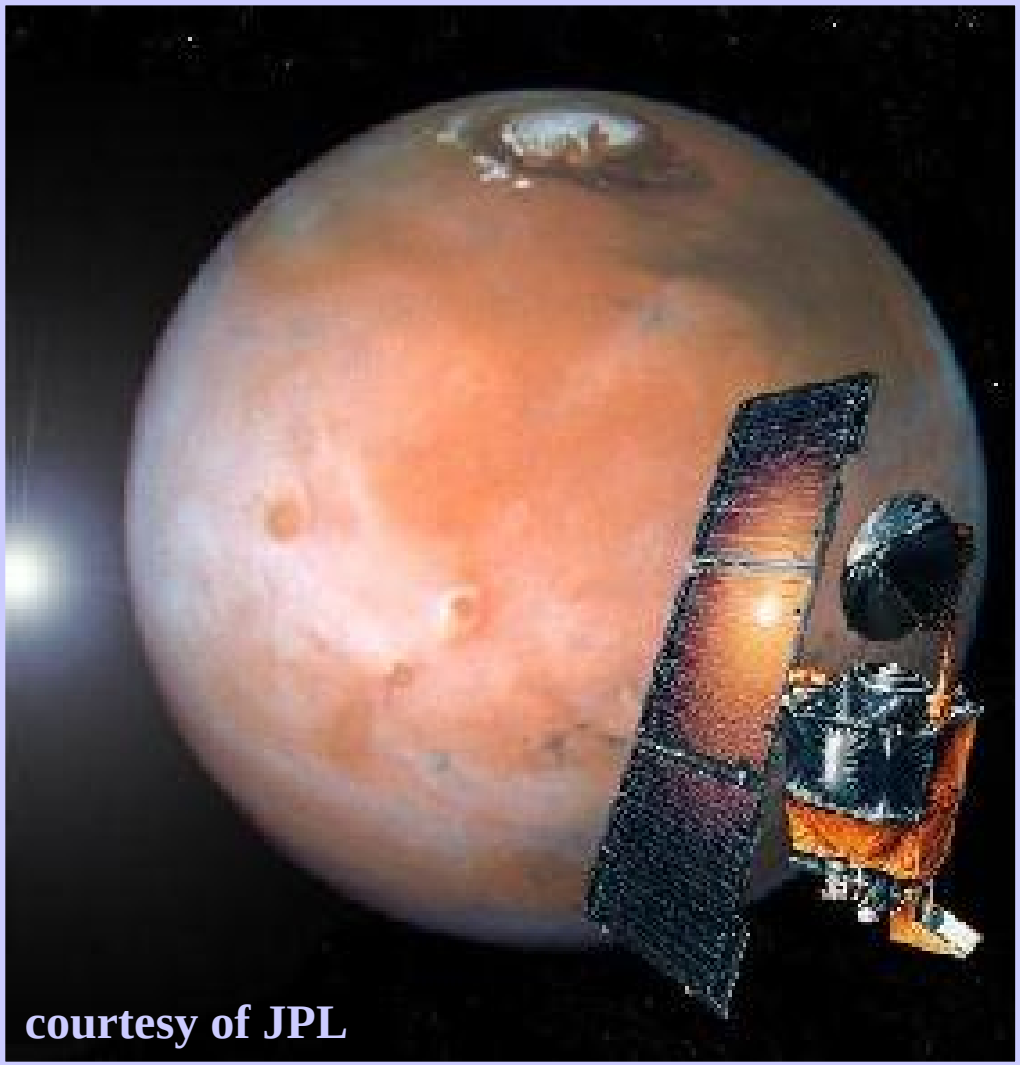
Model-based Programming: From Embedded Systems To Robotic Space Explorers

Brian C. Williams

CSAIL

Massachusetts Institute of Technology

Failures Highlight The Challenge of Robustness



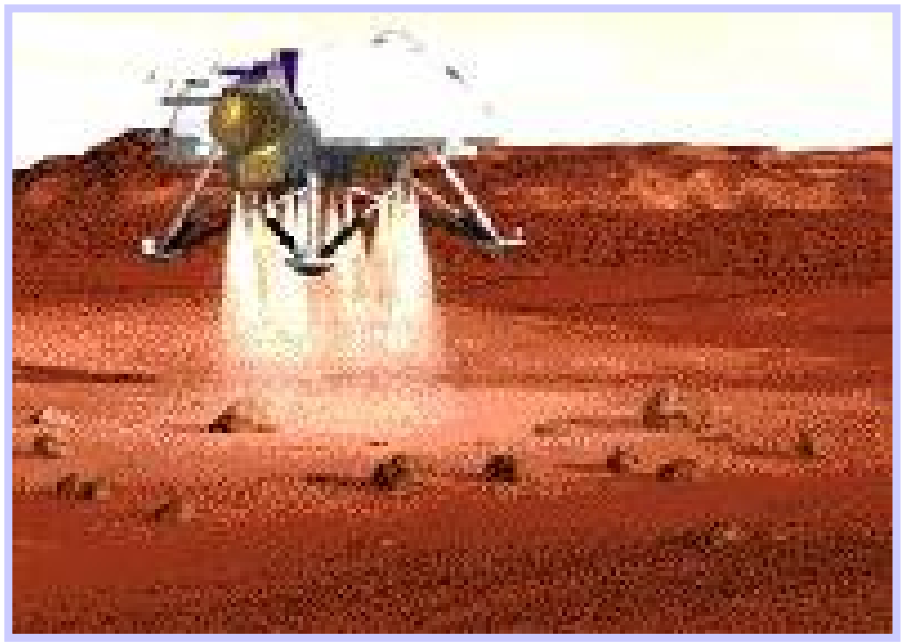
- **Clementine**
- **Mars Climate Orbiter**
- **Mars Orbiter**
- **Mars Polar Lander**

Complexity Is In Coordinating Subsystems

Large collections of devices must work in concert to achieve goals

- Devices indirectly observed and controlled.
- Must manage large levels of redundancy.
- Need quick, robust response to anomalies throughout life.

