Reflecting on Planning Models: A Challenge for Self-Modeling Systems

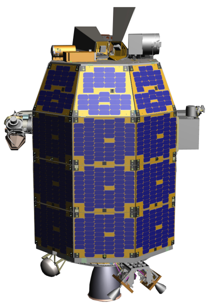
Jeremy Frank
NASA Ames Research Center

Talk Outline

- Motivating Example
  - Planning for cyber-physical systems (a Spacecraft)
  - Command and Telemetry representation
  - Model-based Planning representation
- Declarative Abstractions and Refinements
- Detecting Model Errors
  - Data Driven / Learning
  - Fault Management / 'Oracle'
- The Challenge: Correcting Model Errors

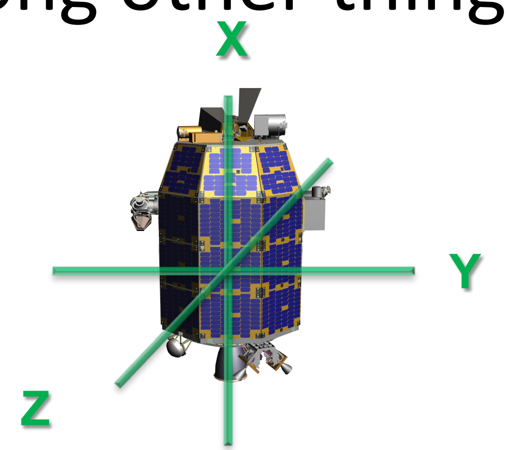
Example

- Suppose we are developing a mission planning system for a spacecraft.
  - This could be for a ground system or as part of an autonomous spacecraft.
- How does a spacecraft plan a change the direction (attitude) it is pointing?
  - A change of pointing is called a slew.



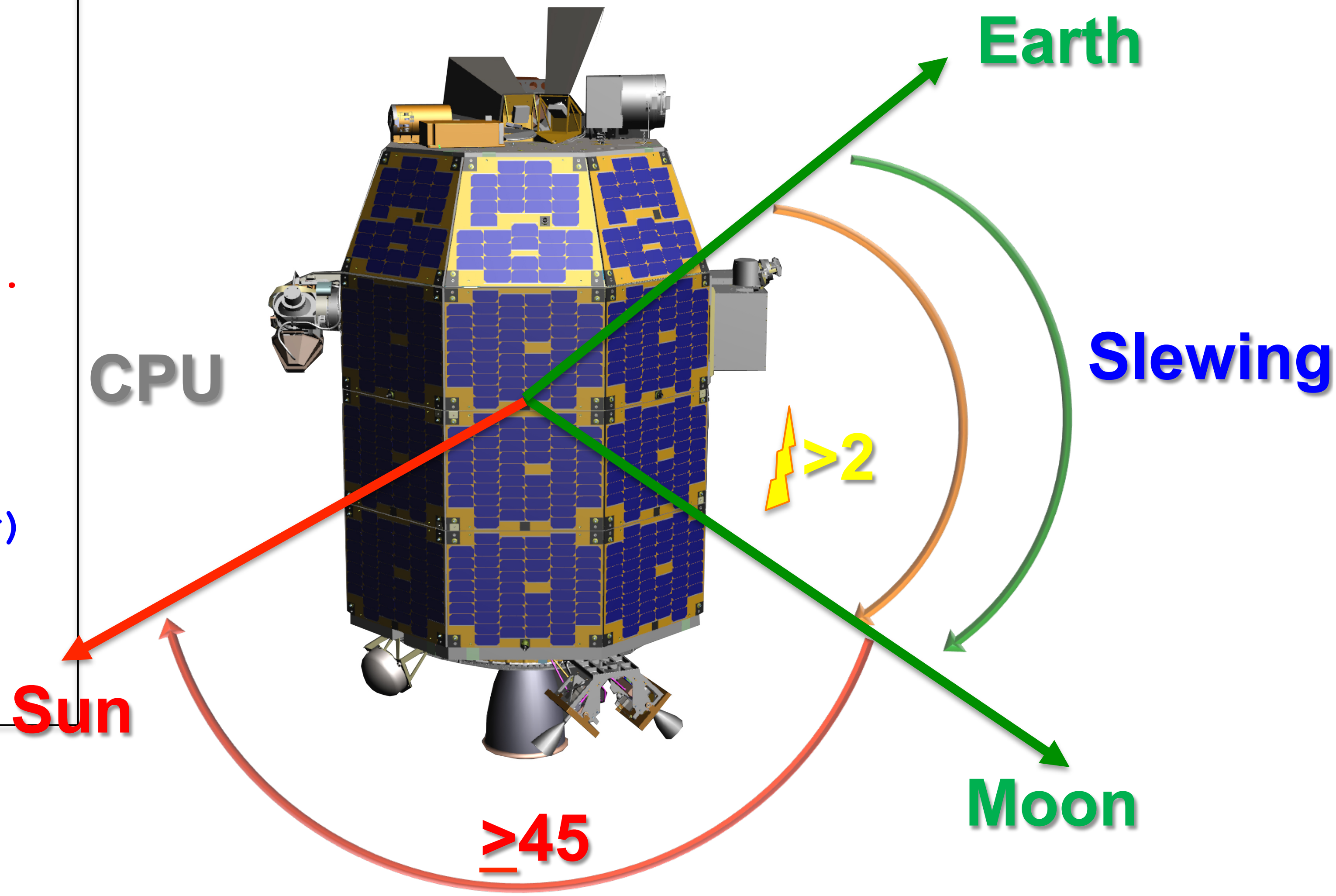
Example

- Attitude is angles <*x*, *y*, *z*> in some absolute coordinate frame (e.g. Earth-centric).
- Slews are constrained by solar panel power generation, thermal, communications to Earth, sensor and instrument performance and safety (among other things).



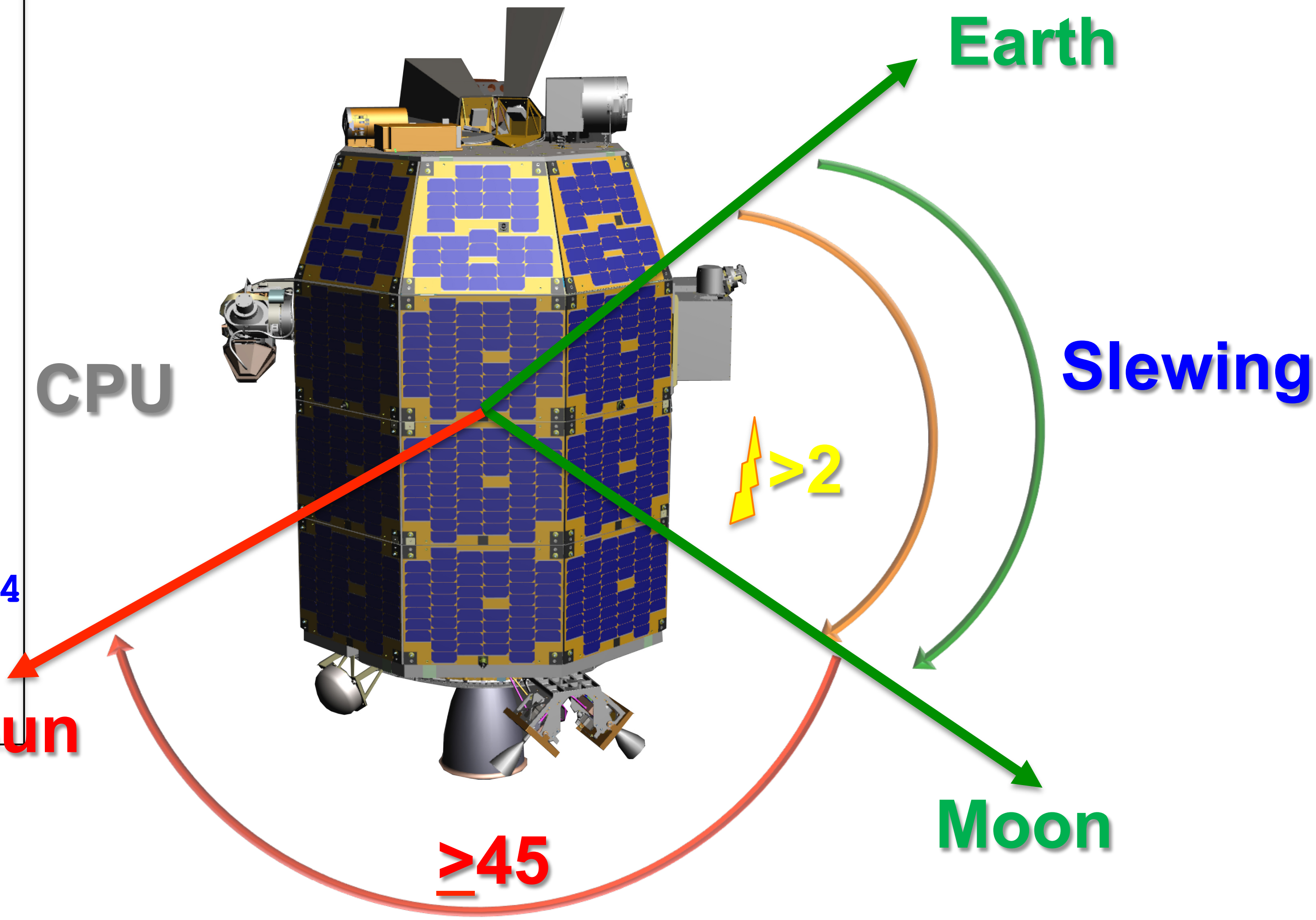
Example (Commands)

```
//Startracker
s({o,f})


//CPU
c({o,f})
u(p)
//Power dist.
pd({s,b})

//Reaction
//Wheels
r(wi,{c,o},v)

//Battery
pd({s,b})
```



Example (Data)

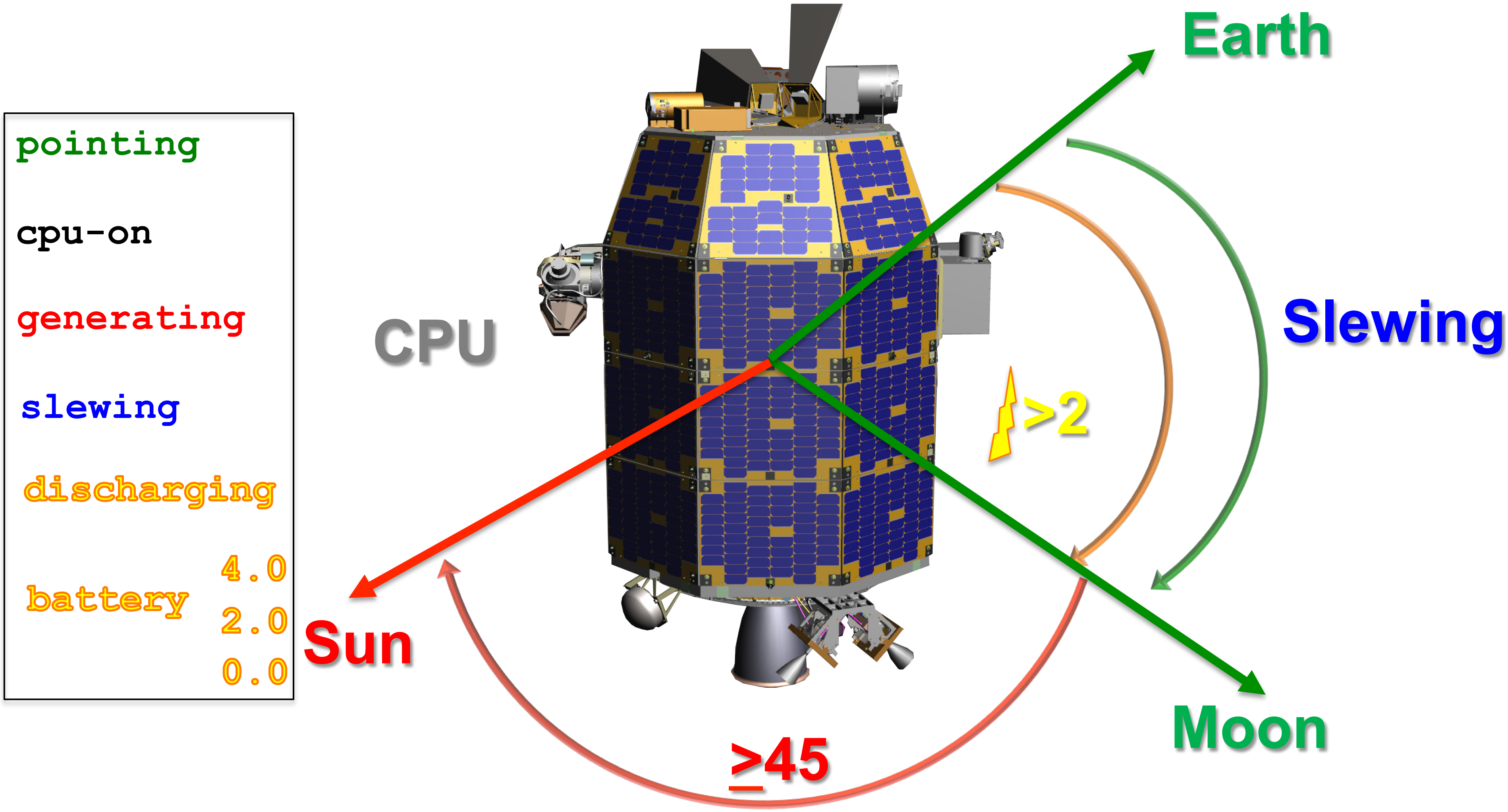
```
//Startracker
x,y,z
dx,dy,dz
s
//CPU
on,off
u
//Power dist.
s1,s2
p
//Reaction
//Wheels
w1,w2,w3,w4
dw1,dw2,dw3,dw4
//Battery
b,db
```



The Planning Model

- A planning *model* consists of:
  - Objects – things in the world.
  - Predicates – properties of things. (True/False)
  - Functions – properties of things. (Numbers)
  - 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.

