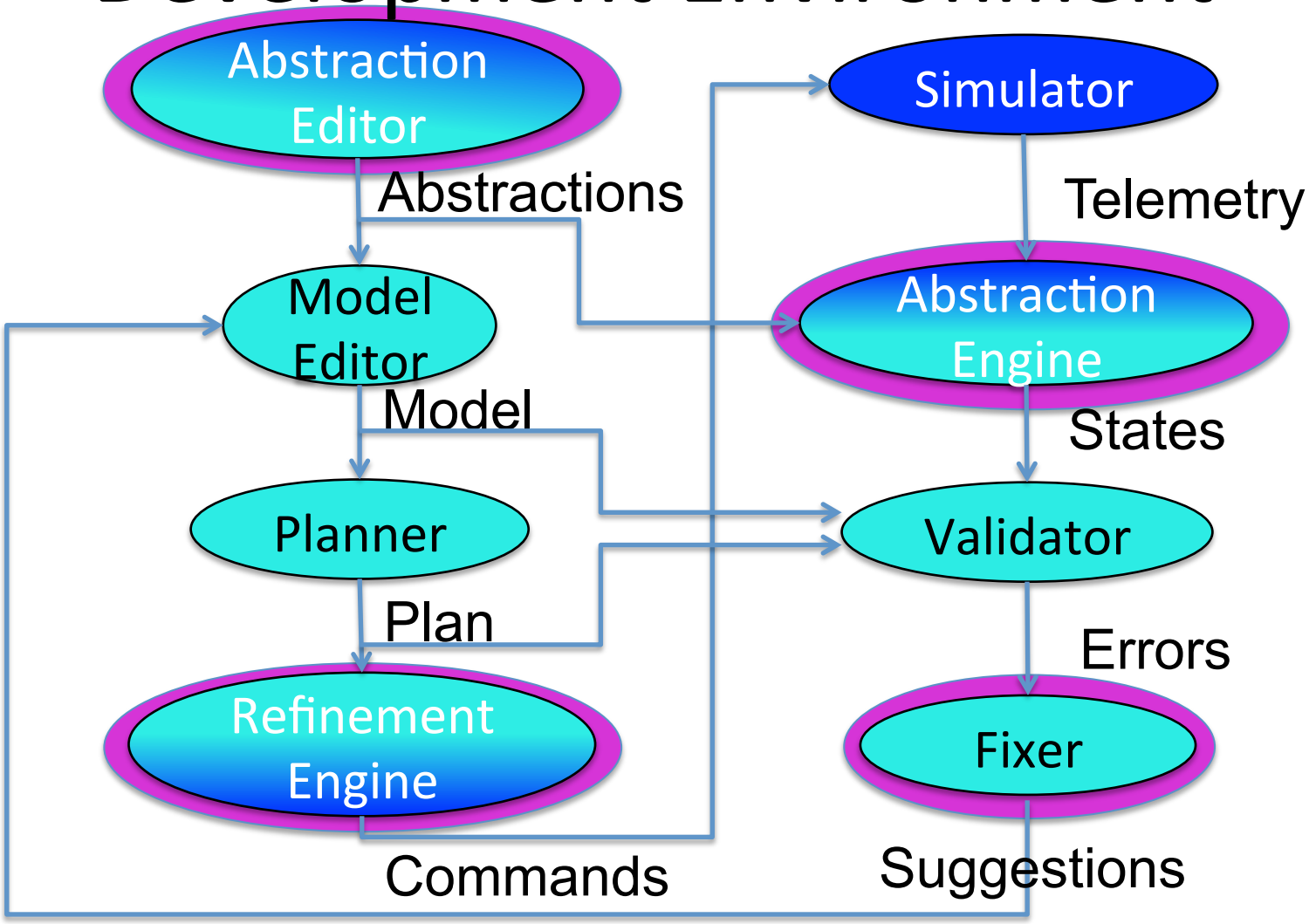
- If there is a discrepancy between planned and simulated states, or the planned action is incompatible with the simulated states:
  - Planner model effects could be missing (e.g. thermal effects).
  - Planner model effects could be wrong.
  - Abstractions of telemetry to states could be wrong (e.g. body rate threshold for pointing could be wrong).

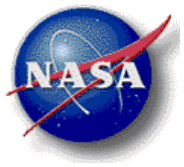


IMDE Architecture: Fixer

- Given a discrepancy between planner model and simulator
  - Identify part of model that is to blame.
  - Ideally: Suggest changes to model that will address the problem.
  - Problem: the error could be almost anywhere!

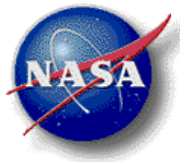
Integrated Model Development Environment





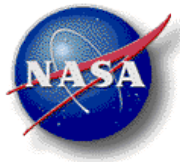
IMDE Algorithms

- All abstractions either
  - Map a domain of a telemetry variable into a domain of smaller cardinality
    - $f(X) \Rightarrow Y$ s.t. $|X| > |Y|$
    - Special cases: eliminate element of a discrete domain, map reals to integers, map integers to positive integers
  - Map n variables to m<n variables
    - $f(x_1 \ldots x_n) \Rightarrow \{y_1 \ldots y_m\}$
    - Cardinality must still be reduced, e.g. $|X_1||X_2|\ldots|X_n| < |Y_1||Y_2|\ldots|Y_m|$
    - Special cases: eliminate variable



IMDE Algorithms

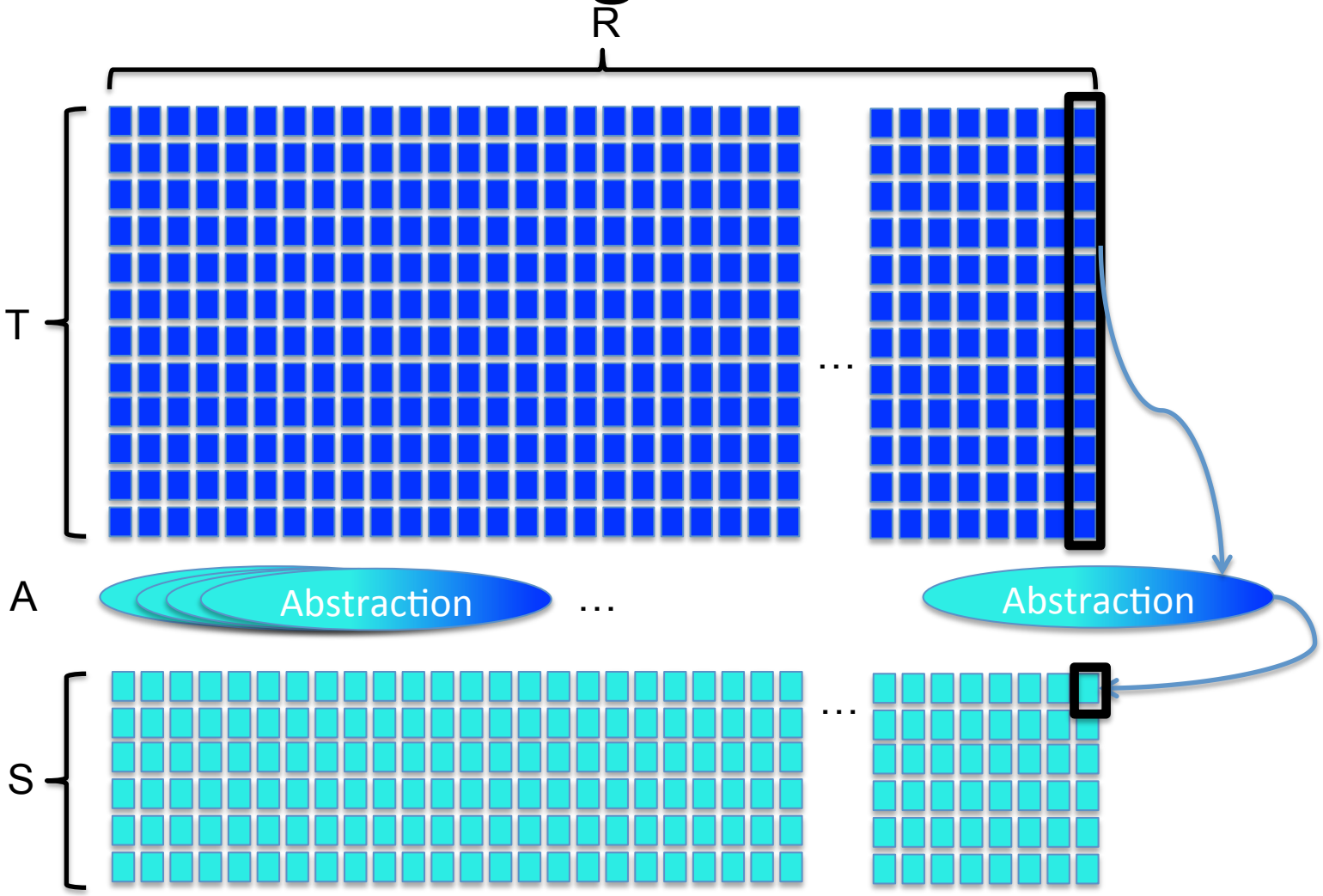
- Planner / plans
  - N actions
  - P_n action conditions / effects for action n
  - S states
  - M state instances in a plan
- Simulator
  - C_n commands in refinement of action n
  - T telemetry items
  - R values for each item per run
- Model
  - A abstractions
- We expect M << TR