Mars Polar Lander Failure



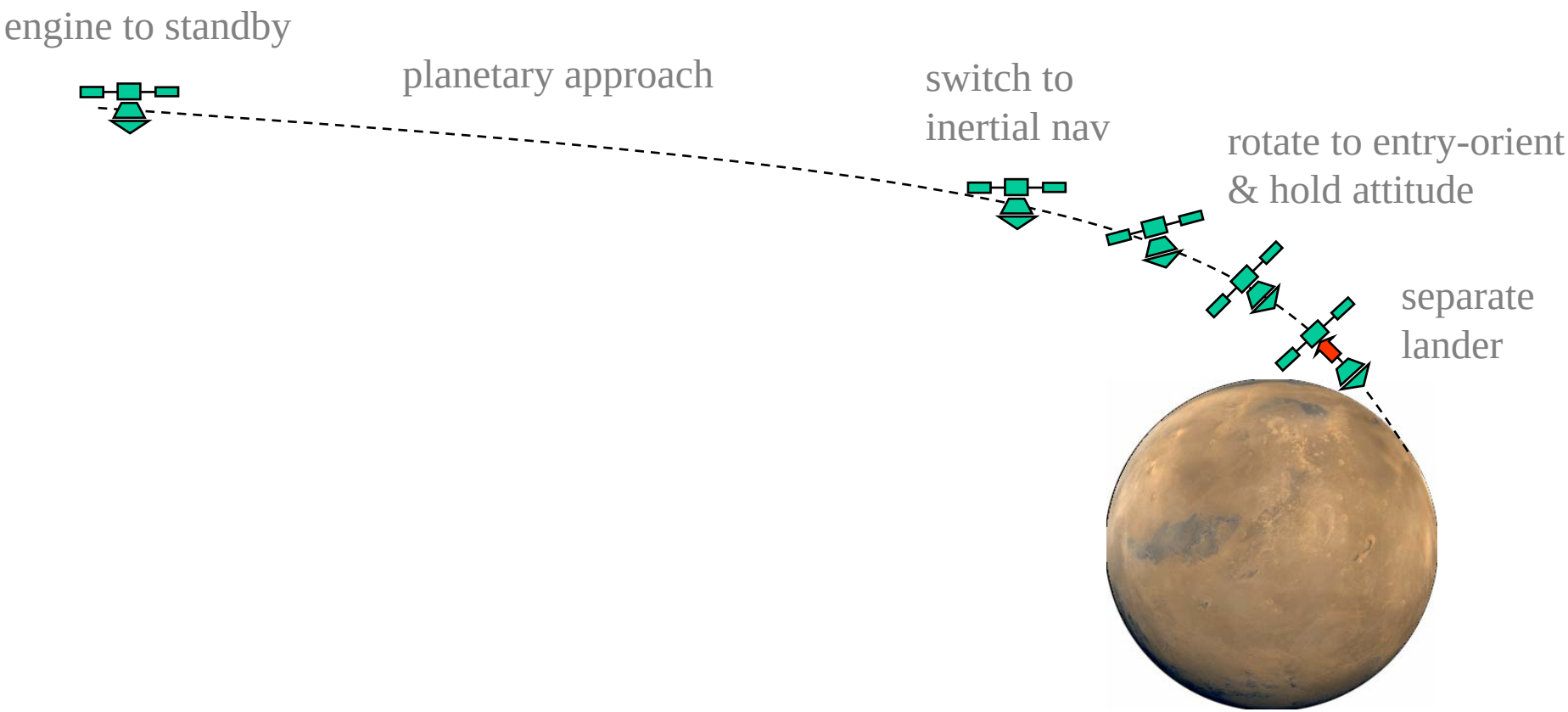
Leading Diagnosis:

- Legs deployed during descent.
- Noise spike on leg sensors latched by software monitors.
- Laser altimeter registers 50ft.
- Begins polling leg monitors to determine touch down.
- Latched noise spike read as touchdown.
- Engine shutdown at ~50ft.