Example


pointing
cpu-on
generating
slewing
discharging
battery
4.0
2.0
0.0
CPU
Earth
Slewing
>2
Sun
Moon
≥45

Model-Based Planning

```
(:durative-action slew
  :parameters (?from - attitude
               ?to - attitude)
  :duration (= slew-time ?from ?to)
  :condition (and
      (at start (pointing ?from))
      (at start (cpu on))
      (over all (cpu on))
      (at start (generating))
      (at start (>= (battery ?sc) 3.0))
  :effect
   (and
      (at start (decrease(battery ?sc) 2.0
      (at start (discharging ?sc))
      (over all (discharging ?sc))
      (at start (slewing))
      (over all (slewing))
      (at end (not slewing))
      (at start (not generating))
      (at start (not pointing ?from))
      (at end (pointing ?to))
```

Action

Object

Condition

Predicates

Temporal Qualifier

Effect

Functions

Predicates

pointing

cpu-on

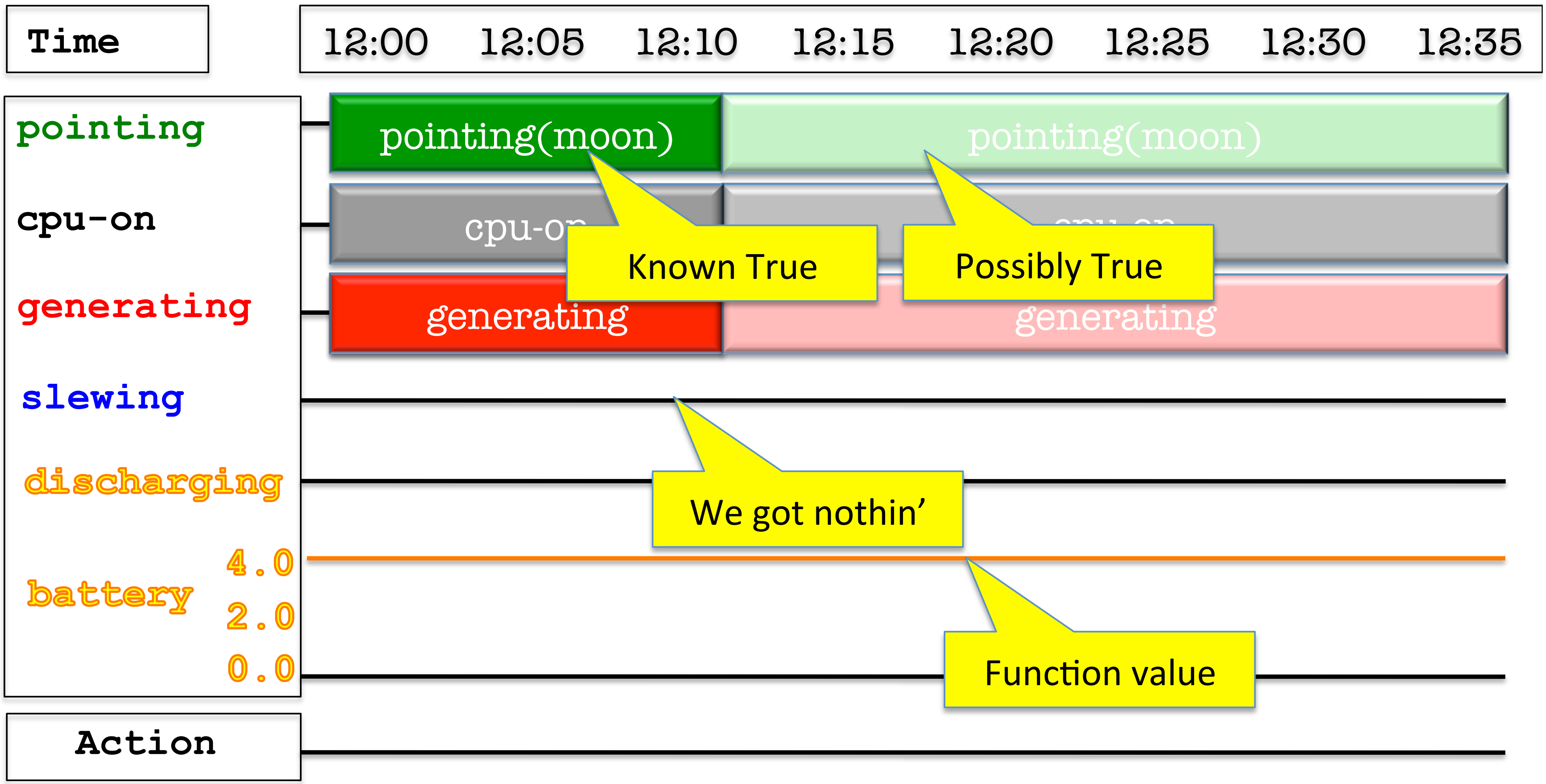
generating

slewing

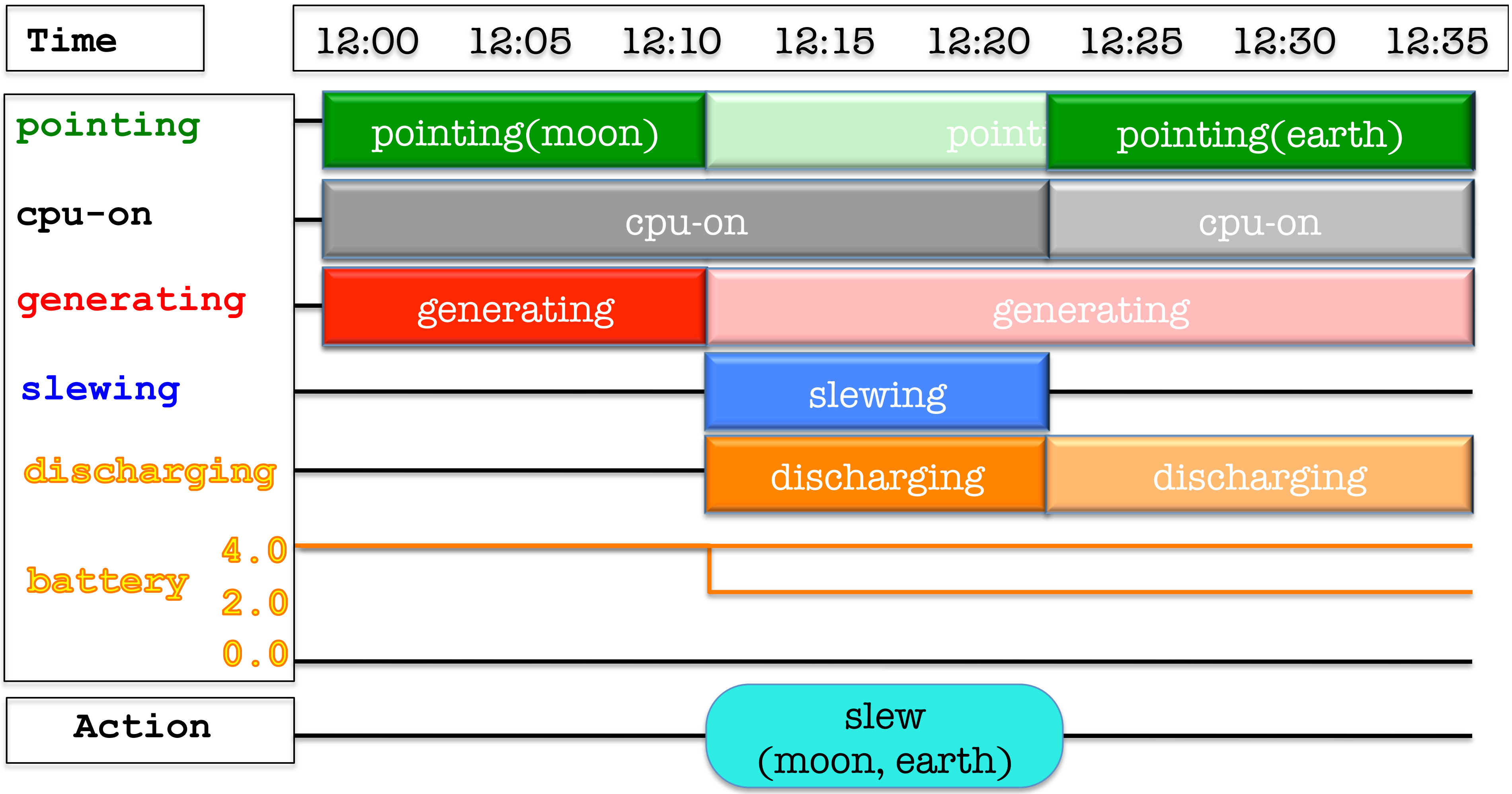
discharging

battery 4.0 2.0 0.0

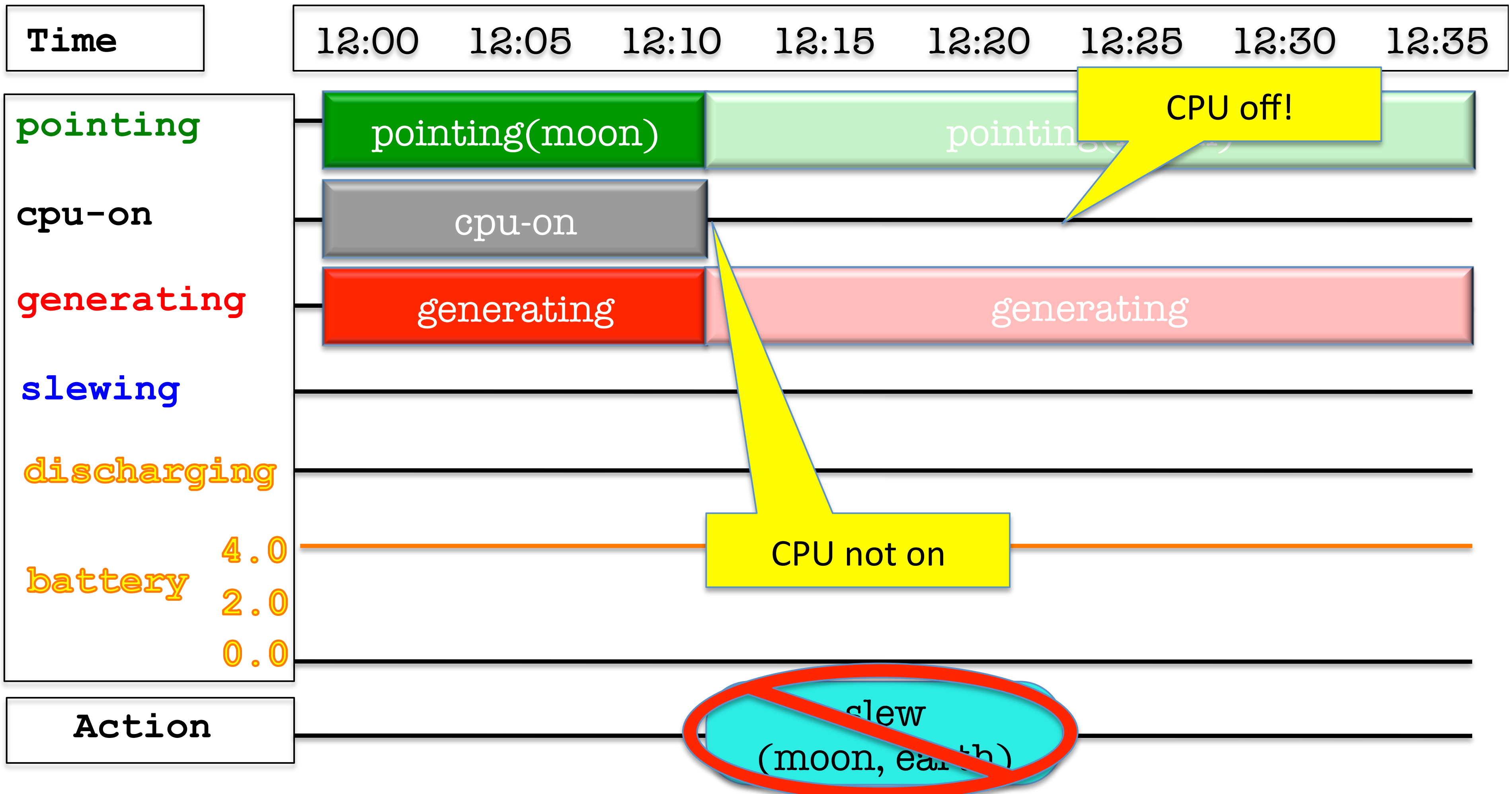
Model-Based Planning



Model-Based Planning

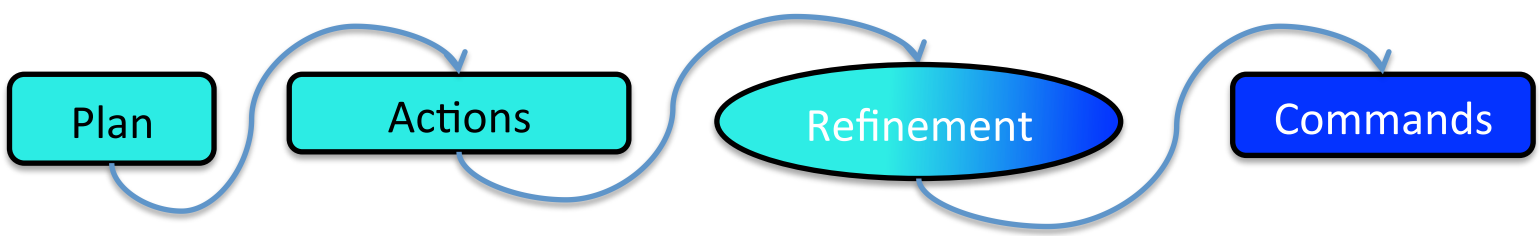


Model-Based Planning



Declarative Abstractions and Refinements