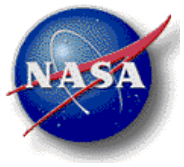


IMDE Algorithms

- Generate refinement from plan
  - Σ C_n (N actions, C_n commands in refinement of action n)
- Generate abstraction
  - TRA + 2SR (T telemetry items, R values for each item per run, A abstractions, S state types)
  - First pass is to generate states: for all R, for all abstractions A, each abstraction uses at most T telemetry items; writes at most S states holding at each instant R instants. This gets us runtime TRA+SR.
  - 2nd pass is to determine start / end times of states; this is another SR.
  - (There are important assumptions about the form of the abstractions i.e. they only use values at one time tic)

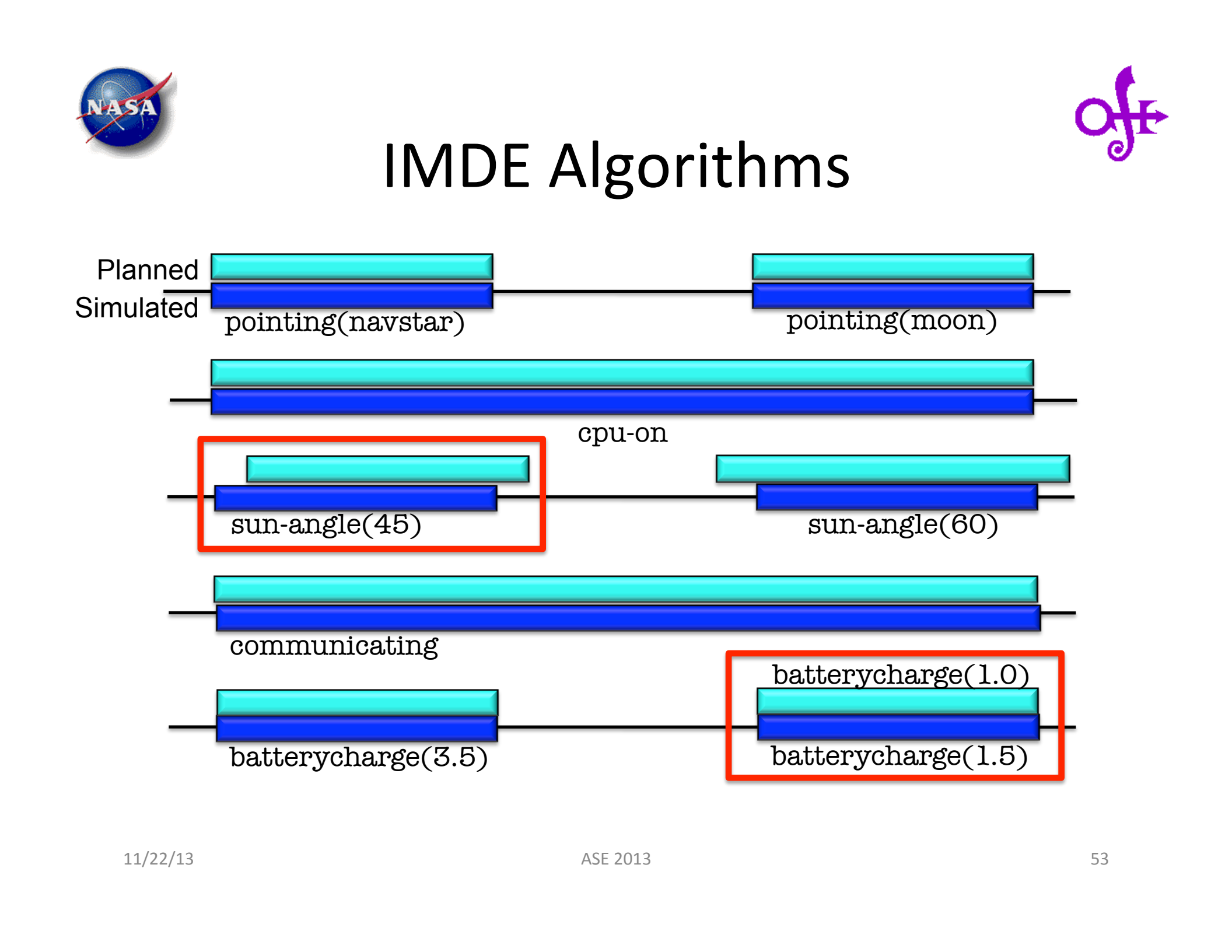
IMDE Algorithms



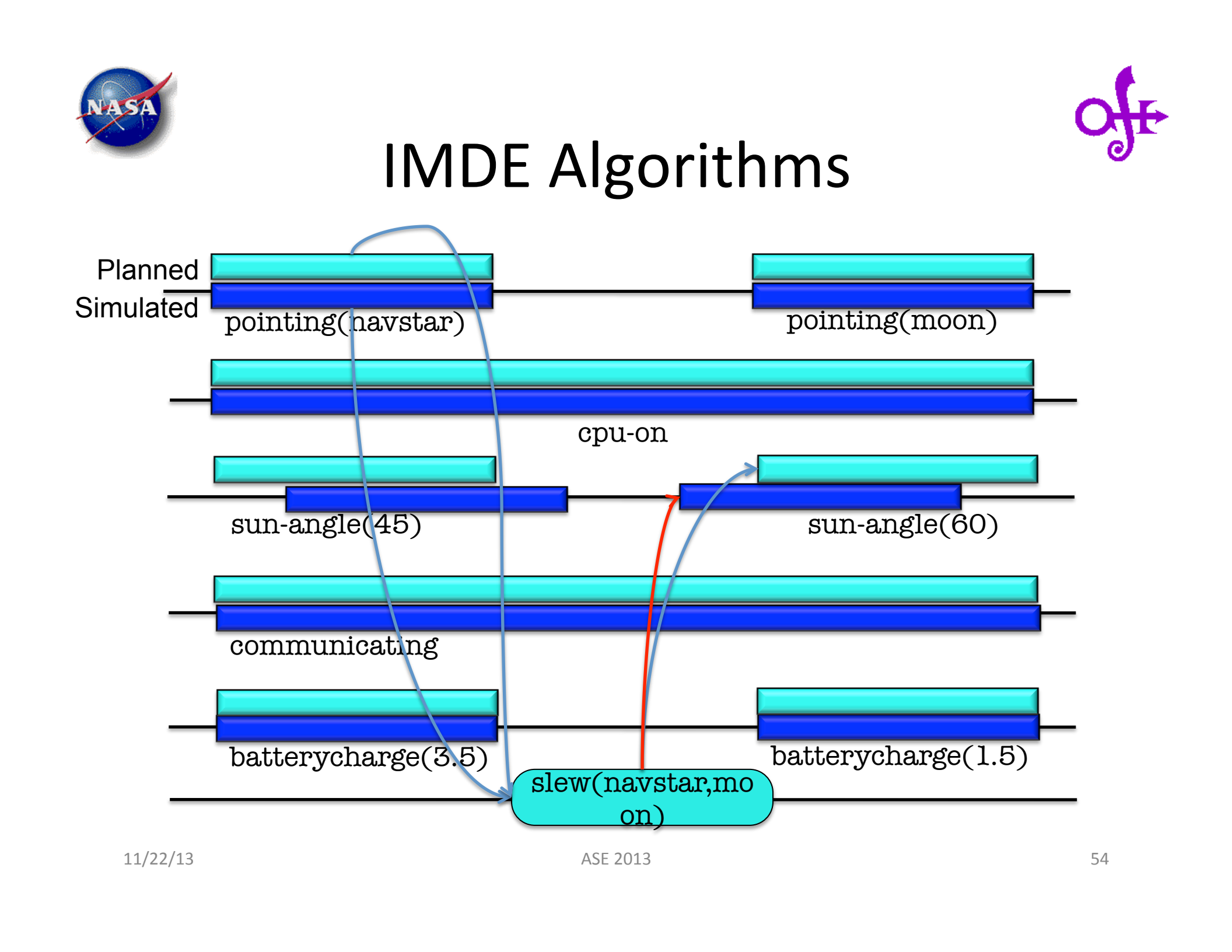


IMDE Algorithms

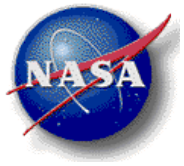
- Generated warnings
  - M^2 (M distinct state instances in plan, check to see if simulated start / end time match compared to planned start / end times; also check to see if simulated states internally consistent)
- Generate discrepancies
  - $M(\Sigma P_n)$ (N actions, P_n action conditions / effects)
  - For each condition/effect of an action, may need to search the M states to match conditions / effects