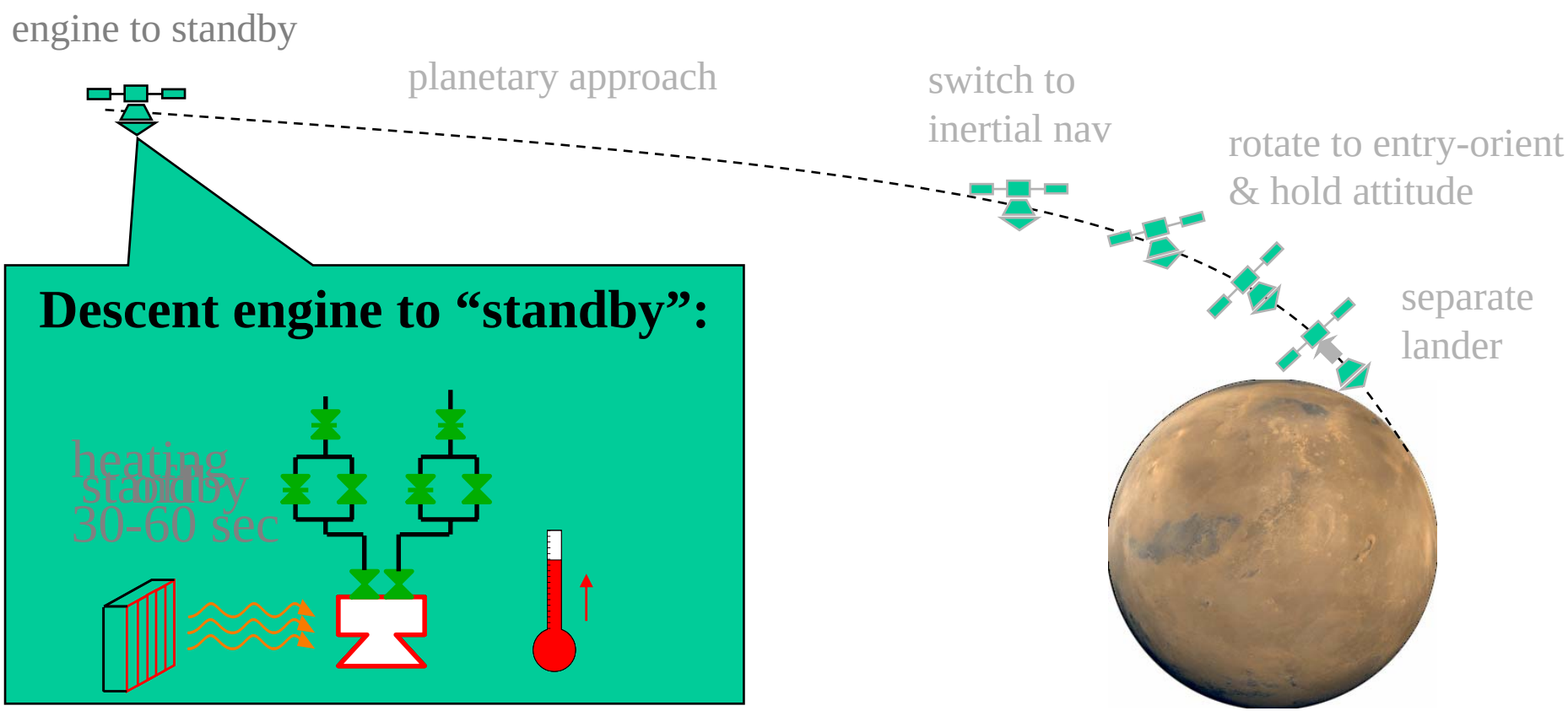
Programmers are overwhelmed by the bookkeeping of reasoning about unlikely hidden states

Fault Aware Systems:
Create embedded languages
That reason and coordinate
on the fly from models