- Unlike other applications, *abstraction* is a key element of modeling in this type of problem.
  - Planners approximate or abstract the actual spacecraft behavior.
  - These abstractions are typically not *documented*.



Declarative Abstractions and Refinements

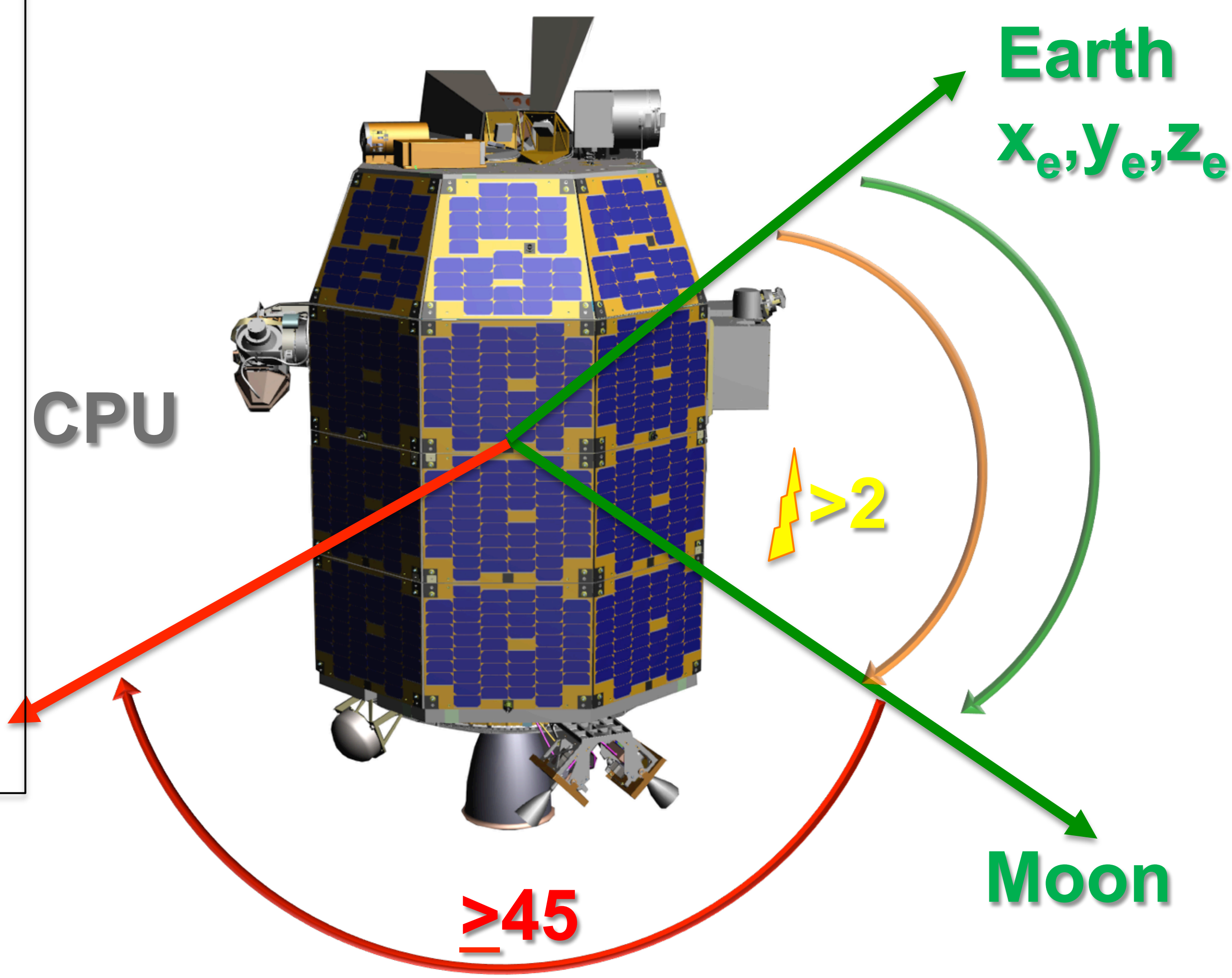
- Predicate and Function Abstractions:
  - Functions that map the spacecraft data to planning model predicates or functions.
  - One abstraction per predicate or function.
- Command Refinements:
  - Function mapping a planning action to a command sequence.
  - One refinement per plan action.

Predicate Abstractions

Pointing =
$(x_e-\varepsilon<x<x_e+\varepsilon) \wedge$
$(y_e-\varepsilon<y<y_e+\varepsilon) \wedge$
$(z_e-\varepsilon<z<z_e+\varepsilon)$

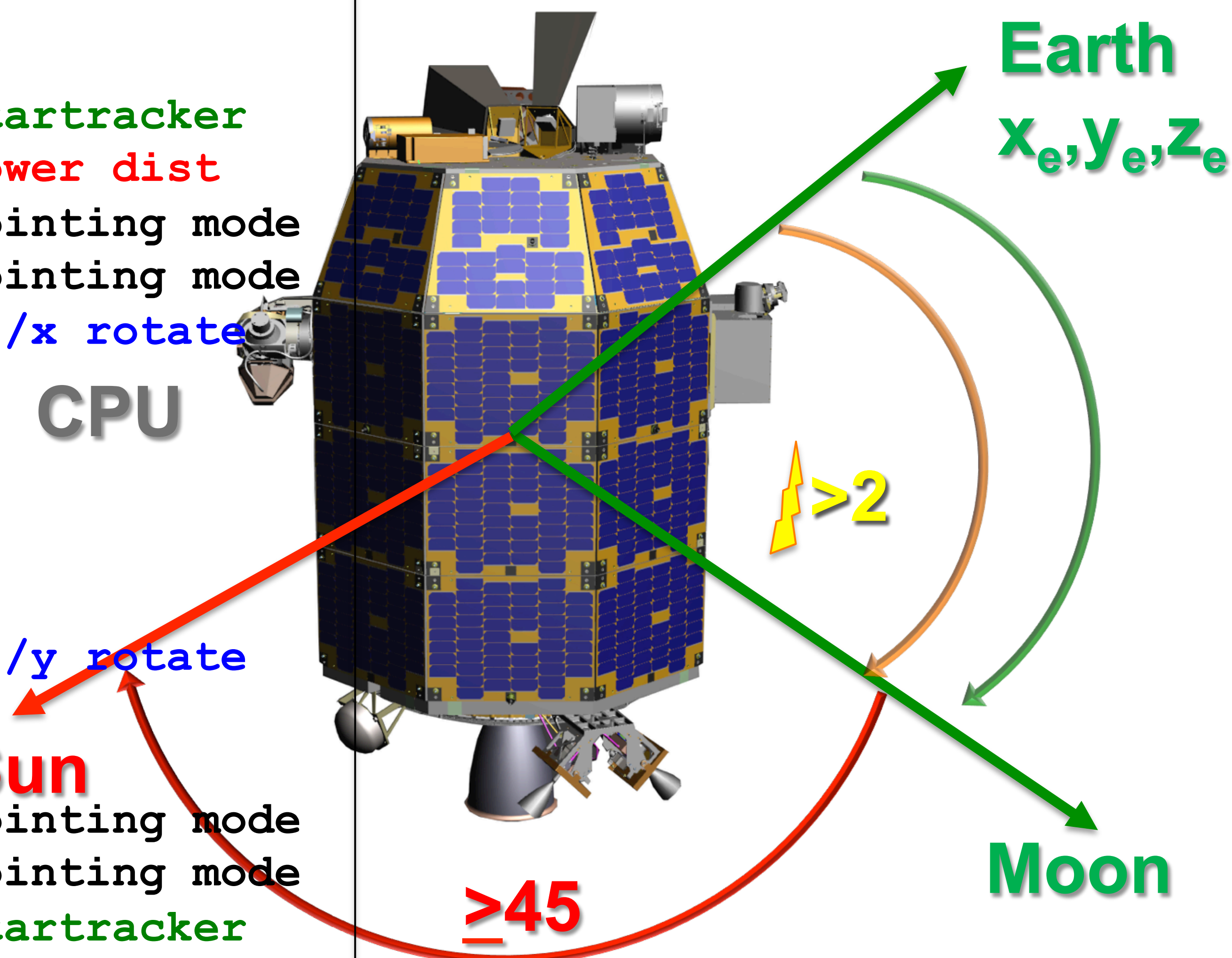
Generating =
$(p=s) \wedge$
$(s1+s2>\varepsilon_1) \wedge$
$(db>\varepsilon_2)$

$2=db_1+db_2+db_3$



Command Refinement

```
Slew:
[0]:
s(f);       //startracker
pd(b);      //power dist
u(pi,o);    //pointing mode
u(ps,f)     //pointing mode
r(w1,c,500); //x rotate
r(w2,c,500);
[10]
r(w1,c,0);
r(w2,c,0);
[11]
r(w1,c,500); //y rotate
…
[14]
u(pi,f);    //pointing mode
u(ps,o)     //pointing mode
s(o);       //startracker
```