IMDE Algorithms

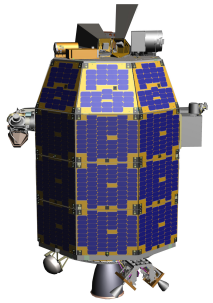


IMDE Algorithms



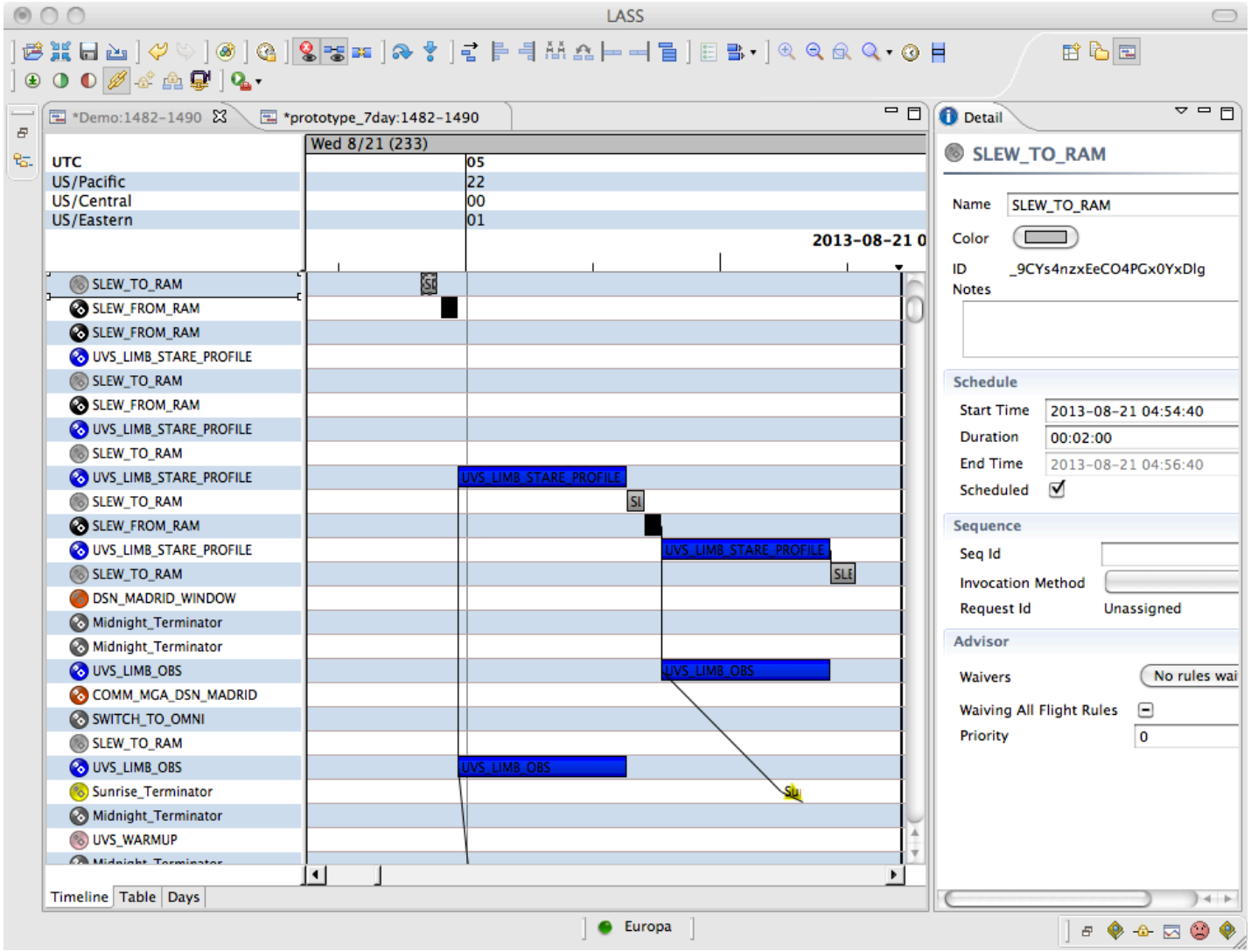
A Few Words About LADEE

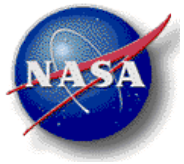
- How would this approach need to scale for LADEE?
  - ~600 Commands
  - ~25000 Telemetry / data
  - 122 Activities
  - 27 States
  - 21 Numerical Resources



- The LADEE planner model has ~ 12000 lines.
- Simulation data produced at 10Hz (cycles / second)