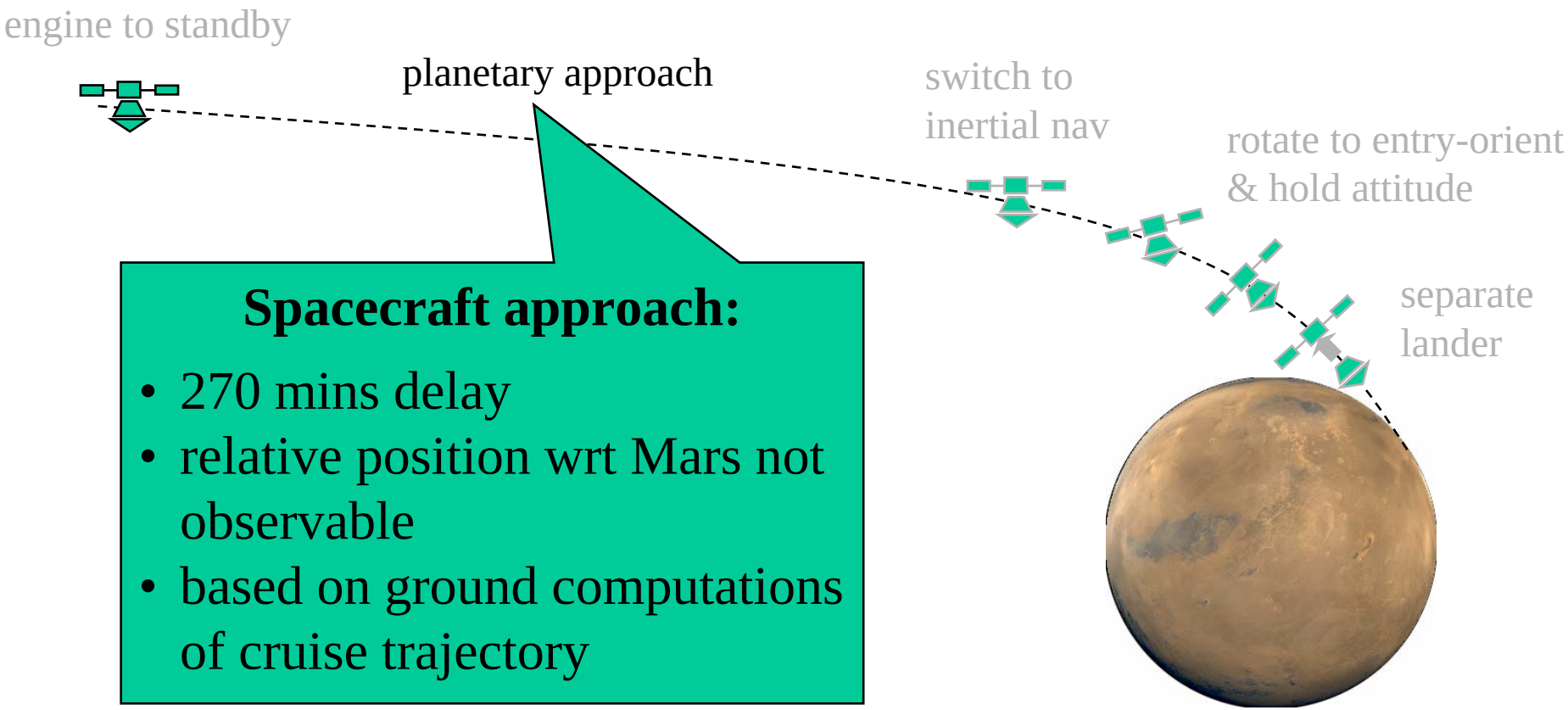
Mission Design Begins With A Storyboard



Mission Storyboards
Specify Evolving States



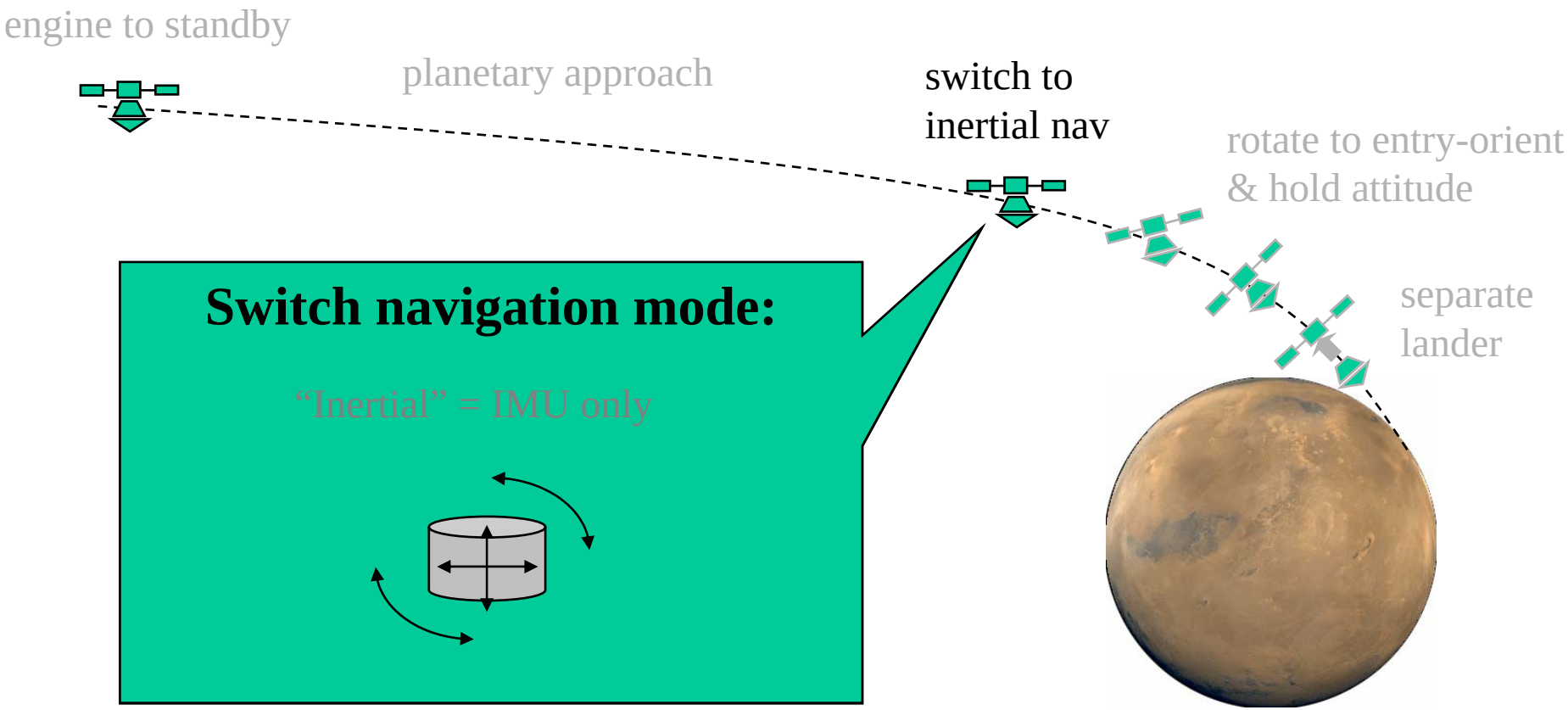
Mission Storyboards
Specify Evolving States



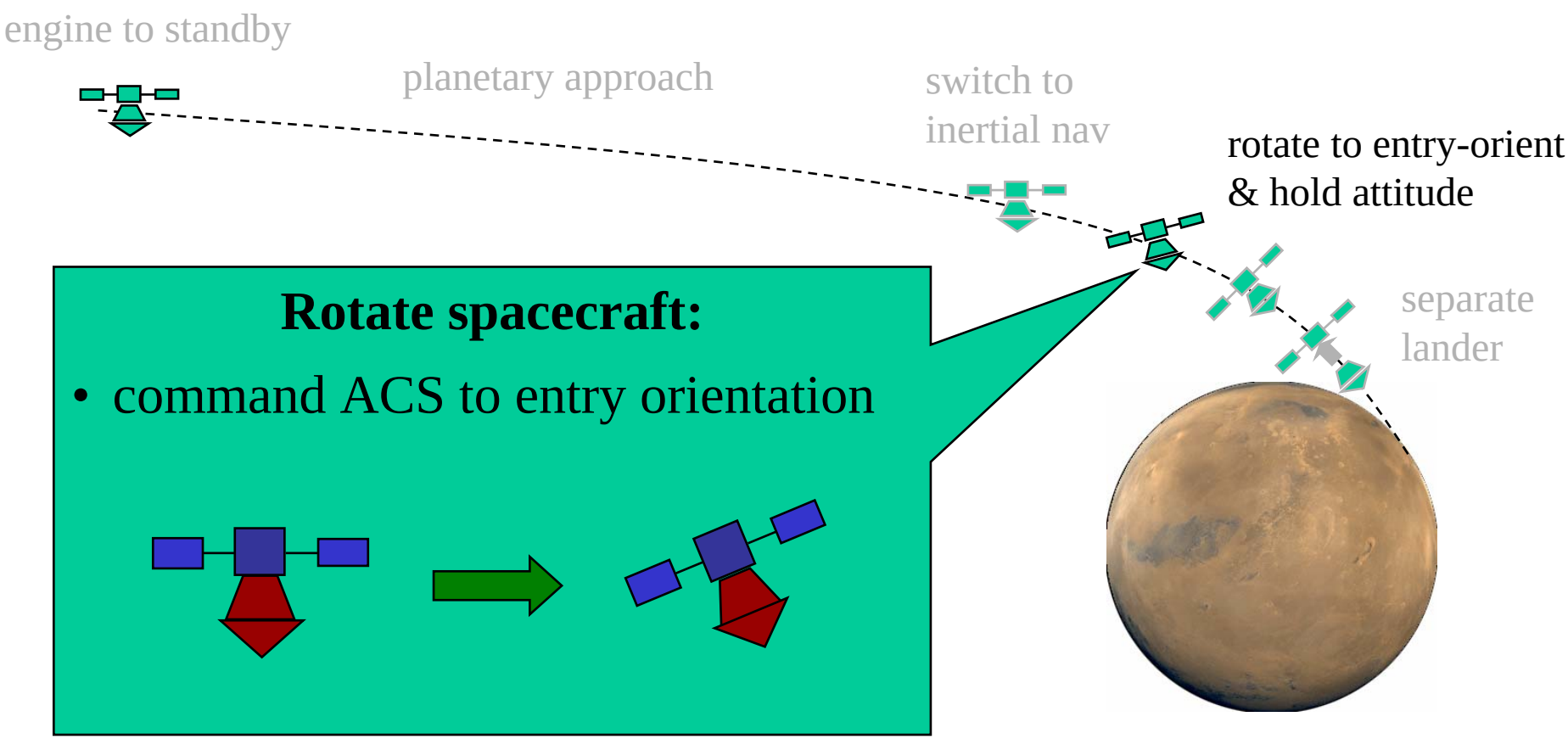
Spacecraft approach:

- 270 mins delay
- relative position wrt Mars not observable
- based on ground computations of cruise trajectory

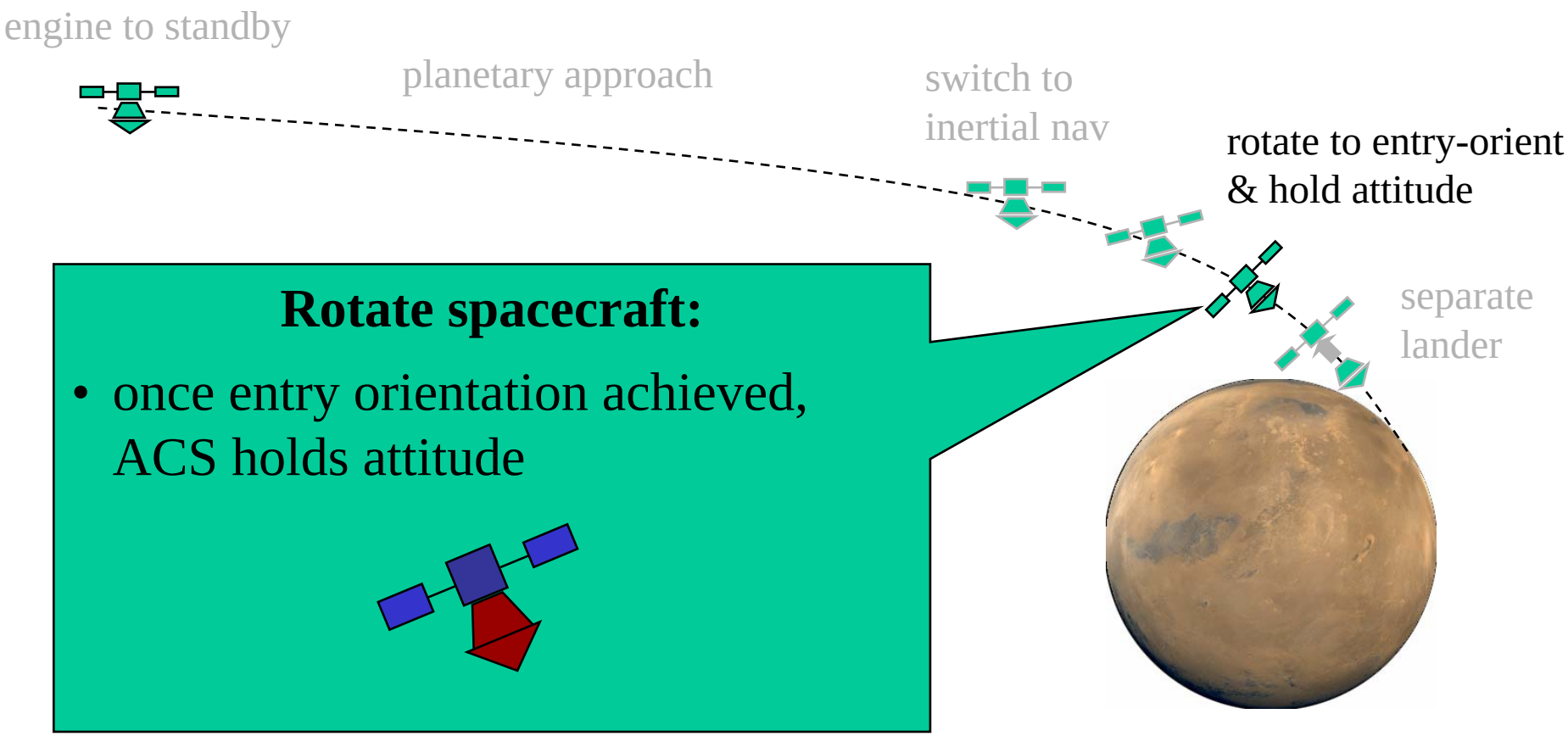
Mission Storyboards
Specify Evolving States