Detecting Modeling Errors and Model Drift

- Modeling is error prone; abstraction compounds the errors that can be introduced.
- Models may also become wrong over time
  - Due to changes in the system
  - Due to changes in the operating environment
  - Due to changing mission goals and objectives

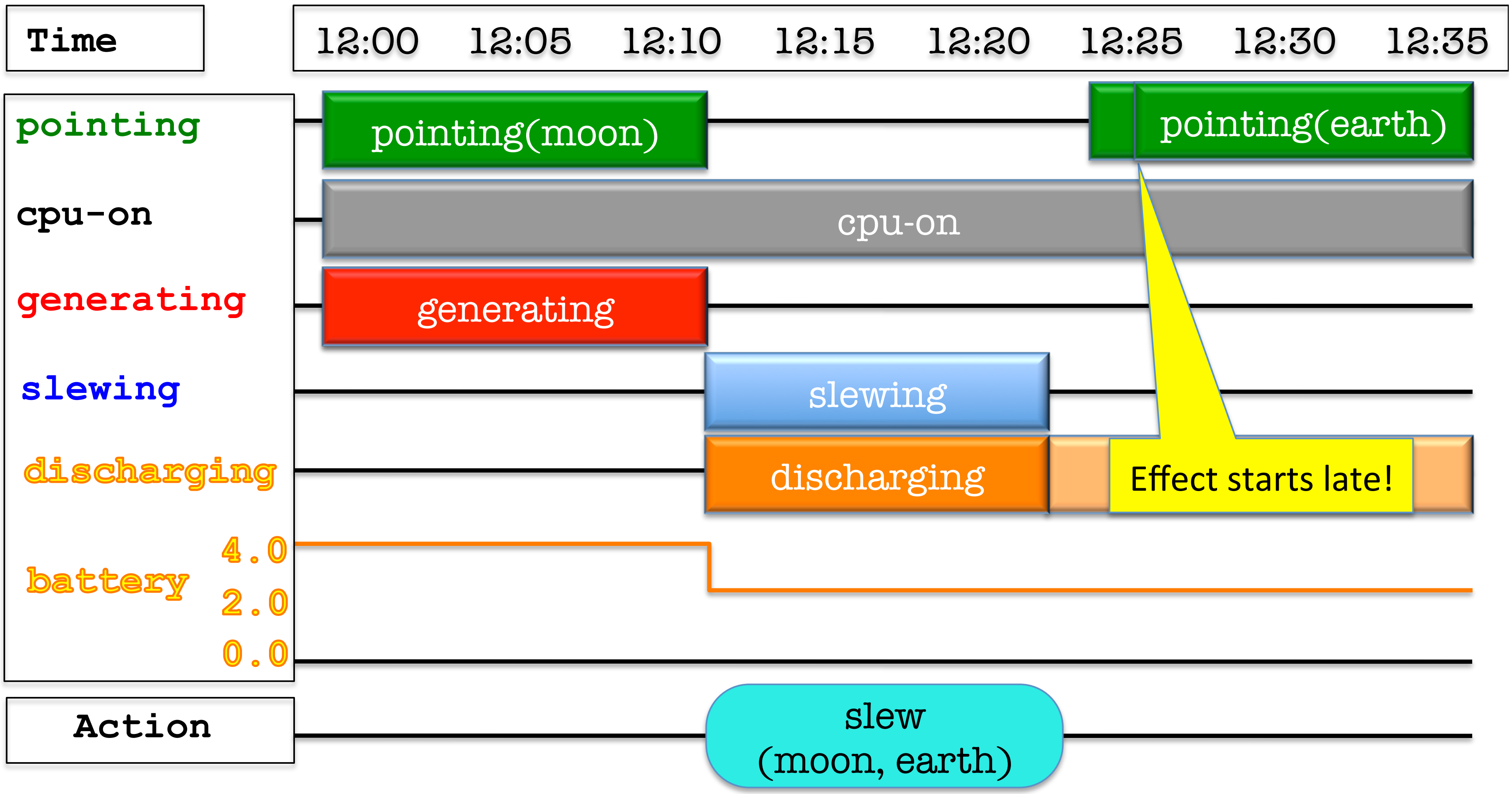
Detecting Modeling Errors and Model Drift

- What manner of modeling errors can occur?
  - Action Failure: Conditions satisfied but action fails
  - Missing Condition: Action condition not in plan so action fails
  - Unexpected Condition: Condition unexpectedly influences action outcome
  - Missing effect: Action succeeds but effect missing
  - Unrealized Effect: Action has unmodeled effect
  - Timing discrepancy: Action length differs or effects occur at different times than expected

Detecting Modeling Errors and Model Drift

- What are the root causes of modeling errors?
  - Missing Command
  - Bad command order
  - Incorrect command input
  - Mis-timed command
  - Missing Abstraction
  - Abstraction error

Detecting Modeling Errors

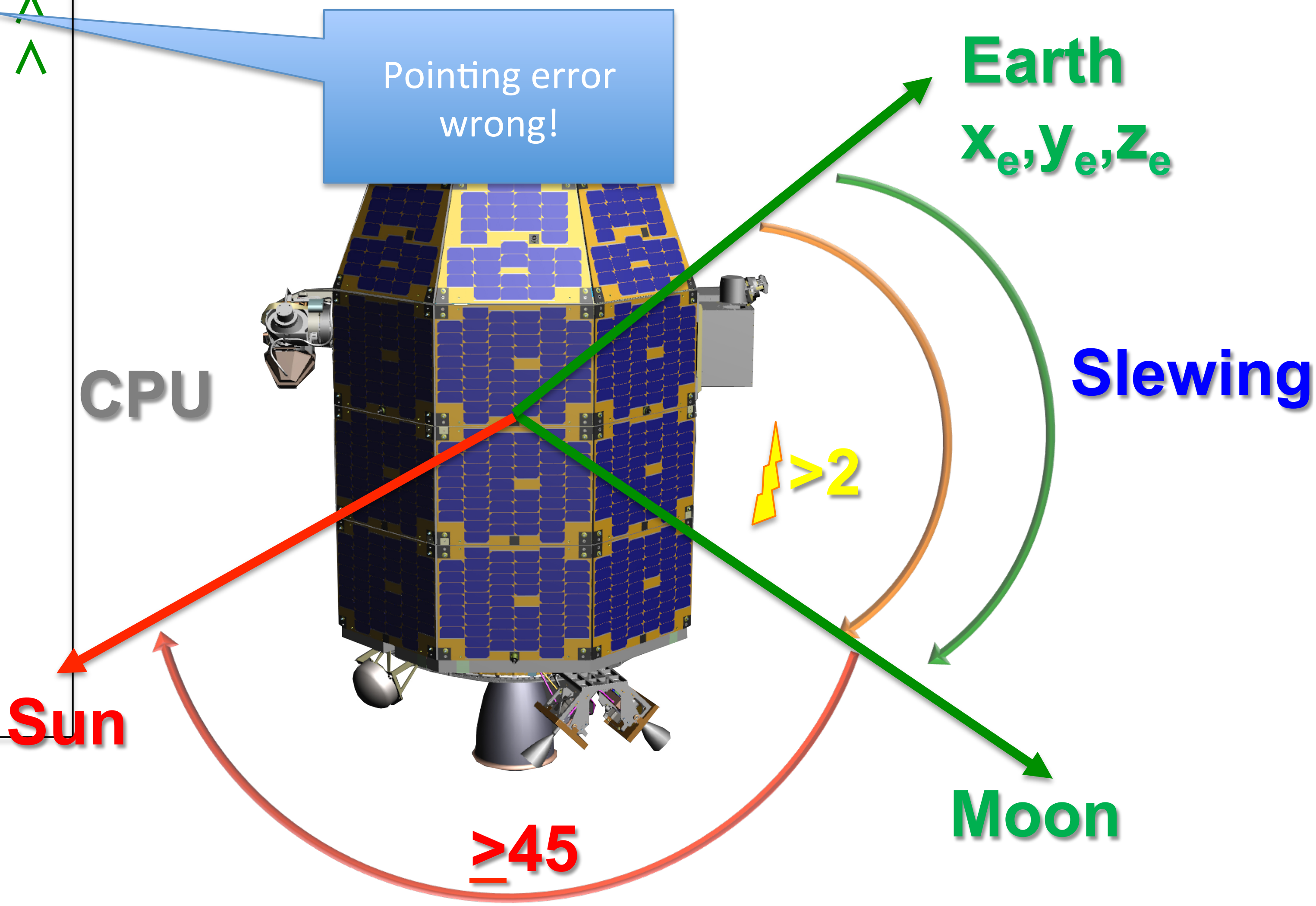


Detecting Modeling Errors

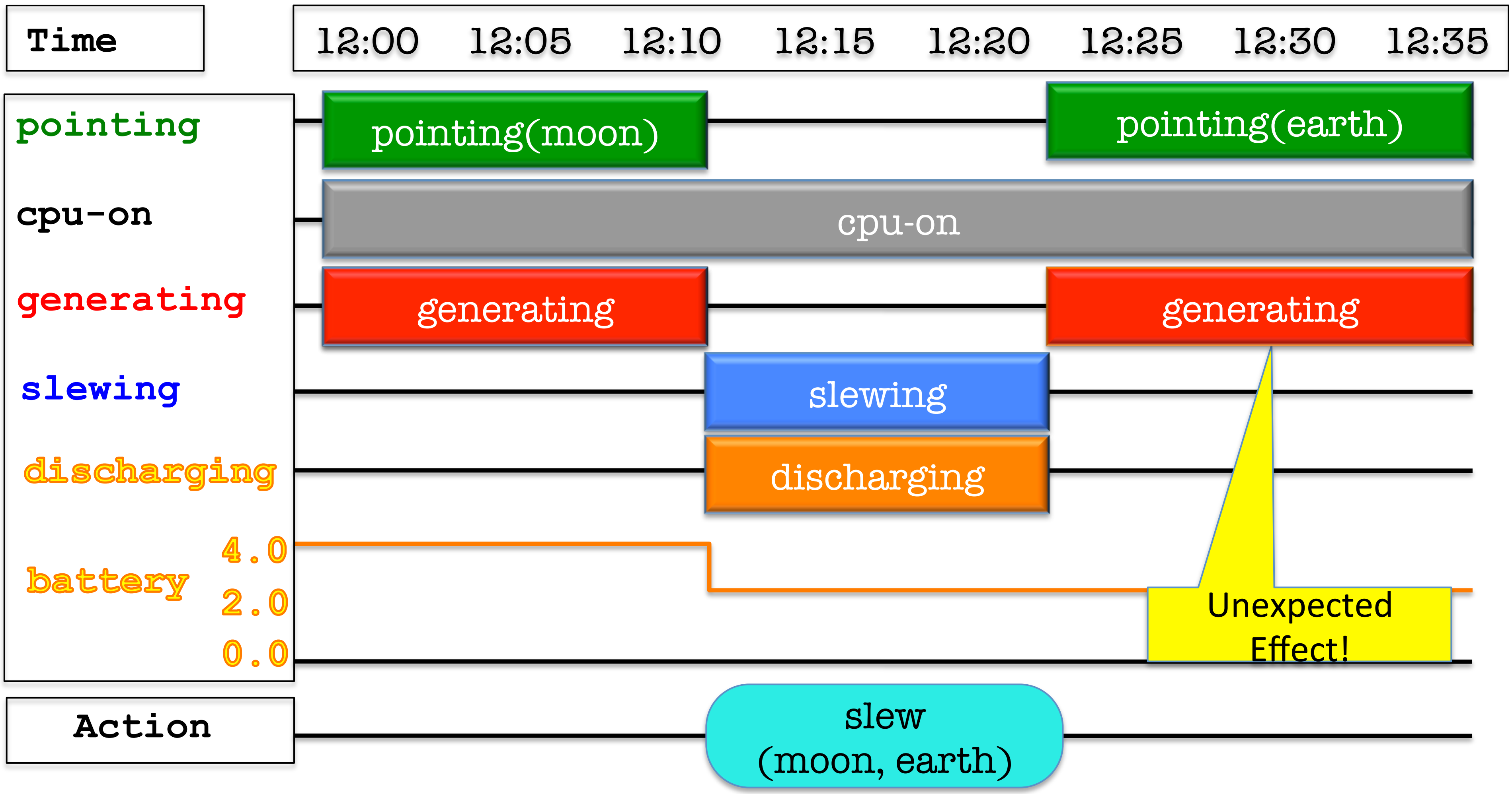
Pointing =
$(x_e - \varepsilon < x < x_e + \varepsilon) \wedge$
$(y_e - \varepsilon < y < y_e + \varepsilon) \wedge$
$(z_e - \varepsilon < x < z_e + \varepsilon)$