A Few Words About LADEE

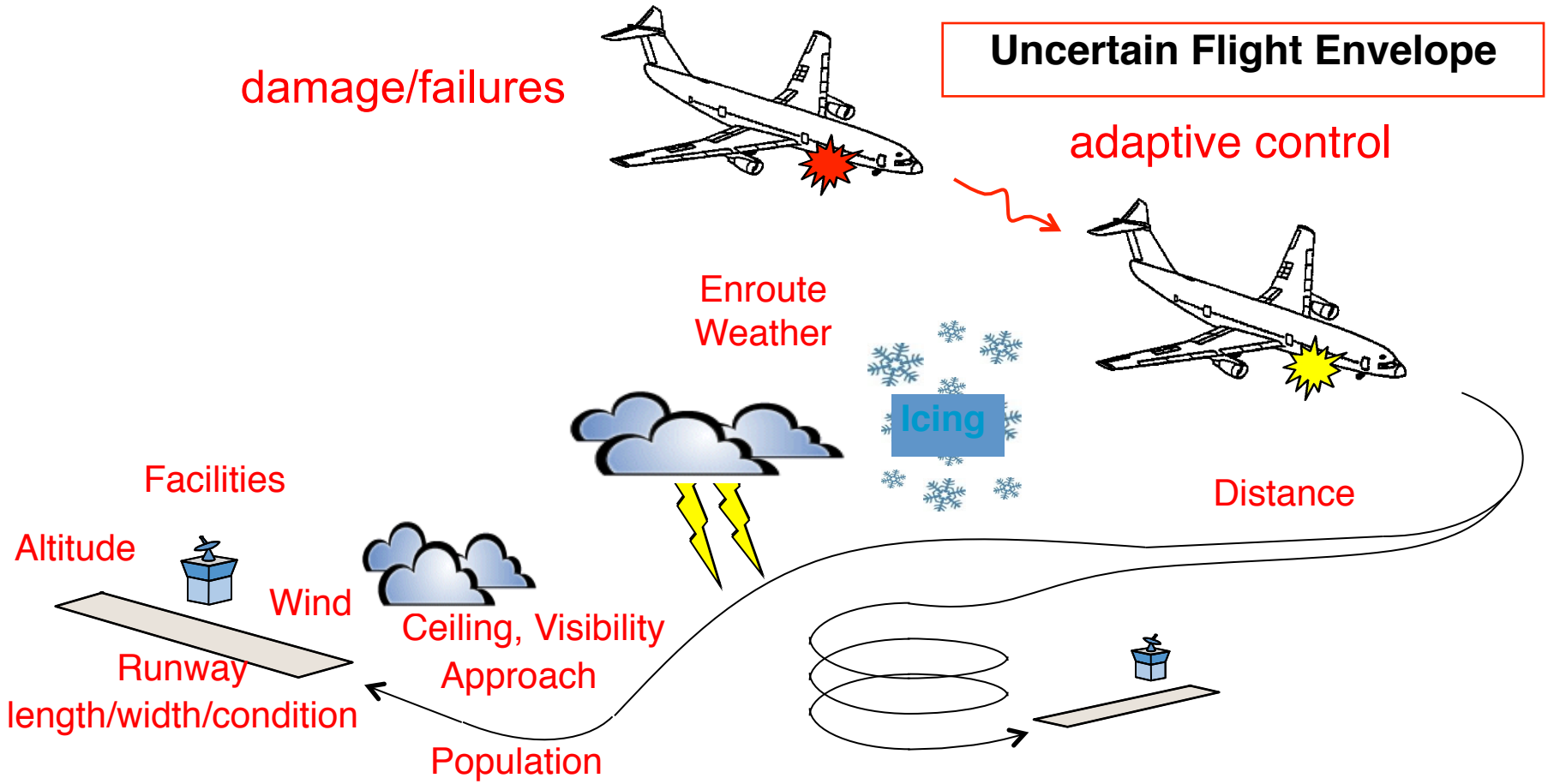




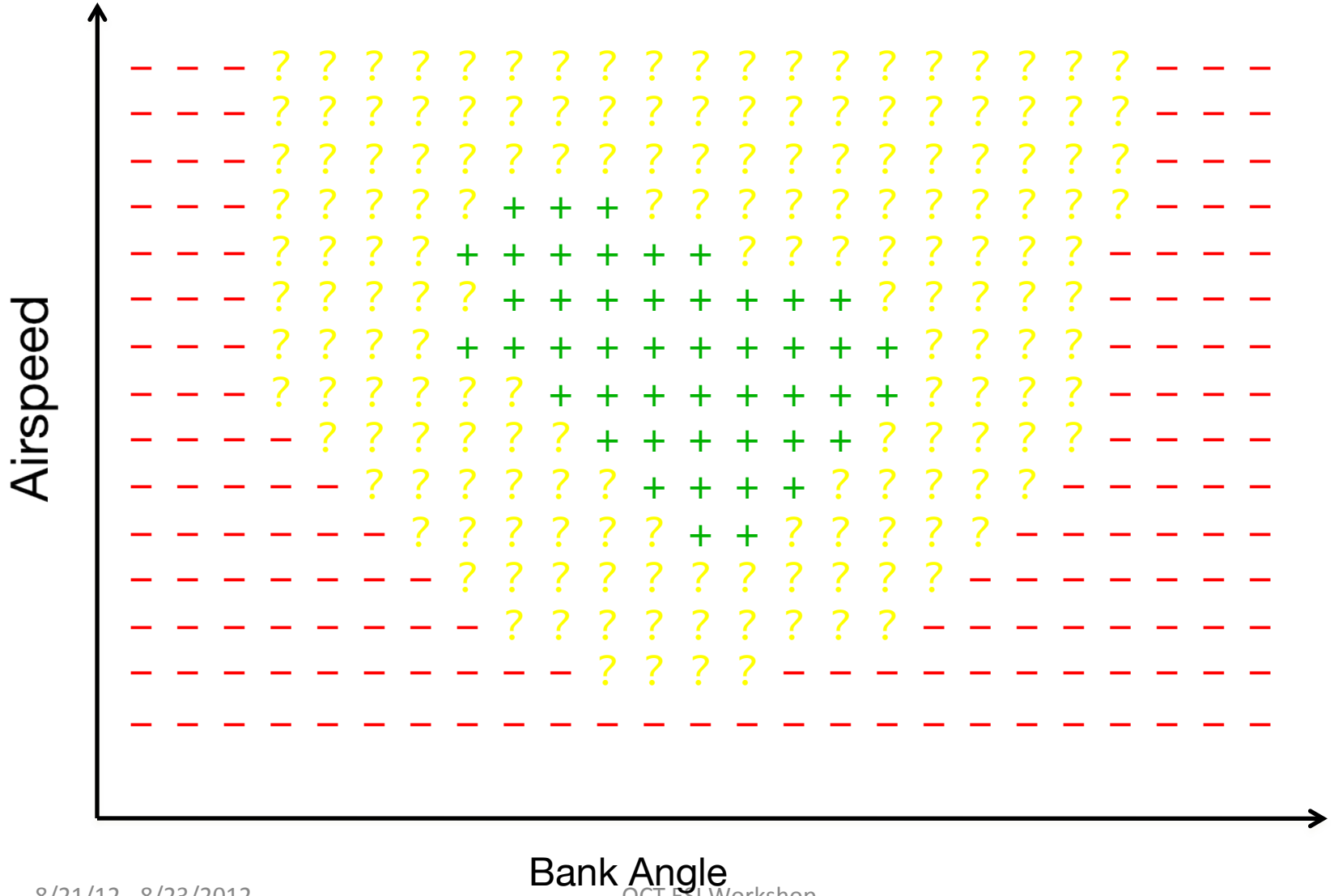
Future Work

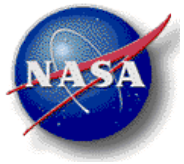
- How can errors be identified and fixed for different modeling language features, such as uncertainty, parameter functions, and decompositions? (e.g. learning)
- How can the architecture be adapted to suggest changes for plan quality?
- How can we take advantage of white box simulators? auto-generate refinements to sim commands? auto-fill model? [13]
- See [14] for tools for authoring abstractions.

Future Work


damage/failures
Uncertain Flight Envelope
adaptive control
Enroute
Weather
Icing
Facilities
Altitude
Wind
Runway
length/width/condition
Ceiling, Visibility
Approach
Population
Distance

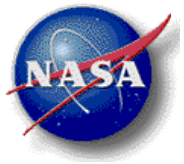
Conclusions and Future Work





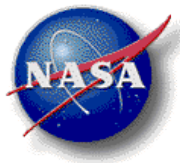
Future Work

- Future mission planning systems could explicitly incorporate uncertainty:
  - (Limited) Contingency Planning [10]
  - Optimal policy generation for (Partially Observable) Markov Decision Processes ((PO)MDPs) [11]
  - Conformant planning [12]
  - 'Generate' using classical planning plus 'test' using simulation [15]
- This will make the V&V challenge harder.



The Takeaway

- Verification and Validating of planning models is a challenge.
- Extending the notion of the traditional software IDE as our solution:
  - Expose simulation API in plan model editor.
  - Providing 'good UI'.
  - Document relationship between elements (abstractions) enables automation:
    - Transform plans to simulation inputs.
    - Transform simulation outputs.
    - Comparison with plans to generate discrepancies and warnings.



References

[1] Smith, B., Feather, M., and Muscettola, N. Challenges and Methods in Testing the Remote Agent Planner. Proceedings of the Artificial Intelligent Planning and Scheduling Conference, 2000.

[2] Cichy, B., Chien, S., Schaffer, S., Tran, D., Rabideau, G., Sherwood, R. Validating the Autonomous EO-1 Science Agent In: *Int'l Workshop on Planning and Scheduling for Space*. 2004.

[3] Vaquero, T., Romero, V., Sette, F., Tonidandel, F., Reinaldo Silva, J. ItSimple 2.0: An Integrated Tool for Designing Planning Domains. In *Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling*, 2007.

[4] Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E. Validation and Verification Issues in a Timeline-Based Planning System. *Knowledge Engineering Review*, 25(3): 299-318, 2010.

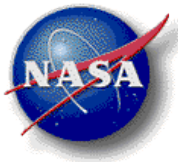
[5] Raimondi, F., Pecheur, C., Brat, G. PDVer, a Tool to Verify PDDL Domains. Proceedings of the ICAPS 2009 VVPS Workshop

[6] Howey, R., Long, D., & Fox, M. (2004). VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, pp. 294–301, Washington, DC, USA. IEEE Computer Society.

[7] Brat, G., Gheorghiu, M, , Giannakopoulou, D., "Verification of Plans and Procedures," In *Proc. of IEEE Aerospace Conf.*, 2008.

[8] B. Clement, J. Frank, J. Chachere, T. Smith and K. Swanson. *The Challenge of Grounding Planning in Simulation in an Interactive Model Development Environment.* Proceedings of the Knowledge Engineering for Planning and Scheduling Workshop, in conjunction with the 21st International Conference on Automated Planning and Scheduling, 2011.

[9] J. Frank, B. Clement, J. Chachere, T. Smith and K. Swanson. *The Challenge of Configuring Model-Based Space Mission Planners.* Proceedings of the 7th International Workshop on Planning and Scheduling for Space, 2011.



References

[10] N. Meuleau and D. Smith. Optimal Limited Contingency Planning. Proceedings of the Conference on Uncertainty in Artificial Intelligence, 2003.

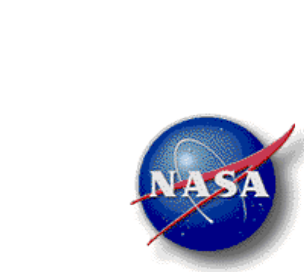
[11] Mausam, A. Kolobov. Planning with Markov Decision Processes: An AI Perspective. Morgan and Claypool Publishers, 2012.

[12] Hoffmann, J., & Brafman, R. (2006). Conformant planning via heuristic forward search: A new approach. Artificial Intelligence, 170(6-7), 507–541

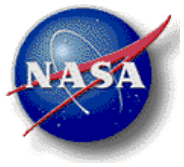
[13] Schumann J., Gundy-Burlet K., Păsăreanu C., Menzies T., Barrett, A. Tool Support for Parametric Analysis of Large Software Simulation Systems. Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering.