Mission Storyboards
Specify Evolving States



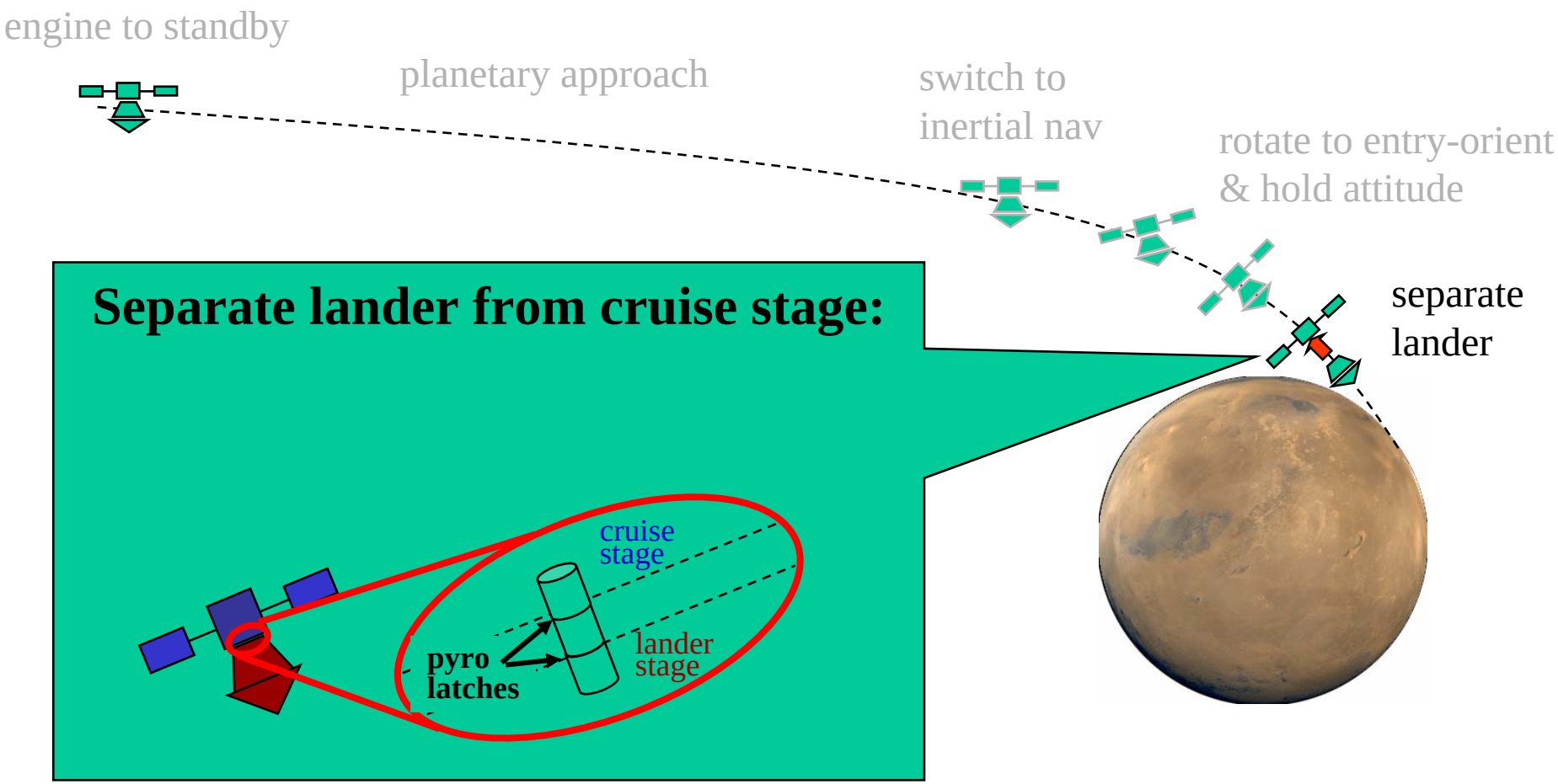
Mission Storyboards
Specify Evolving States