Generating =
$(p=s) \wedge$
$(s1+s2>\varepsilon_1) \wedge$
$(db>\varepsilon_2)$

$2 = db_1 + db_2 + db_3$



Detecting Modeling Errors



Detecting Modeling Errors

pointing
cpu-on
generating
slewing
discharging
battery 4.0 2.0 0.0

```
(:durative-action slew
  :parameters (?from - attitude
               ?to - attitude)
  :duration (= slew-time ?from ?to)
  :condition (and
      (at start (pointing ?from))
      (at start (cpu on))
      (over all (cpu on))
      (at start (generating))
      (at start (>= (battery ?sc) 3.0)))
  :effect
   (and
      (at start (decrease(battery ?sc) 2.0)
      (at start (discharging ?sc))
      (over all (discharging ?sc))
      (at start (slewing))
      (over all (slewing))
      (at end (not slewing))
      (at start (not generating))
      (at start (not pointing ?from))

      (at end (pointing ?to))
```

Detecting Modeling Errors

```
(:durative-action slew
  :parameters (?from - attitude
               ?to - attitude)
  :duration (= slew-time ?from ?to)
  :condition (and
      (at start (pointing ?from))
      (at start (cpu on))
      (over all (cpu on))
      (at start (generating))
      (at start (>= (battery ?sc) 3.0)))
  :effect
   (and
      (at start (decrease(battery ?sc) 2.0)
      (at start (discharging ?sc))
      (over all (discharging ?sc))
      (at start (slewing))
      (over all (slewing))
      (at end (not slewing))
      (at start (not generating))
      (at start (not pointing ?from))
      (at end (generating))
      (at end (pointing ?to))
```

pointing

cpu-on

generating

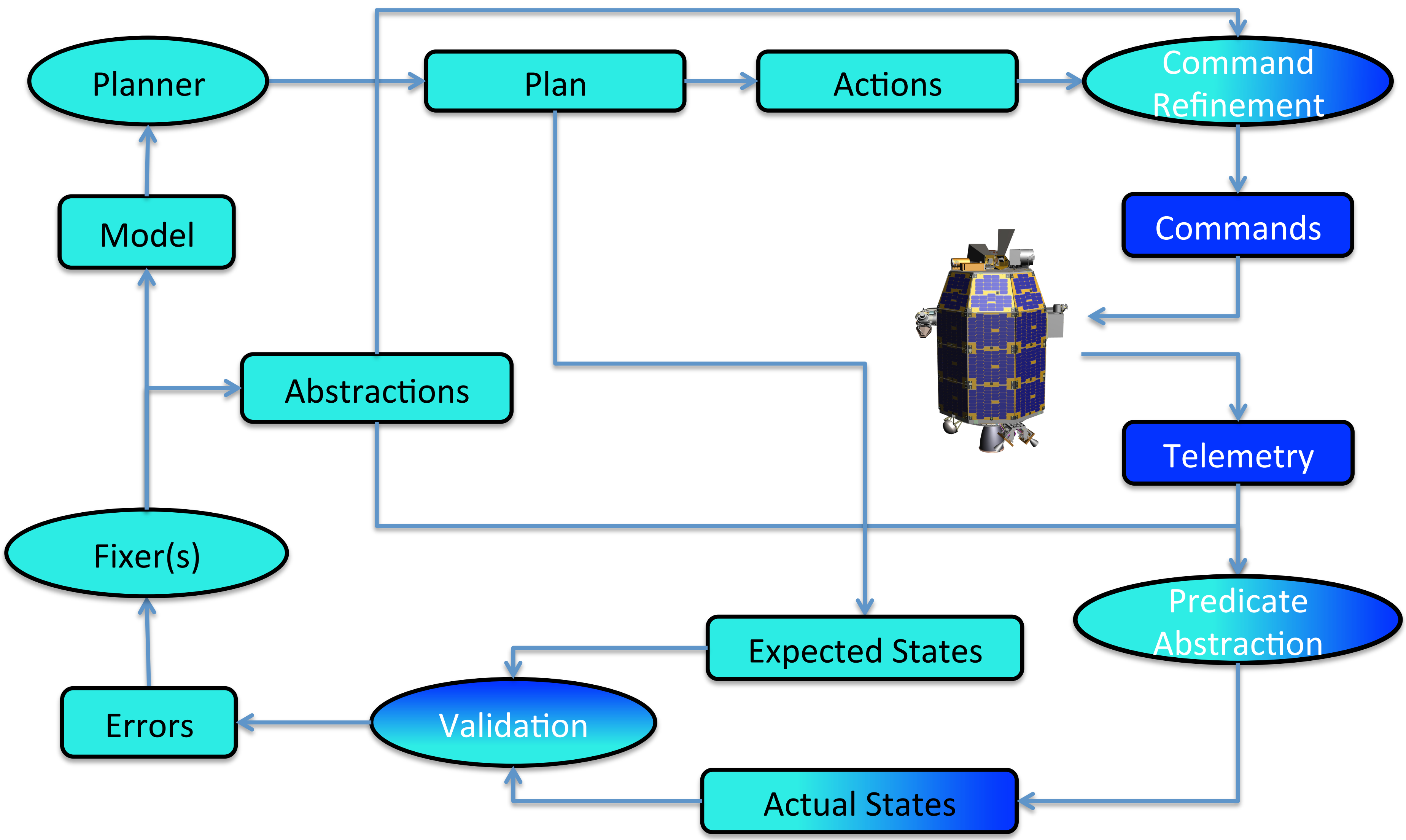
slewing

discharging

battery 4.0 2.0 0.0

Fixed Model

“Reflection” Architecture



“Reflection” Architecture: Algorithm Sketch

- 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: $|X_1||X_2|\ldots|X_n|<|Y_1||Y_2|\ldots|Y_m|$
    - Special cases: eliminate variable

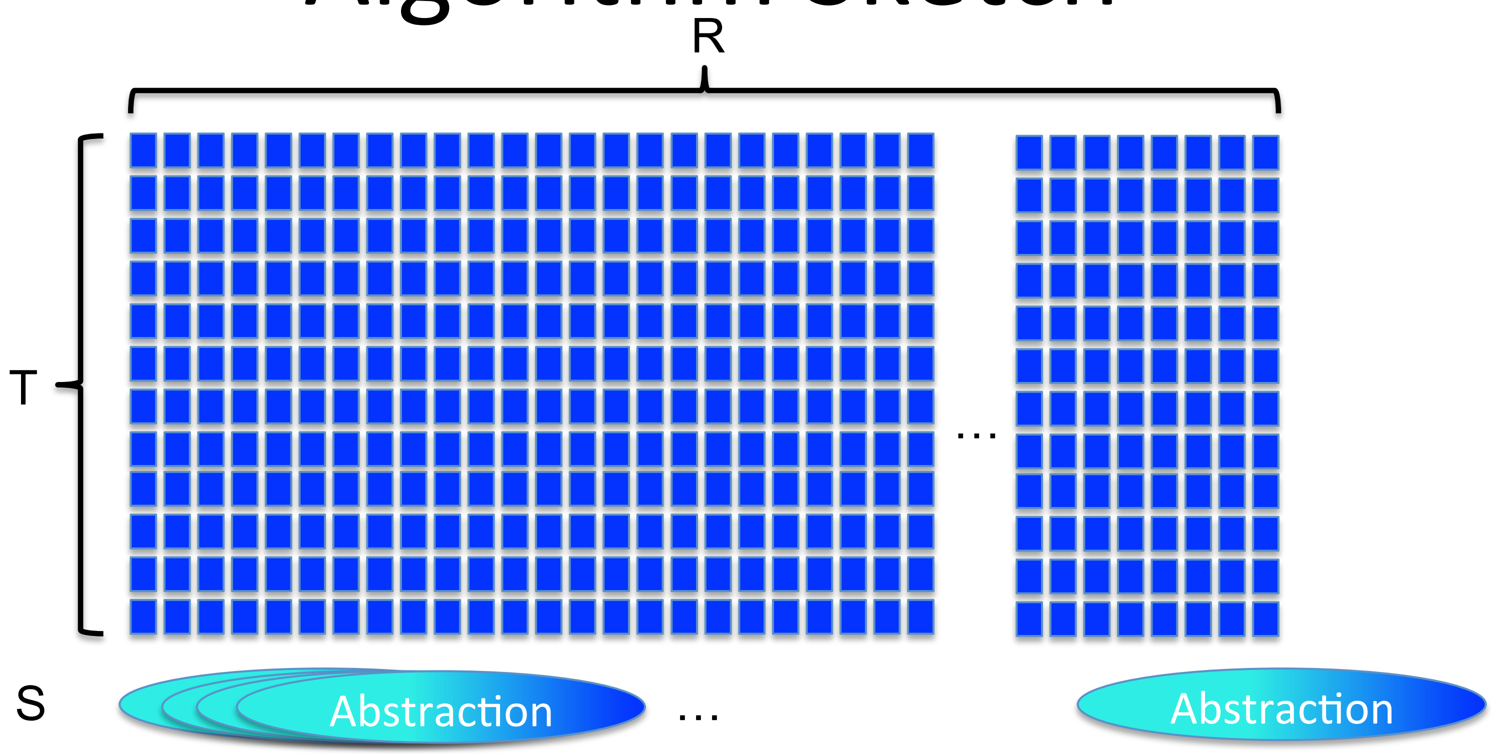
"Reflection" Architecture: Algorithm Sketch

- Planner / plans
  - N actions
  - P_n action conditions / effects for action n
  - S predicates in the model (one abstraction per predicate)
  - M_p predicate instances in a plan
- System
  - C_n commands in refinement of action n
  - T telemetry items
  - R values for each item per run
  - $M_s \ll TR$ predicate instances produced by simulation
- A similar analysis can be done for numerical abstractions
  - A finite number of numerical abstractions are formalized!

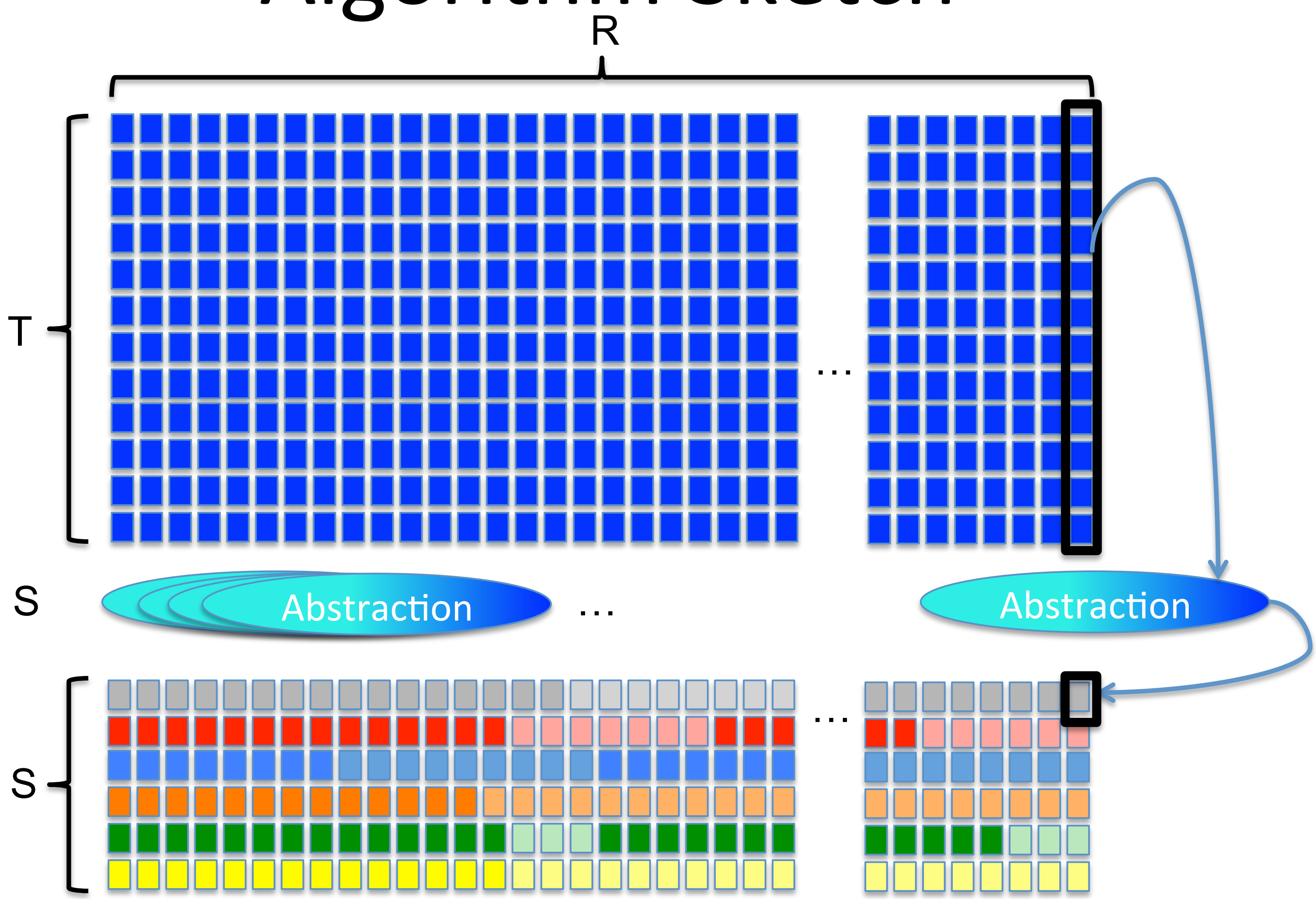
"Reflection" Architecture: Algorithm Sketch