[14] Bell, S., Kortenkamp, D., and Zaientz, J. A Data Abstraction Architecture for Mission Operations. In *Proc. of the International Symposium on AI, Robotics, and Automation in Space*, 2010.

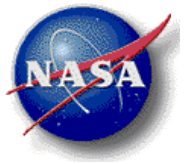
[15] M. Fox, D. Long and D. Magazzeni (2012) "Plan-based Policies for Efficient Multiple Battery Load Management", Journal of Artificial Intelligence Research, Volume 44, pages 335-382



Thank You!

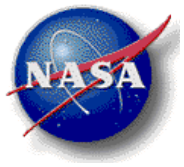


BACKUP SLIDES



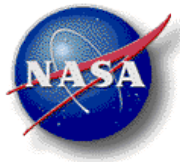
Outline

- Challenge
- Automated Planning Systems
  - Example
  - Mission Planning
  - Model-Based Planning
  - Model Debugging
- Previous and Related Work
- Integrated Model Development Environment (IMDE)
  - Concept of Operations
  - IMDE Architecture
  - IMDE Algorithms
  - A Few Words about LADEE
- Conclusion and Future Work



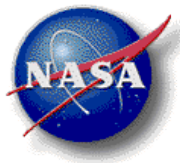
Example (Redux)

- Simulate a plan solution with a single **`slew`** to Earth and a goal **`(pointing Earth)`**.
  - Construct simulation commands from plan
  - Construct initial state from plan
  - Select other initial conditions not part of plan initial state
- Simulation output shows no change in attitude.
  - Must be an error in the model or abstractions because the goal fails.
- No other output suggests anything went wrong.
  - No information to identify error!
- Simulate more slews with different initial states.
  - Some simulations show successful slews, and others show no slew.
  - The most noticeable pattern is that whenever the simulator **`cpu-on`** is zero, the slew is not executed.
- Fix the model:
  - add the **`cpu-on`** simulator state variable to planning model
  - add a **`cpu-on`** precondition to the **`slew`** activity OR
  - add a **`turnOnCpu()`** simulator command to the refinement OR
  - add the **`turnOnCpu()`** simulator command to the planning model and add a temporal constraint that **`slew`** be preceded by **`turnOnCpu()`**.



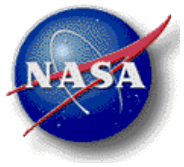
Model-Based Planning

- Example: spacecraft attitude change (slew)
  - pointing – a state variable indicating spacecraft orientation or change of orientation
    - Takes as parameter one of a (possibly infinite) set of directions, attitudes, or poses
  - communicating – a state variable indicating whether or not the spacecraft is communicating with a ground antenna
  - batterycharge – a state variable (resource) indicating battery state of charge
    - numerical value (hours of charge remaining)
  - sunangle – a state variable tracking where the sun is relative to the spacecraft
    - Numerical value (vector from spacecraft to sun)



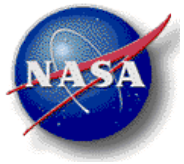
Motivation (redux)

- Model checking does not help validate model in system context.
  - Prior plan verification/validation work
    - verifies constraints/properties encoded in another language,
    - ensures against *foreseeable* problems,
    - verifies self-consistency, or
    - validates plan as a solution.
  - We are interested in checking against system behavior as represented by simulation.
  - Simulation can be used to check simultaneously for all problems, many of which
    - may not be expressible in the model checking language or
    - may be *unforeseeable*.



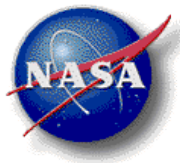
Motivation (redux)

- NOT auto-generating planning model from simulation, because
  - The planning domain modeler does not build the simulation.
  - The simulation is usually a black-box.
  - Even if it weren't it is too difficult to transform the white-box model into our planning representations.
  - And the simulator can't be used for up-front planning. (It is not designed to be used this way).



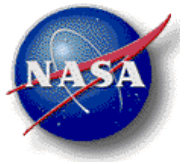
Integrated Model Development Environment

- Instead of testing all possible plans, start with a single model element.
- For example, simulate **`slew`** for different initial states and parameterizations.
- Then simulate all two-activity plans (**`slew`** and another activity) with different temporal relations.
- Continue while growing numbers of activities.
- May be able to avoid plans like
  `turnOnCpu` → **`turnOnCpu`** → **`slew`** . . .
- Likelihood of finding error decreases and confidence in model increases.
- Is there a point where we can stop and claim **`slew`** is correct/ valid?
  No, but we may with assumptions of causal independence.
- If each action model is valid, the entire model is valid!
- Validate model as it is built from scratch!



Technology Foundations

- DAASA
  - S/C Telemetry abstraction authoring and execution
- VAL
  - Validation of plans against specifications
- ItSimple, PRIDE
  - Integrating modeling and simulation
- LOCM
  - Learning plan models from plan traces
- PAGODA
  - Learning plan models from simulations
- SLATE
  - Validating composite actions/behaviors through testing

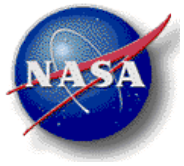


Questioins

- Finite modes and mode transitions in LADEE FSW; does this map to LASS? How?
- Put 'talk takeaway' on one slide (upfront? At end? Both?)
- How to emphasize future of IMDE?
- Didn't talk about mnultiple runs for either abstraction or uncertainty; how to cover that?

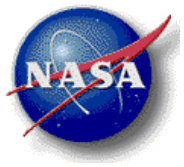
Example

- **<u>Attitude Control System Flight Rule</u>**
- **<u>Rule:</u>** The final attitude of a turn shall place the Sun greater than 45 degrees away from the S/C Y axis (+Y/-Y)
- **<u>Rationale:</u>** Flight software will reject pointing commands that place the Sun within 45 degrees of the Y axis. Rejection of pointing command will result in the slew not being executed and the spacecraft remaining in its current ACS mode and attitude.
- **<u>Criticality:</u>** Category B
- **<u>Mission Phase Dependency:</u>** Cruise, Orbit, Science
- **<u>Commands Affected:</u>** AC_SET_DV_RCS_TGT, AC_SET_INERTIAL_TGT, AC_SET_SCI_NADIR_TGT, AC_SET_SCI_TRACK_TGT
- **<u>Cognizant Individual:</u>** Attitude Control System Engineer
- **<u>Notes:</u>** Flight software will permit slewing through this keepout zone, planning should avoid slewing through this zone if possible.



Model-Based Planning

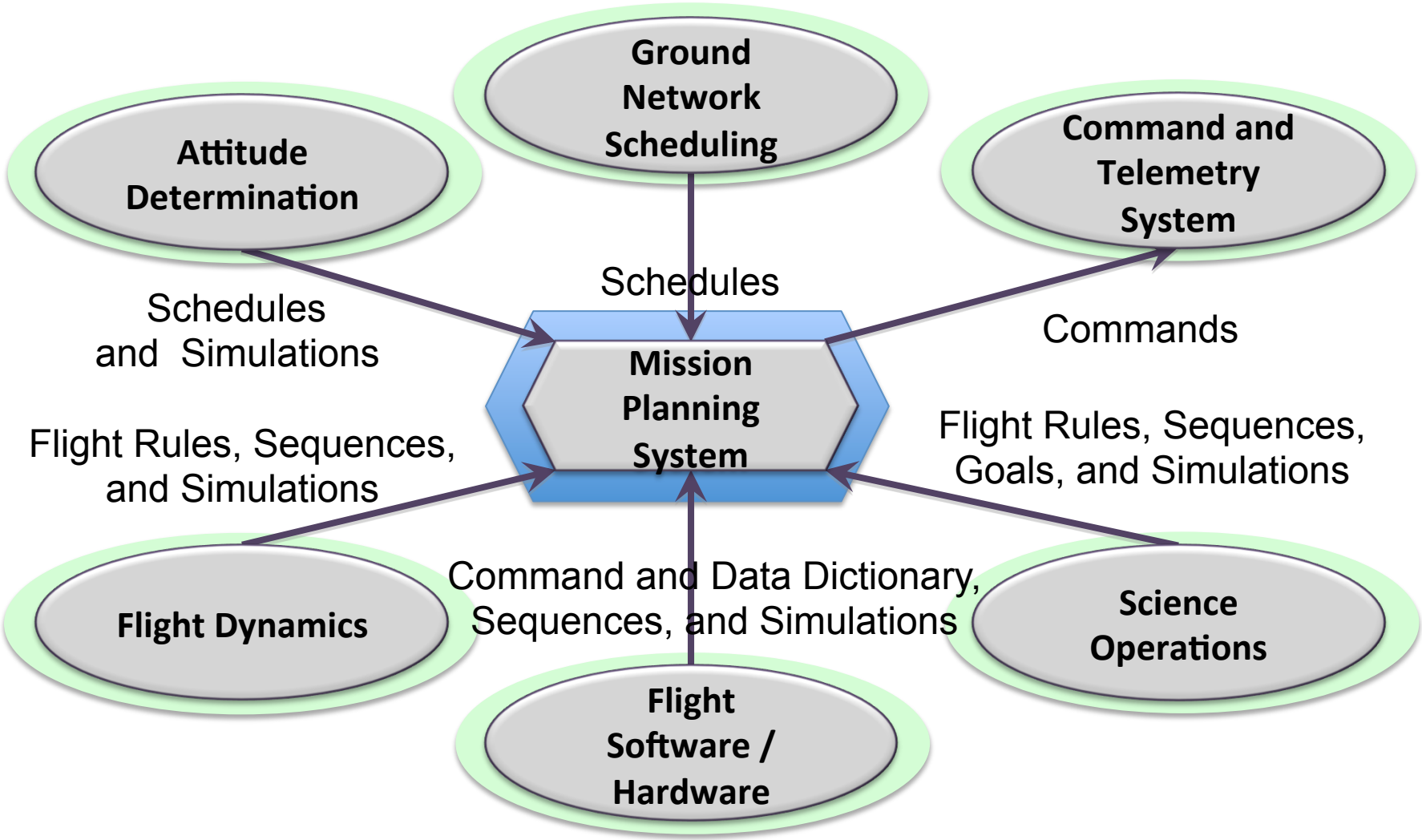
- Model elements in more detail
  - Objects – things in the world
    - E.g. targets, spacecraft components
  - States and Resources
    - Properties of things e.g. available power, mode of system
    - Properties change over time; the term State Variable refers to the history of the changing property
    - Resources are states represented by numbers; resources are depleted and produced by actions
  - Actions
    - Conditions – what must be true, when, and for how long, for an action to have the desired effect
    - Effects – what changes when the action is executed, when, and for how long