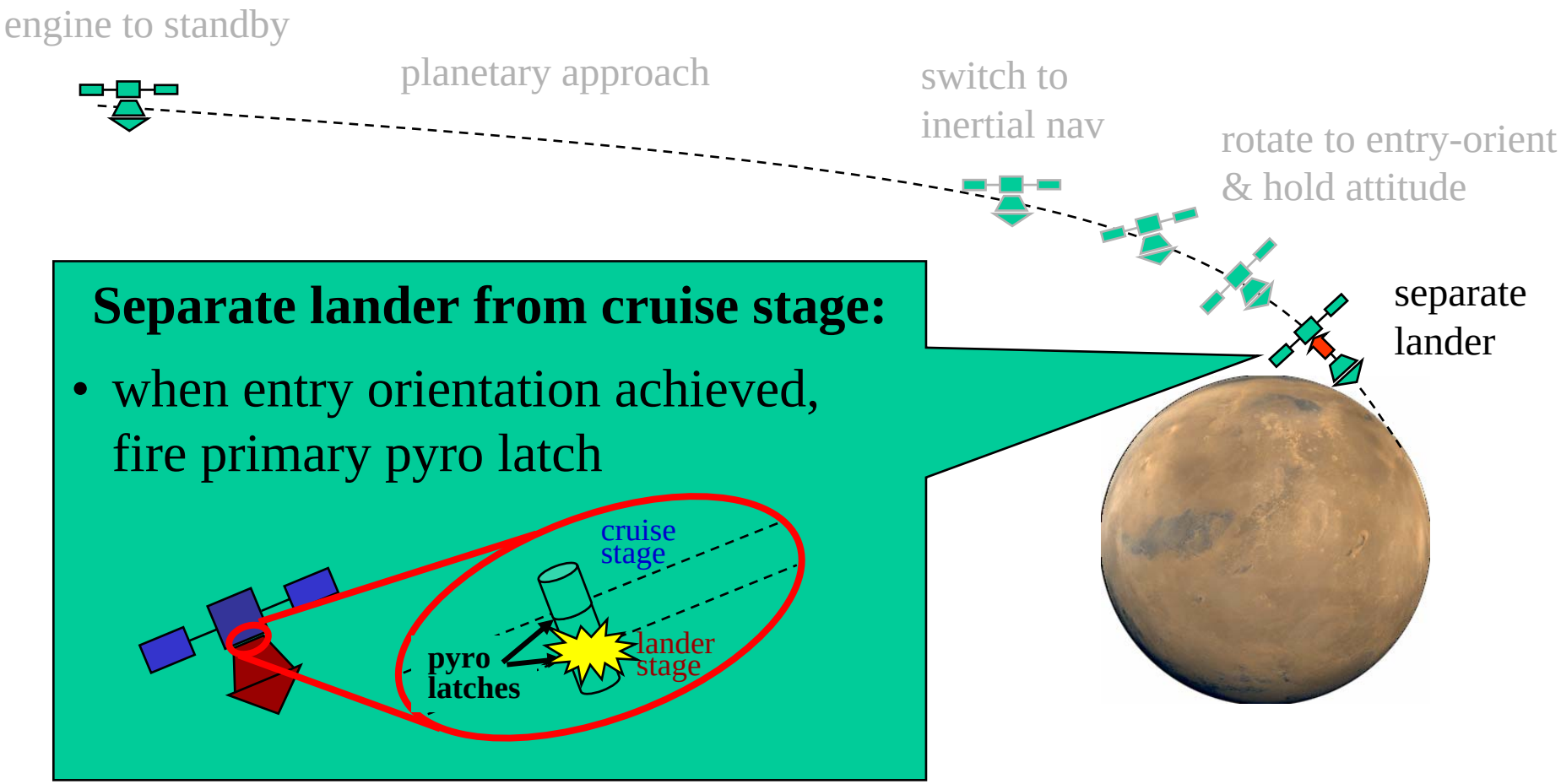
Rotate spacecraft:

- once entry orientation achieved, ACS holds attitude

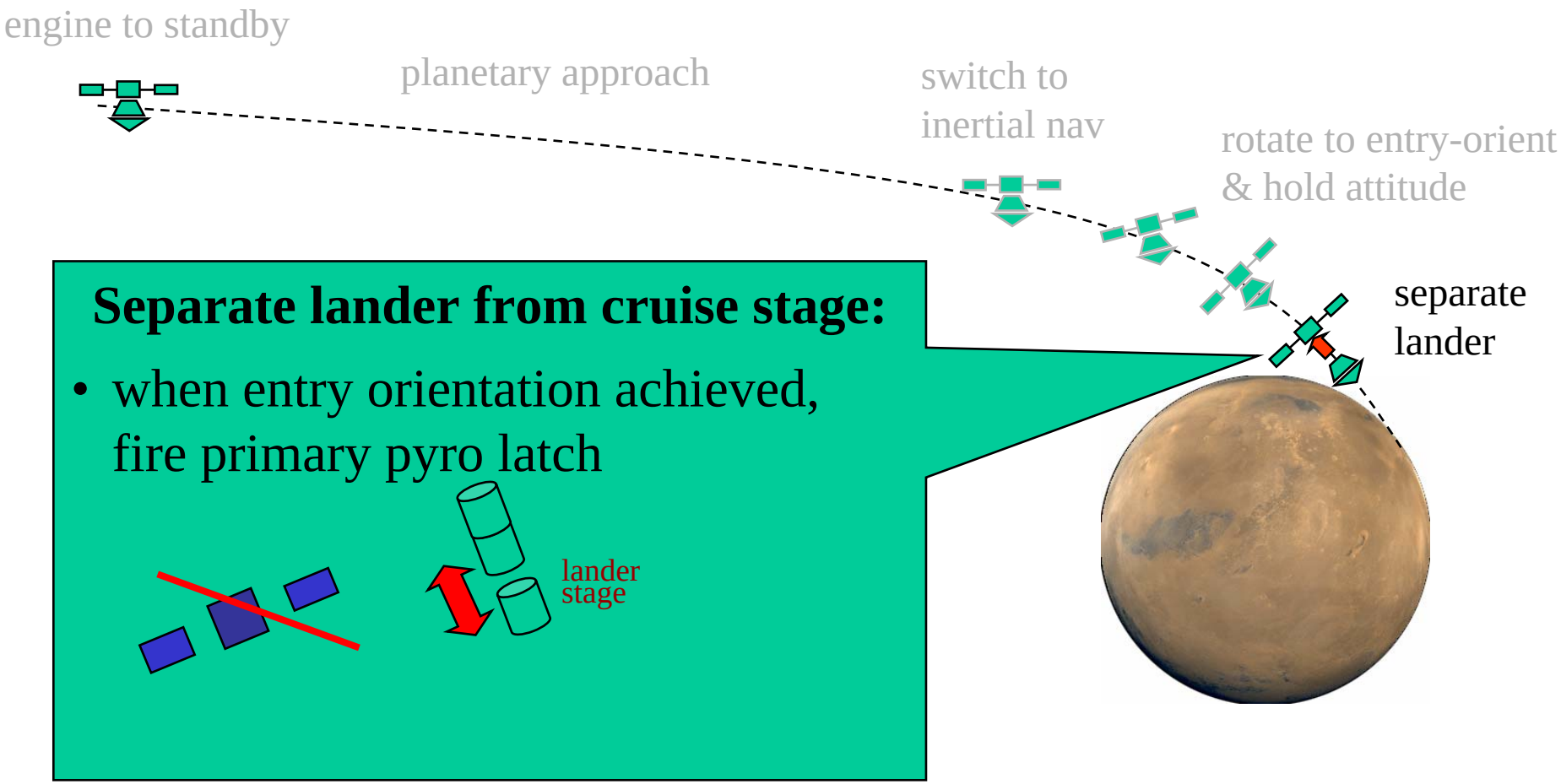
Mission Storyboards
Specify Evolving States