- Generate refinement from plan
  - $\Sigma\, C_n$ (N actions, C_n commands in refinement of action n)
- Execute / simulate refined command sequences
- Generate plan abstraction from telemetry
  - T telemetry items, R values for each item per run, S predicate types.
  - First pass is to generate states: for all R, for all abstractions S, each abstraction uses at most T telemetry items. This gets us runtime TRS.
  - 2nd pass is to determine start / end times of predicates; this is another SR.
  - (There are important assumptions about the form of the abstractions i.e. they only use values at one time tic)
  - Total: TSR + SR

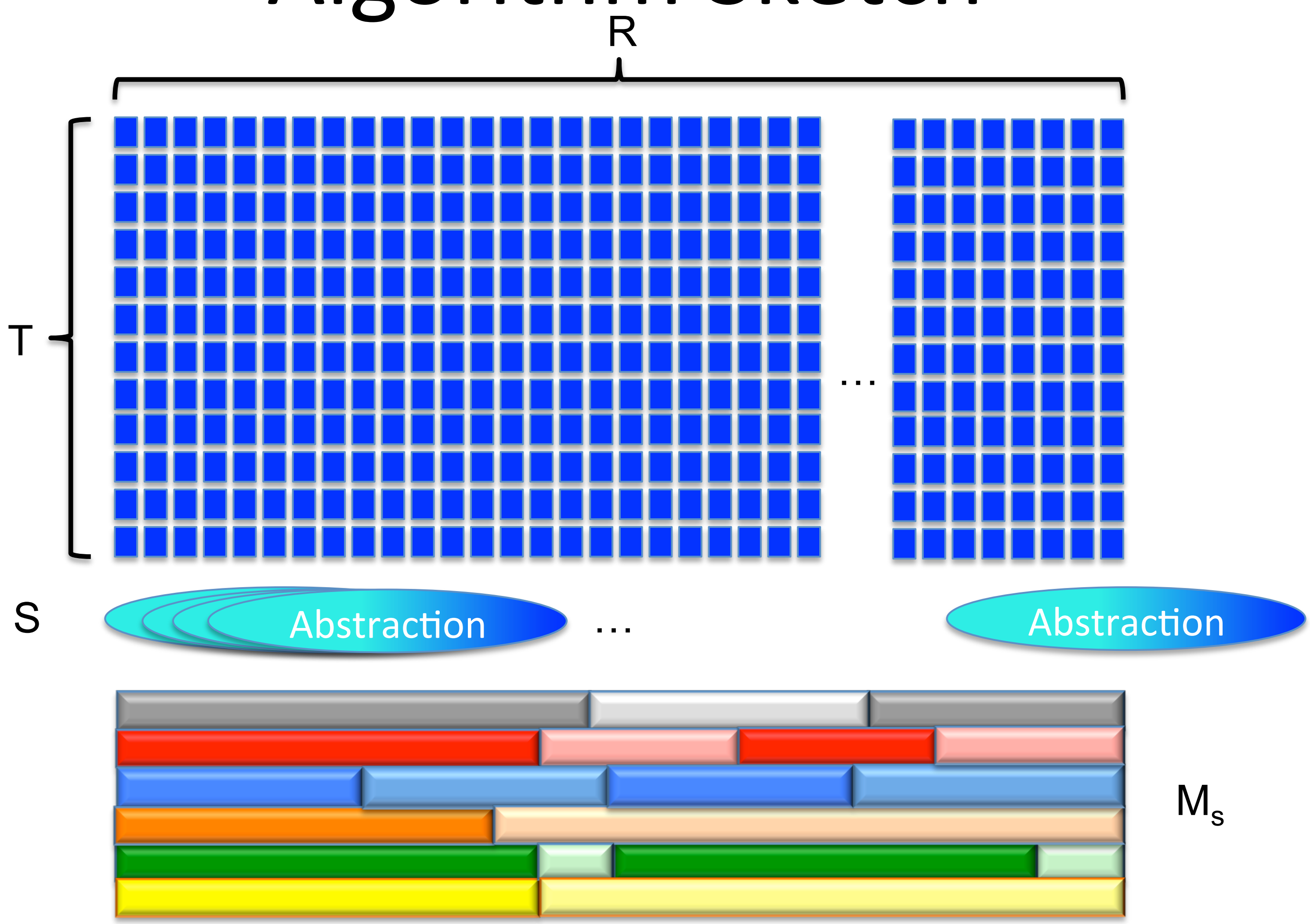
"Reflection" Architecture: Algorithm Sketch



"Reflection" Architecture: Algorithm Sketch



“Reflection” Architecture: Algorithm Sketch



“Reflection” Architecture: Algorithm Sketch

- Generated warnings
  - Check to see if simulated predicate start / end time match compared to plan predicate start / end times for same predicate types; sufficient, but overkill, to check every pair $m \in M_p$ $o \in M_s$.
  - $O(M_p M_s)$
- Generate action discrepancies
  - N actions, P_n action conditions / effects
  - For each condition/effect of an action, may need to search all M_s predicate to match conditions / effects
  - $O(M_s \Sigma_{n \in N} (P_n))$

Promise of Reflection on Models

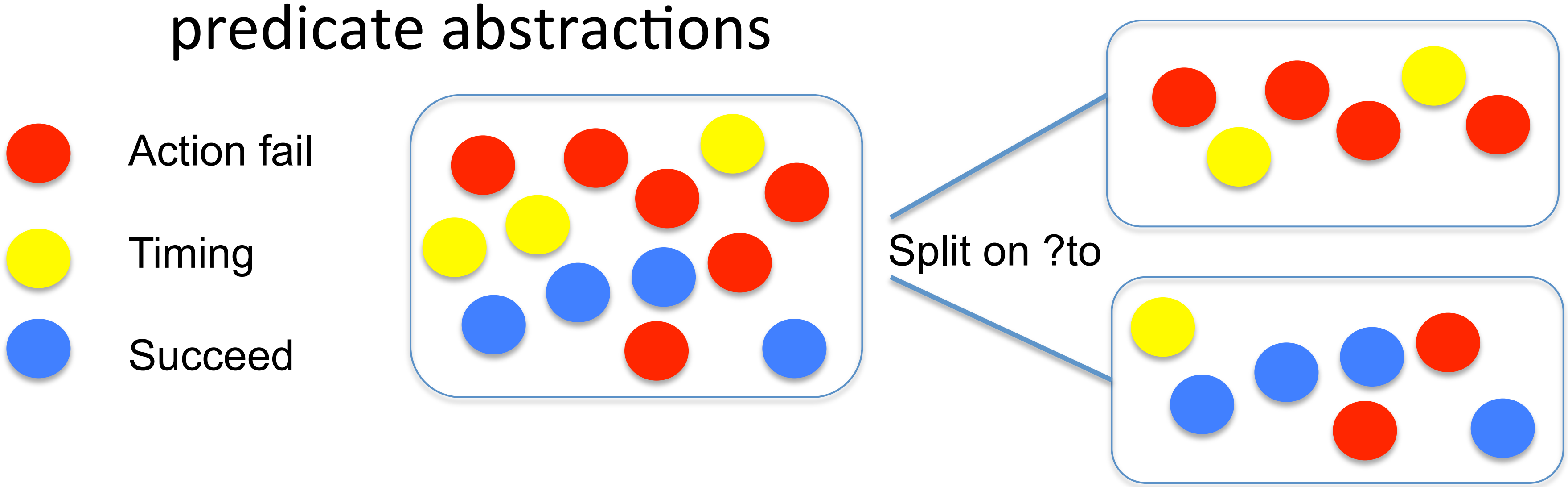
- Detect modeling errors prior to launch through testing
- Adapt to changes in spacecraft environment
  - Solar panel power generation due to dust (e.g. during surface operations)
  - Unpredictable gravitational field impact on attitude and orbit determination (e.g. small bodies like asteroids or comets)
  - Unpredictable communications performance
  - Unpredictable lighting conditions
- Adapt to changes in spacecraft performance
  - Solar panel power generation due to age
  - CMG degradation
  - Battery performance (e.g. cell or string failure)

Promise of Reflection on Models

- Advantages of reflection and adaptation onboard:
  - More data than telemetered back to ground
  - Higher rate data
  - Ability to reflect continuously
  - No need to pay costs of communication

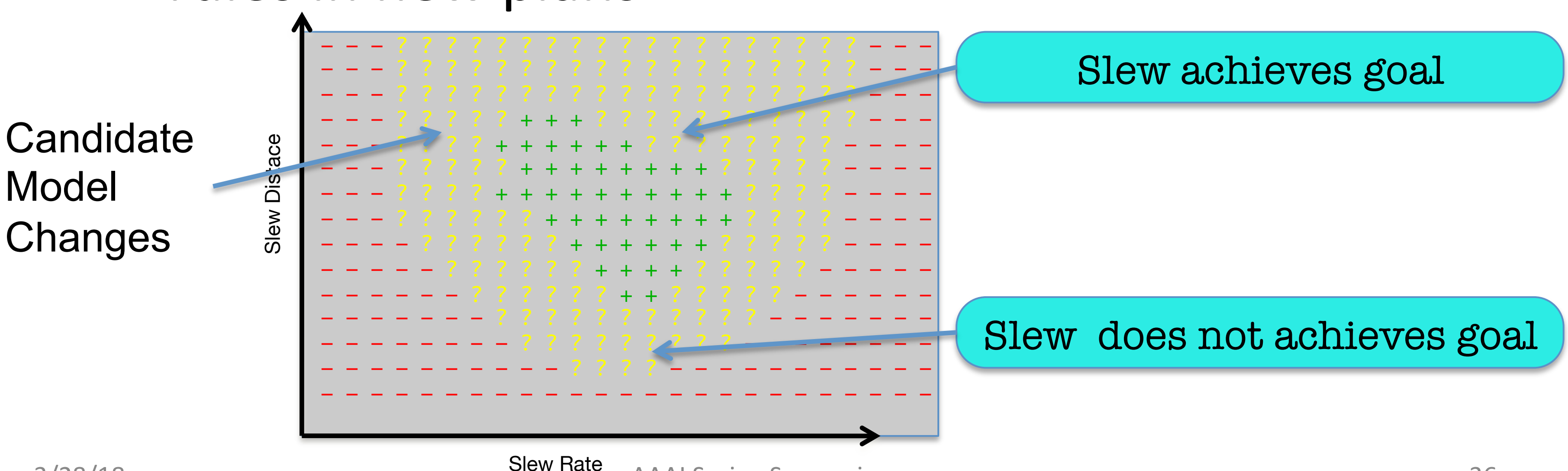
Promise of Reflection on Models

- Promising techniques
  - Classification – identify rules distinguishing cases when actions fail vs when they succeed
  - Function approximation – attempt to debug predicate abstractions