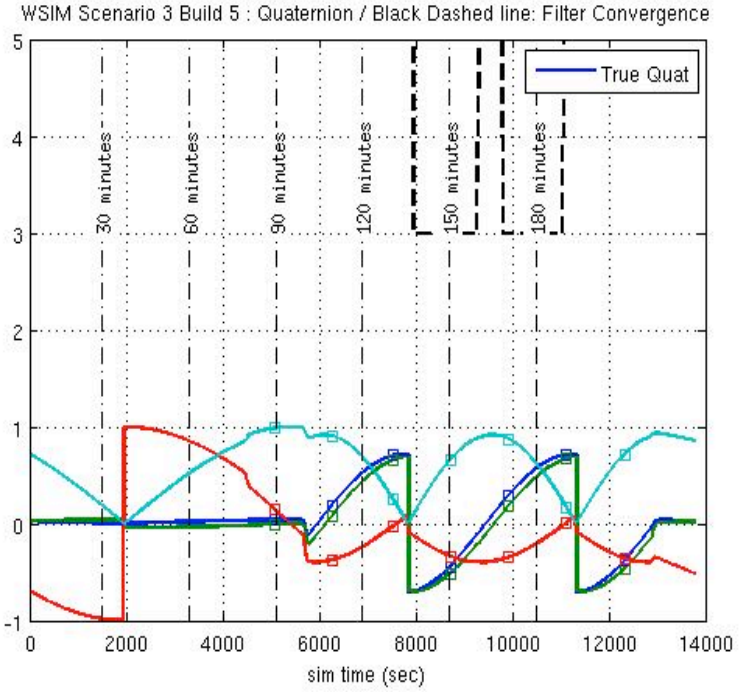
Example

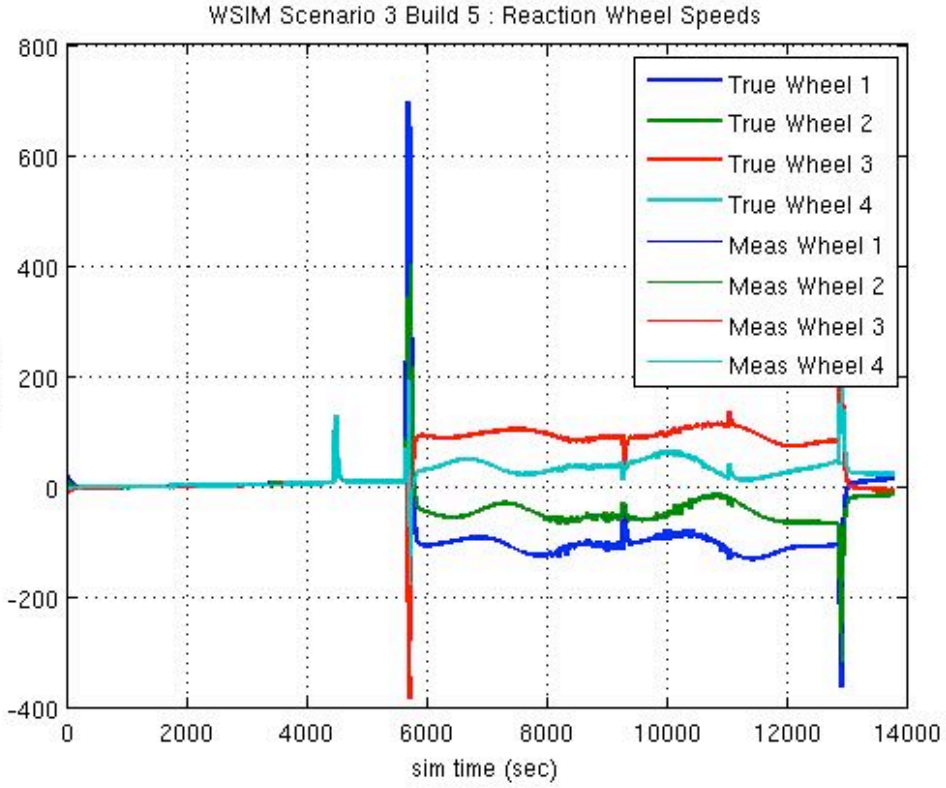
- ACS
  - 4 reaction wheels
    - only 3 used
    - Which ones depends on type of slew
  - Reaction wheel commands
    - Start, change speed, off
  - Reaction wheel data
    - RPM, direction, on, off
- Startracker
  - Sun position
- EPS
  - Subsystems powered
  - Battery state of charge
  - Voltage on channel
- CPU
  - On, Off
- IMU
  - On, off
  - X,Y,Z attitude
  - X,Y,Z rate of change

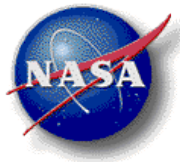
Mission Planning System



Model Debugging

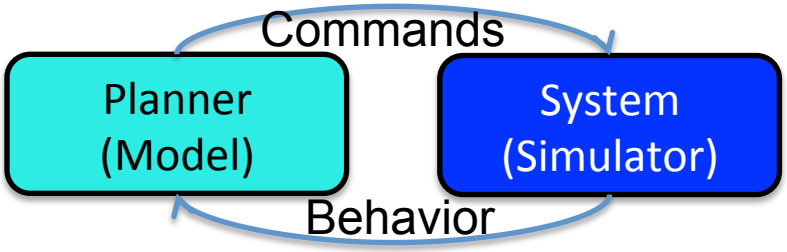


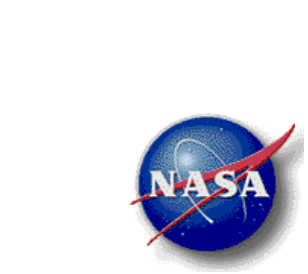




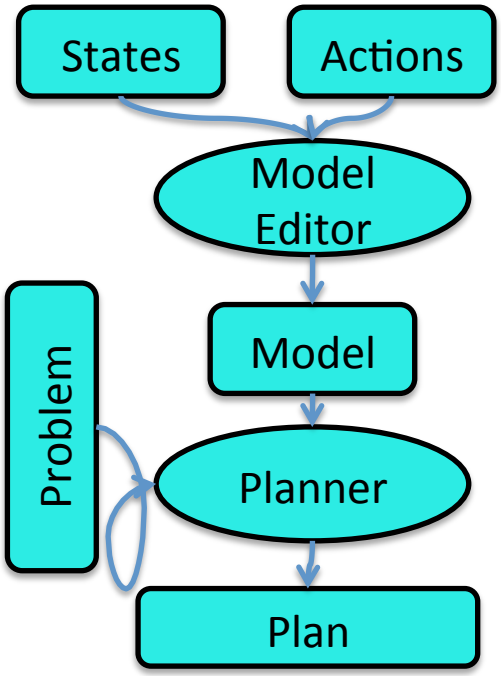
Model Debugging

1. Manually document system commands correspondence to activities
2. Write action and state/resource models using text editor based on available descriptions of system
3. Manually create planning problems and run planner
   - look at solutions to debug model
   - if no solution, figure out why
4. Translate plans to commands; run commands through simulator
5. Translate simulator output out if it did what it expected
6. correct and augment model based on new knowledge