Promise of Reflection on Models

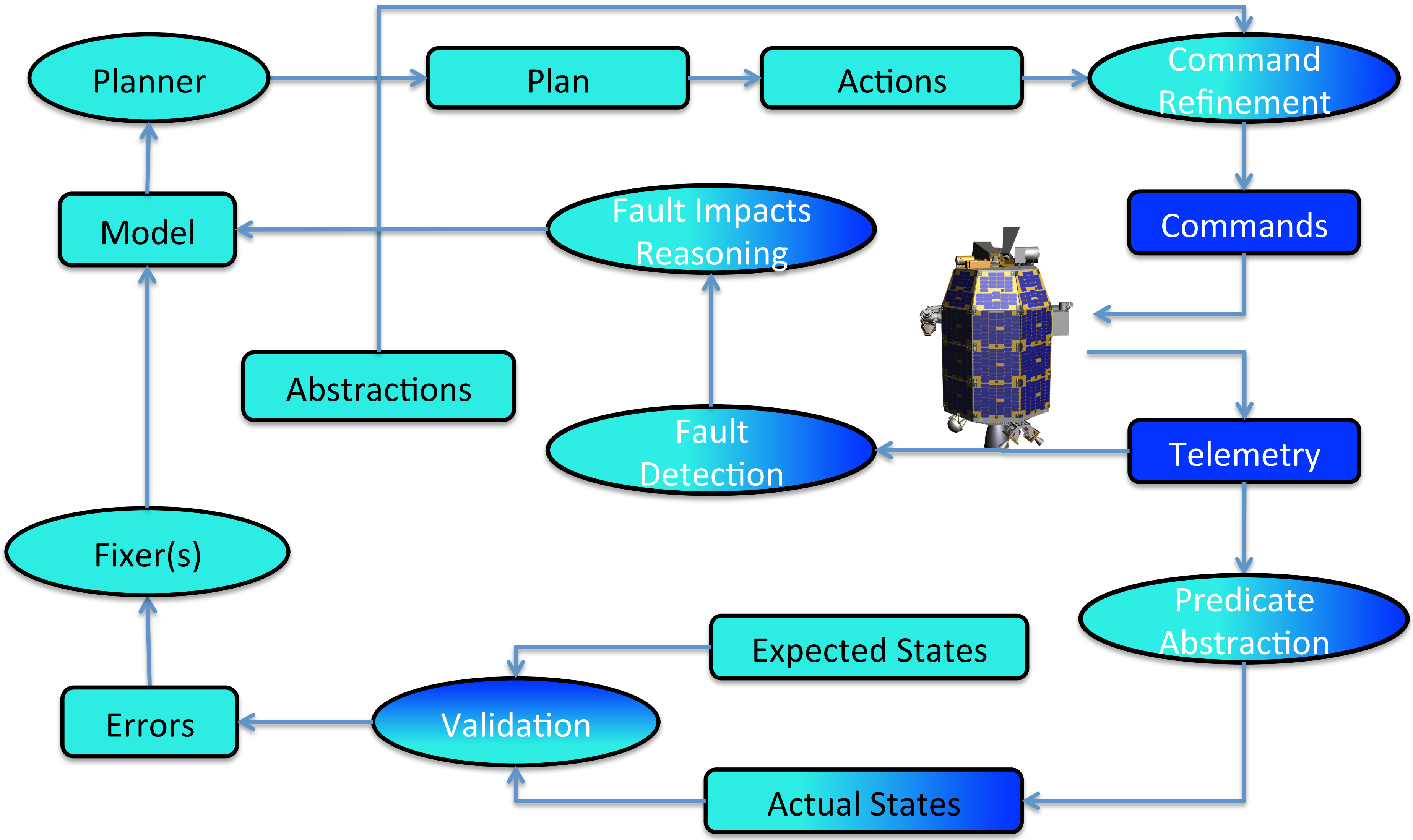
- Promising techniques
  - Clustering – identify patterns of behavior in data and map them to predicates
  - Exploratory actions – judicious use of proposed rules in new plans



Promise of Reflection on Models

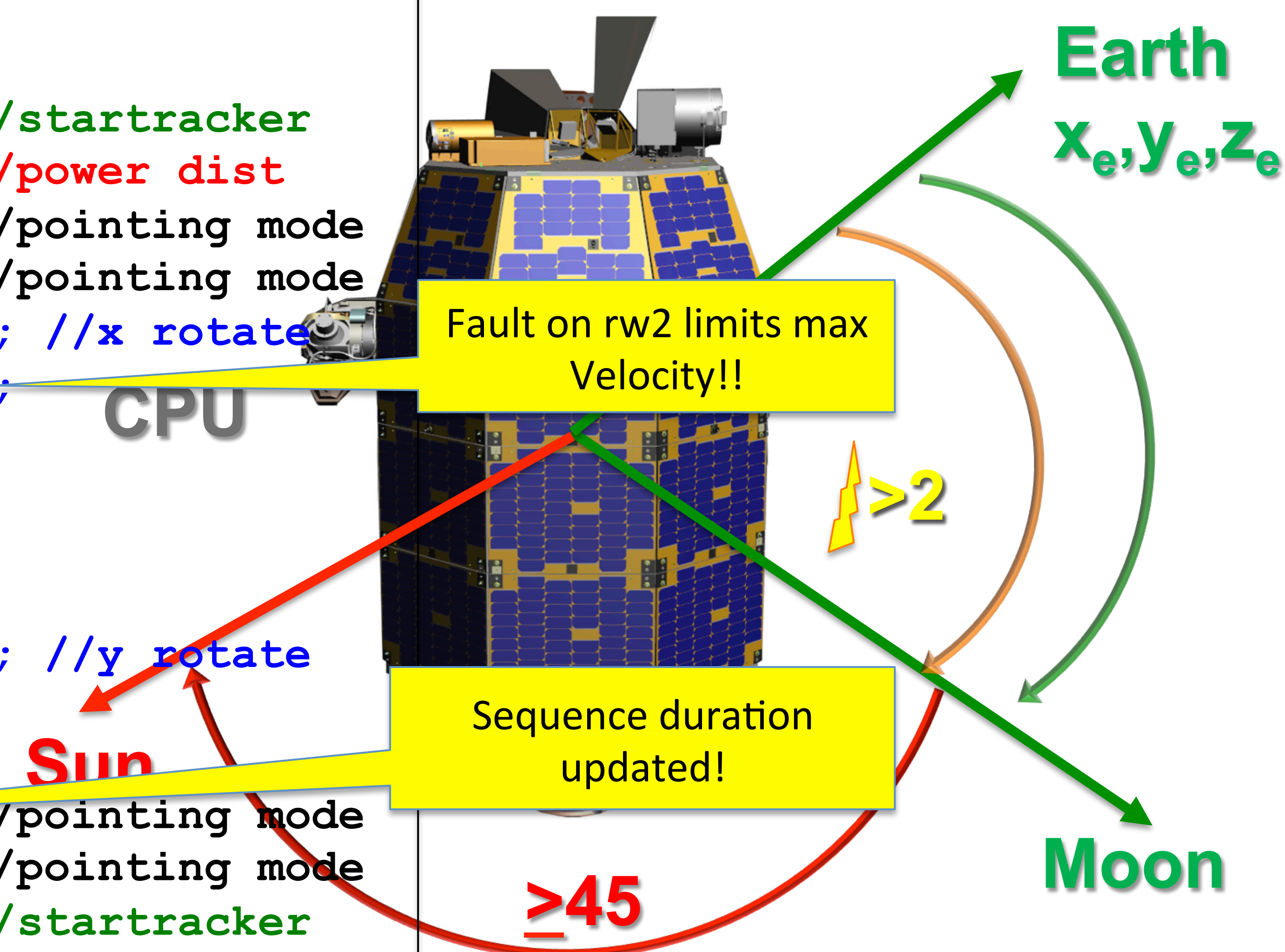
- What about faults?
  – A special class of degradation / unexpected event
  – Performance changes are detected and reported by fault management algorithms
- Instead of using data to learn and characterize changes in planning model, make use of these fault detection algorithms' outputs directly.
  – Treat fault management algorithm as 'oracle'

“Reflection” Architecture



Promise of Reflection on Models

```
Slew:
[0]:
s(f);       //startracker
pd(b);      //power dist
u(pi,o);    //pointing mode
u(ps,f)     //pointing mode
r(w1,c,500); //x rotate
r(w2,c,200);
[12]
r(w1,c,0);
r(w2,c,0);
[14]
r(w1,c,500); //y rotate
…
[17]
u(pi,f);    //pointing mode
u(ps,o)     //pointing mode
s(o);       //startracker
```



Promise of Reflection on Models

```
(:durative-action slew
  :parameters (?from - attitude
               ?to - attitude)
  :duration (= slew-time ?from ?to)
  :condition (and
      (at start (pointing ?from))
      (at start (cpu on))
      (over all (cpu on))
      (at start (generating))
      (at start (>= (battery ?sc) 3.0)))
  :effect
   (and
      (at start (decrease(battery ?sc) 2.0)
      (at start (discharging ?sc))
      (over all (discharging ?sc))
      (at start (slewing))
      (over all (slewing))
      (at end (not slewing))
      (at start (not generating))
      (at start (not pointing ?from))
      (at end (pointing ?to))
```

Fault increases duration!

```
pointing

cpu-on

generating

slewing

discharging

              4.0
battery       2.0
              0.0
```

Challenges of Reflection

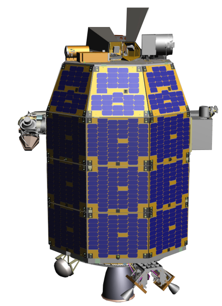
- Algorithms are resource (computationally, memory) intensive
  - "Software has weight"
- Existing algorithms for proposing model fixes may not be sufficient
- Algorithms may not find answers due to insufficient data
- Experimentation in mission critical environments may be dangerous
- Model "configuration management" may be needed if a proposed fix does not perform well

Challenges of Reflection

- Fault management algorithms in general reason at low level of abstraction
  - Output often can't directly be used to change planning models
  - Requires a similar abstraction between components that can fail and actions in planning model
- Fault management algorithms exist for detecting loss of capability and redundancy, and leaks
- Making connection to action model is hard
  - Example: leak detection. Using leaky system increases resource consumption rate.
  - Operationally, may prefer simply not to use this subsystem (malfunction => don't use) but sometimes ya gotta do it

A Few Words About a REAL Spacecraft: LADEE

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



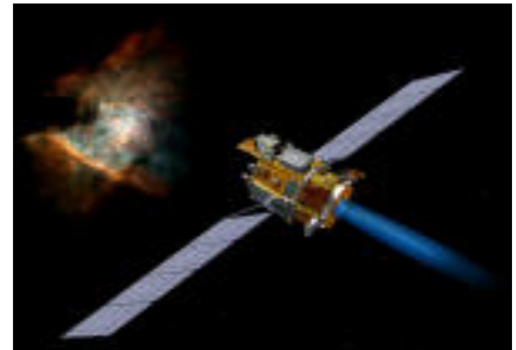
- The LADEE planner model has ~ 12000 lines.
- Simulation data produced at 10Hz (cycles / second)

The Takeaway

- Model-based planning:
  - Planning performed at a *high level of abstraction* (compared to system behavior).
- Availability of a simulation as an 'oracle':
  - Abstractions relate planning model to simulation at lower level of abstraction.
  - Abstractions used to identify model errors.
- The detection of model errors can be automated; it is a challenge to automatically propose corrections to the model or the abstractions to eliminate errors.
  - Doing so enables reflection on planning models and therefore self-improvement.

Thank You!