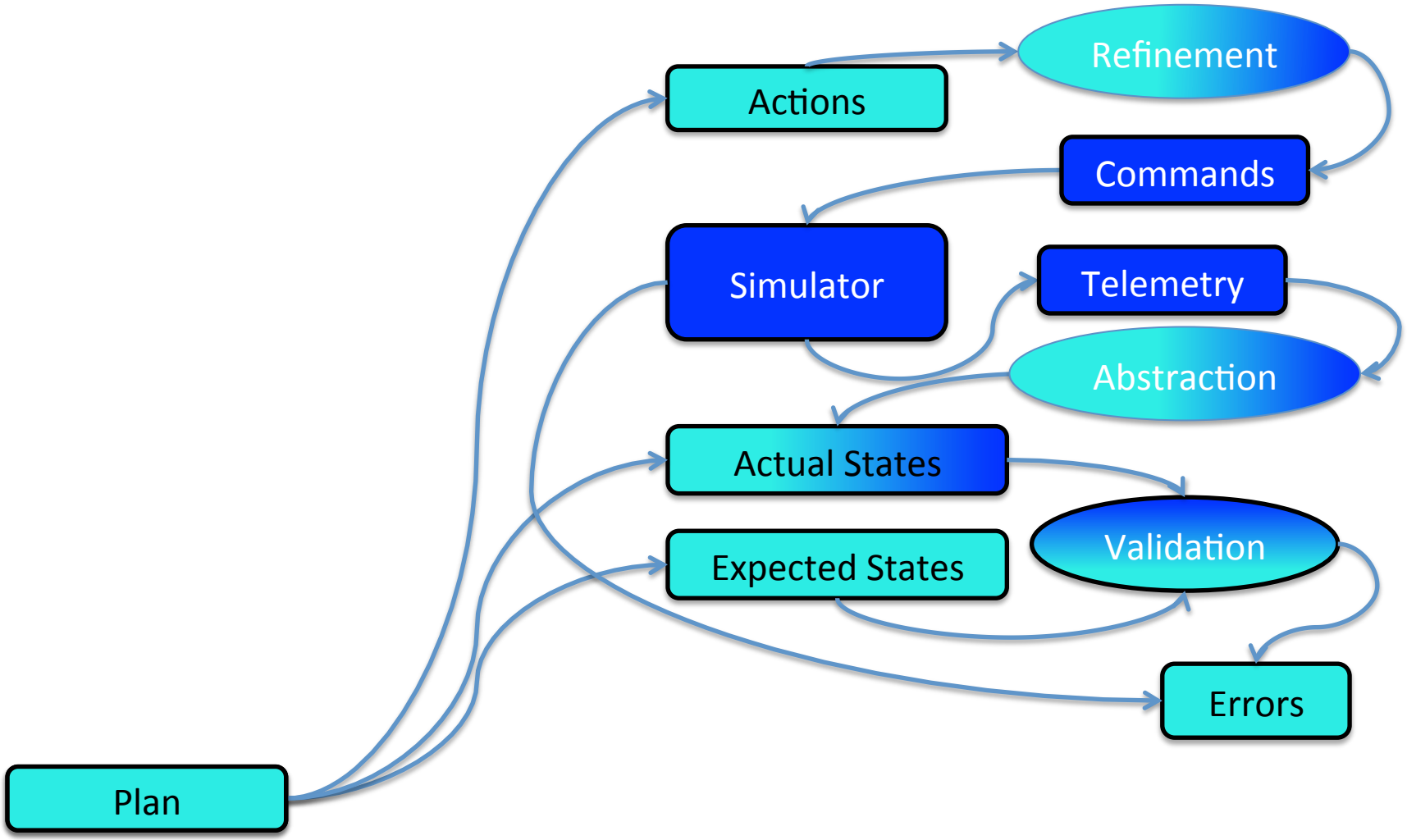




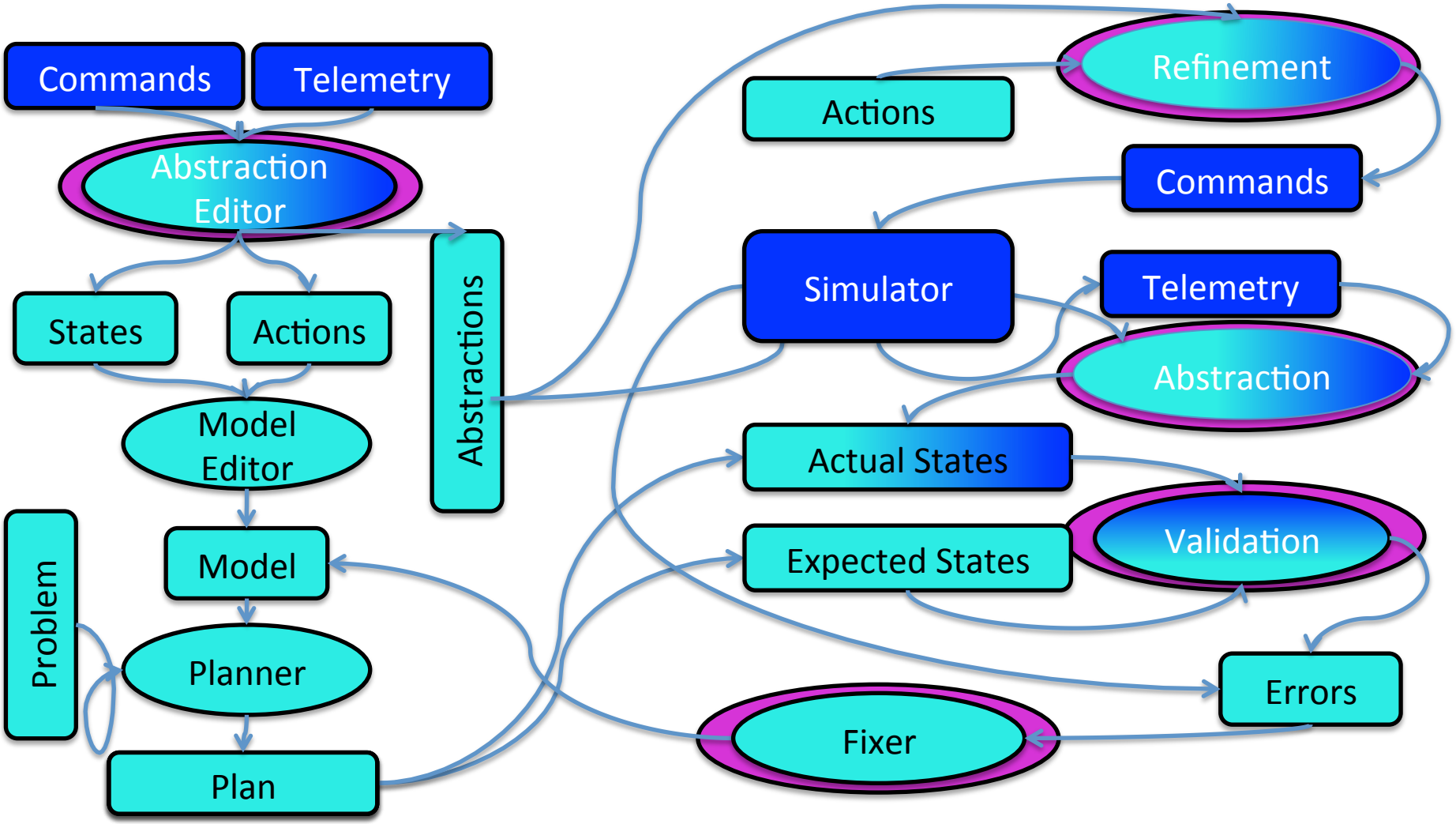
Integrated Model Development Environment



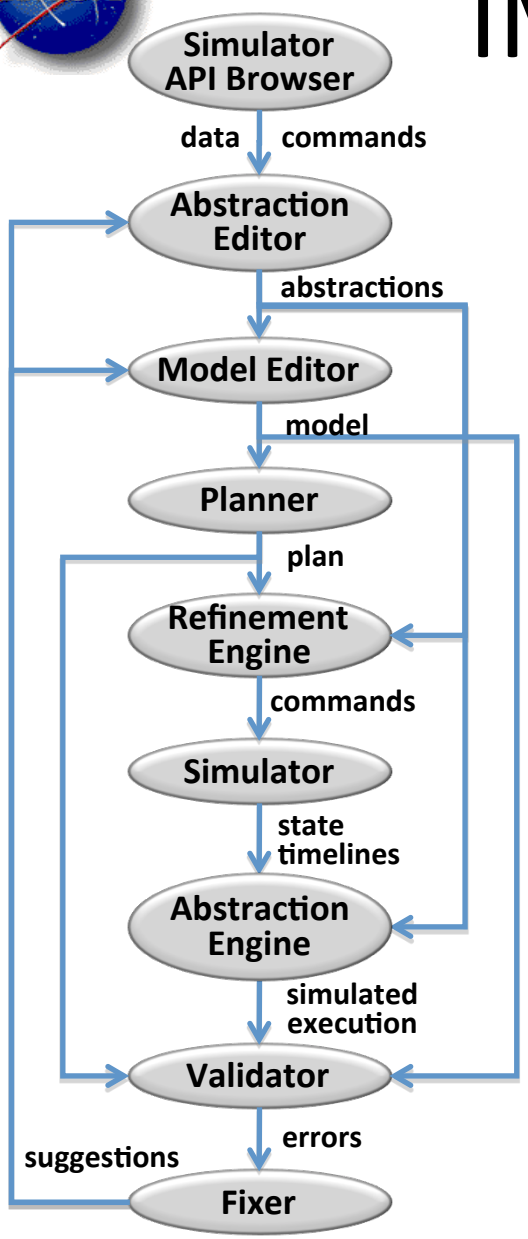
Integrated Model Development Environment



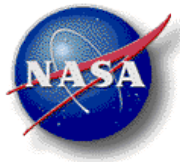
Integrated Model Development Environment



IMDE Architecture



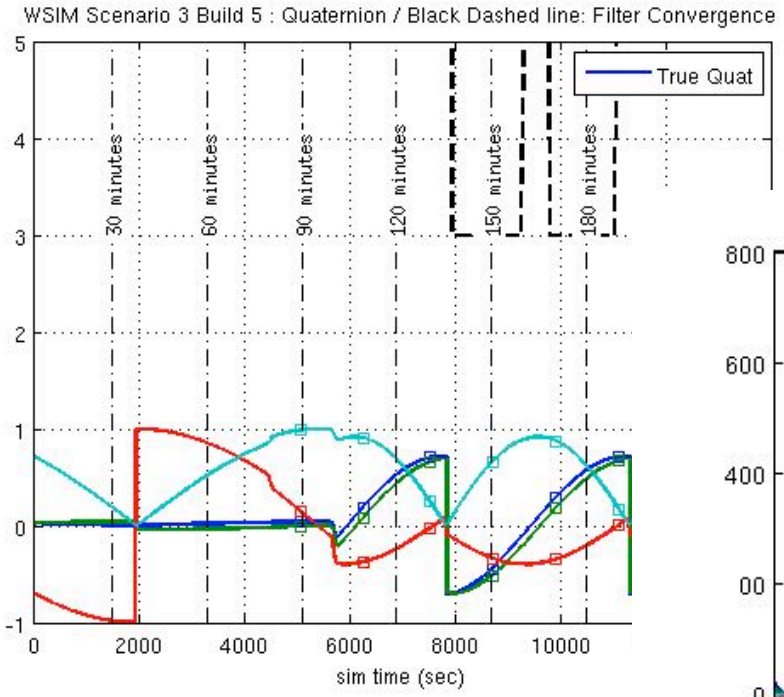
- Abstraction Editor
  - Store relationship between plan and simulation entities
- Model Editor
  - Integrated tightly with simulation
- Refinement Engine
  - Translations between plans and simulations; employs stored abstractions
- Abstraction Engine
  - Translates simulation traces to 'as-executed' plans
- Validator
  - Checks 'as-executed plans' against model
- Fixer
  - Proposes fixes to model

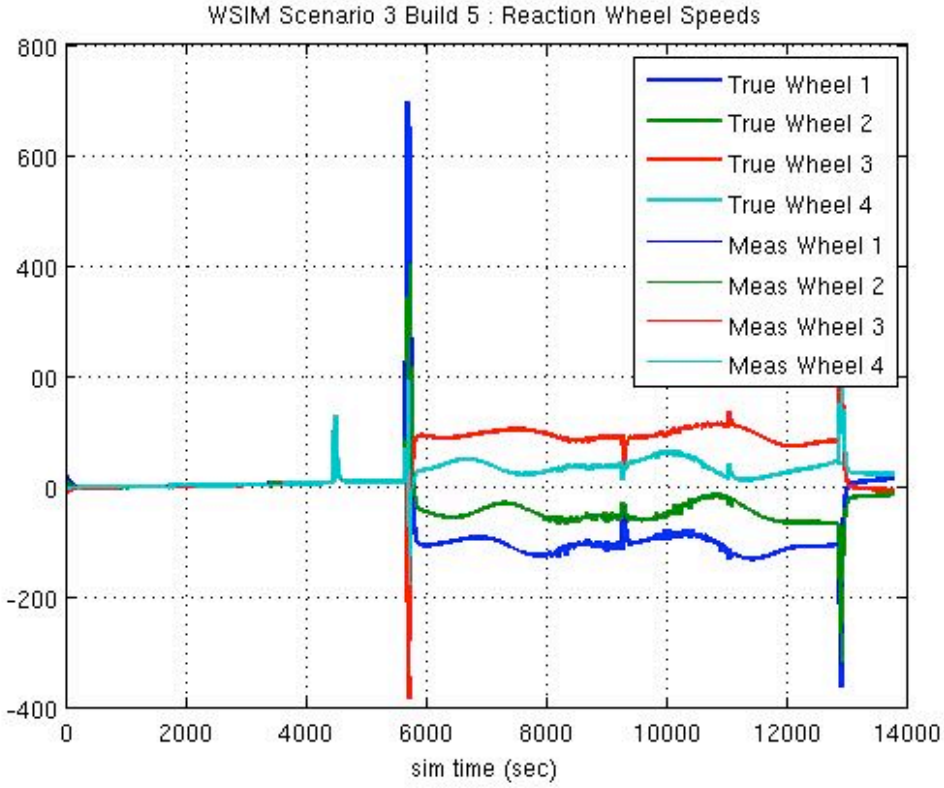


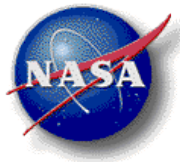
IMDE Architecture: Abstraction Editor

- There are some complexities in abstracting spacecraft telemetry into states
  - Different telemetry rates
    - Mostly spacecraft 'frame' all data but not always
  - State duration and transitions
    - If body rate exceeds cutoff for 1 millisecond does the state transition from pointing to slewing? (YES!)
    - Does this need to be declared in the abstraction? (YES!)

A Few Words About LADEE

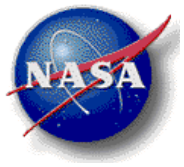






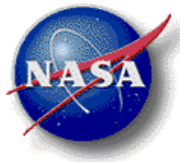
Model Debugging

- What could go wrong?
  - Slew commands not executed.
  - Slew commands executed at wrong time.
  - Slew takes longer or shorter than expected.
  - Slew commands execute, but simulation output shows no change in attitude.
  - Slew commands executed but simulation shows spacecraft pointed wrong. (Or two directions at the same time!)
  - Slew executes when it is not supposed to (e.g. violates sun pointing constraint)
  - Slew works fine, but other simulated behavior is not as expected (e.g. excess power consumption)



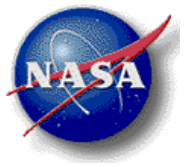
Conclusions and Future Work

- But now you have to V&V:
  - (Probability distributions over) sensor input values
  - (Probability distributions over) possible action outcomes
  - (Probability distributions over) unexpected events
  - (Probability distributions over) plan quality



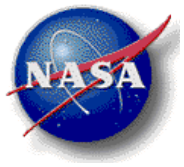
Conclusion and Future Work

- How can a complete but tractable space of test case plans be identified for activity model validation?
- Can a single test case contribute to the validation of multiple model elements?
- Where is the true cause of the error? model? command refinement? data abstraction? simulator? actual system?
- What are the features of a learning problem for classifying an error? numbers of actions of each type? temporal ordering? initial state? simulation state at time of error?
- How can suggested fixes be generated for these errors?



Example

- When planning the slew:
  - How long does a slew take?
  - When is slewing allowed? Not allowed?
  - How does the abstract slew activity map to sequences of spacecraft commands?
  - How do we know when continuous <*x*, *y*, *z*> is pointing to discrete locations? (e.g. the Earth)



Integrated Model Development Environment

- The IMDE automates detection of errors in model based on inconsistencies between the plan and simulation.
- The IMDE automates identification of possible causes of errors from multiple runs.
  - Automate suggested fixes to model.
- More information in [8,9].