Previous Work (Applications)



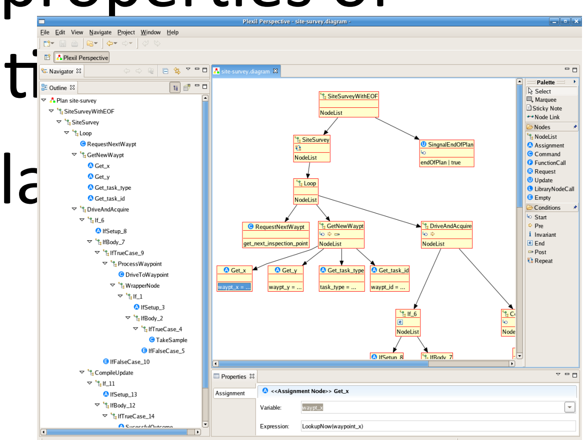
- Remote Agent [1], EO-1 [2]
  - Extensive model reviews.
  - Safety reviews to elicit potential hazards.
  - Automated tests stochastically generated by perturbations of nominal scenarios.
  - Executed on simulation platforms of varying fidelity where spacecraft, operations, and safety constraints were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

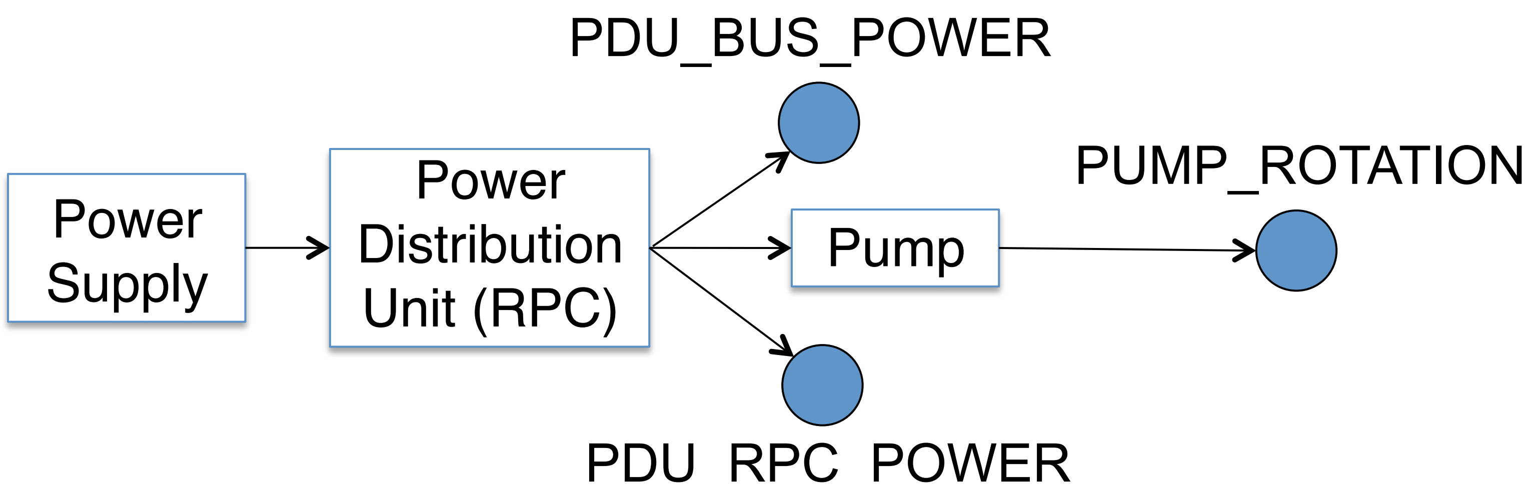
- VAL [6]
  - Given a plan and a model, determines whether the plan satisfies the constraints in the domain.
  - Limited ability to automatically fix plans.
- Model checking as plan verification [7]
  - Employs Java Pathfinder to check properties of PLEXIL, a language and plan executi
  - Requires a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



Previous Work (Summary)

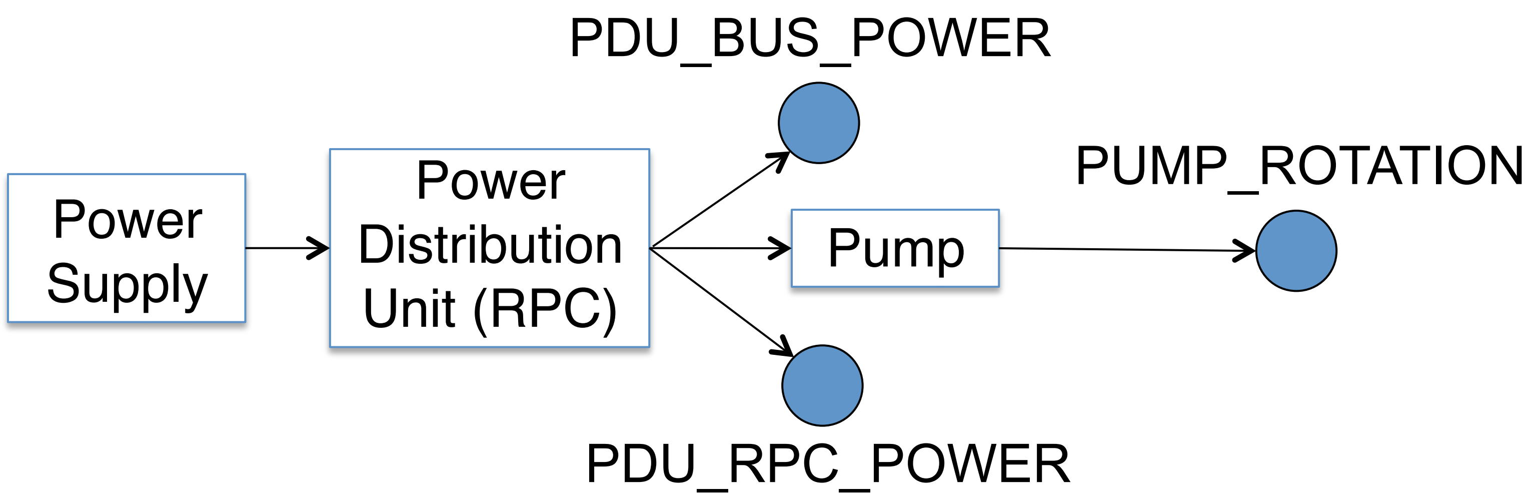
- There are tools to assist in verification of plans against planning models.
- There are tools to assist in test case generation and model verification.
- *Few to no tools* to assist in validation of models.
- *No tools* to assist in validation against simulations.

Diagnostic Reasoning – Example



	PDU_Bus_Power	Pump_Rotation	PDU_RPC_Power
Pump.Mechanical_Failure		1	
Power_Supply.Fail_to_Gen_Power	1	1	1
RPC.Fail_Open		1	1

Diagnostic Reasoning – Example

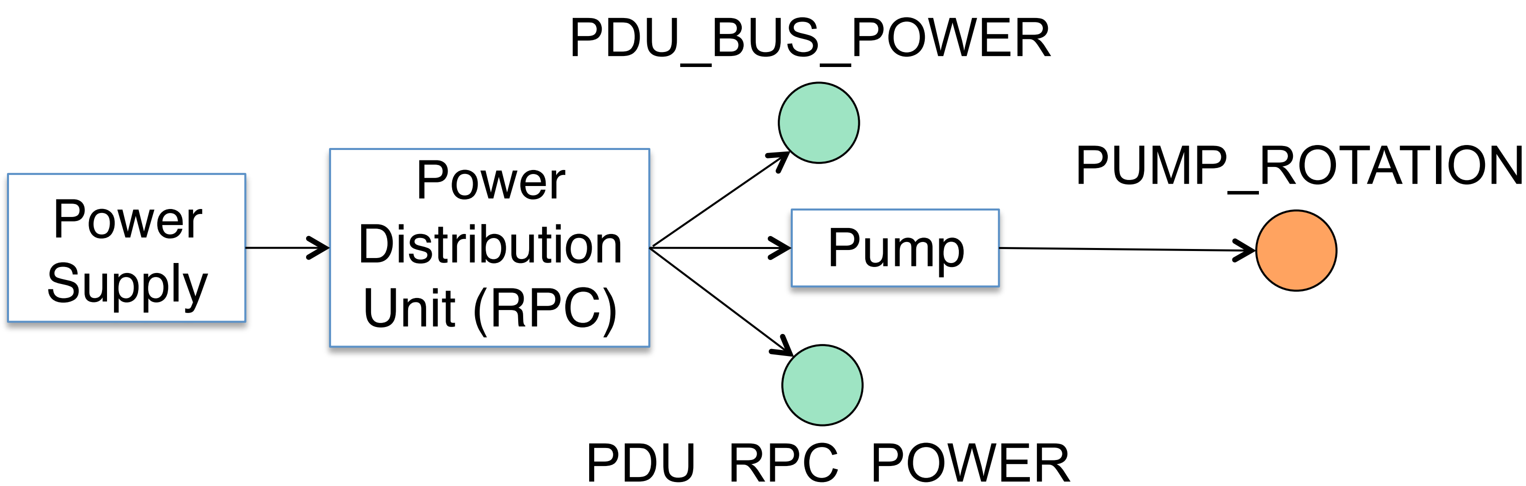


FAIL Test

PASS Test

	PDU_Bus_Power	Pump_Rotation	PDU_RPC_Power
Pump.Mechanical_Failure		1	
Power_Supply.Fail_to_Gen_Power	1	1	1
RPC.Fail_Open		1	1

Diagnostic Reasoning – Example

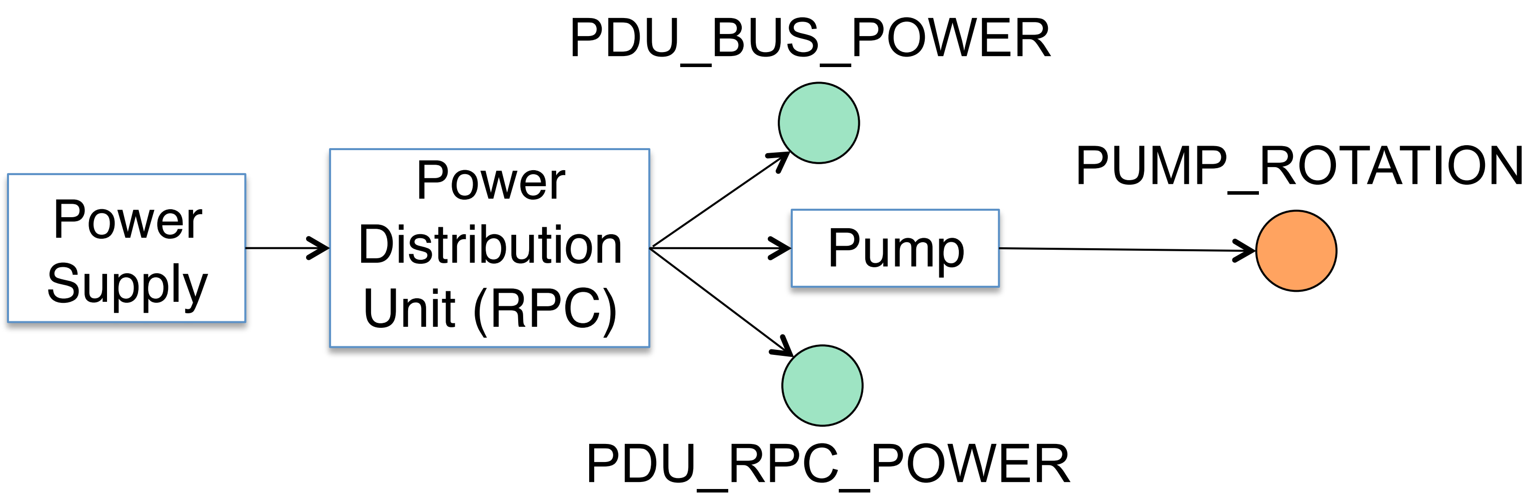


	PDU_Bus_Power	Pump_Rotation	PDU_RPC_Power
Pump.Mechanical_Failure		1	
Power_Supply.Fail_to_Gen_Power	1	1	1
RPC.Fail_Open		1	1

FAIL Test

PASS Test

Diagnostic Reasoning – Example



	_Power	otation	_Power
Pump.Mechanical_Failure		1	
Power_Supply.Fail_to_Gen_Power	1	1	1
RPC.Fail_Open		1	1

Exo
Powe

Must be Pump Mechanical Failure!

ion

FAIL Test

PASS Test

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.

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

[16] P. Morris, M. Do, R. McCann, L. Spirkovska, M. Schwabacher, J. Frank. Determining Mission Effects of Equipment Failures. Proceedings of AIAA Space, 2014.

[17] G. Aaseng, E. Barszcz, H. Valdez, and H. Moses. Scaling Up Model-Based Diagnostic and Fault Effects Reasoning for Spacecraft. Proceedings of AIAA Space 2015, Pasadena, CA, August 2015