Mission Storyboards
Specify Evolving States



Specify Evolving States



Storyboards Elaborated With Failure Scenarios

gine to standby

planetary approach

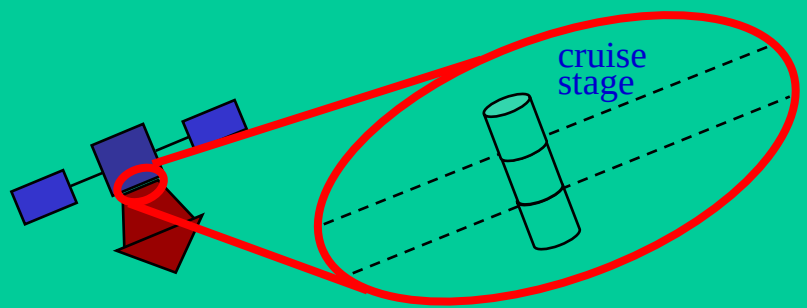
switch to inertial nav

rotate to entry-orient & hold attitude

separate lander

Separate lander from cruise stage:

- in case of failure of primary latch, fire backup pyro latch



Storyboards Elaborated With Failure Scenarios

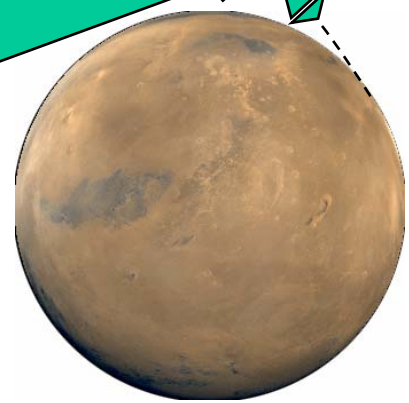
gine to standby

planetary approach

switch to inertial nav

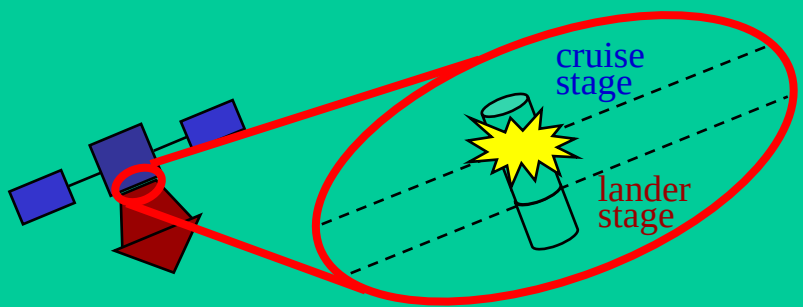
rotate to entry-orient & hold attitude

separate lander



Separate lander from cruise stage:

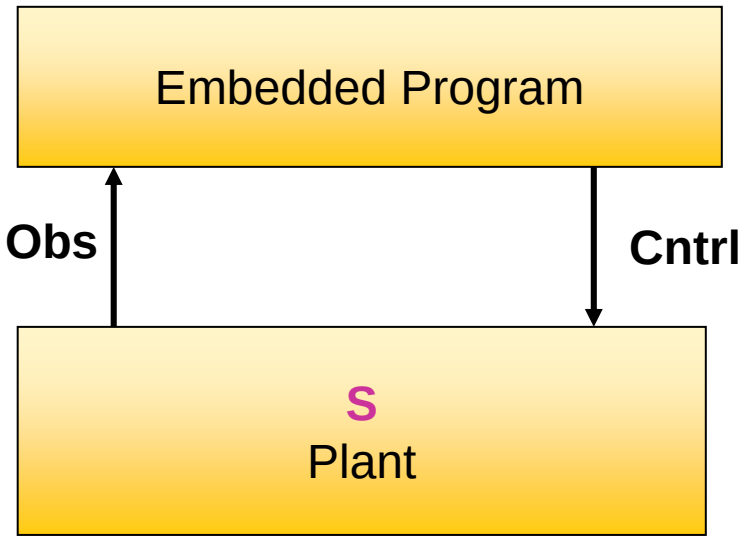
- in case of failure of primary latch, fire backup pyro latch



Like Storyboards, Model-based Programs Specify The Evolution of Abstract States

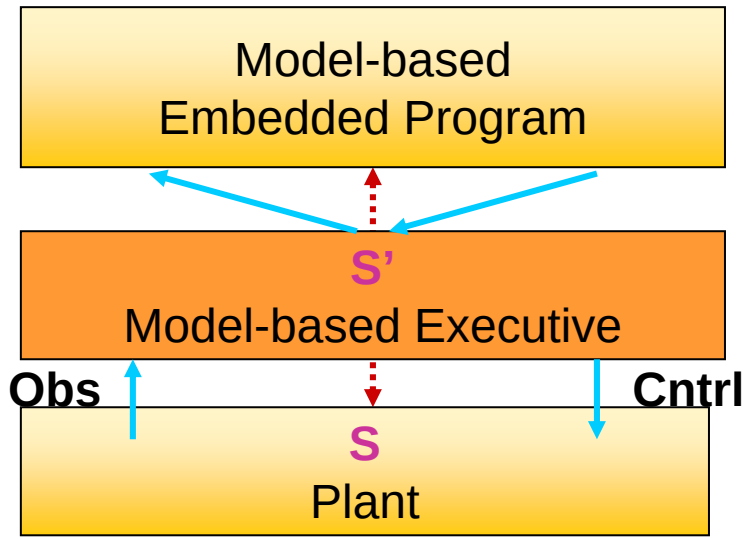
Embedded programs evolve actions by interacting with plant sensors and actuators:

- Read sensors
- Set actuators



Model-based programs evolve abstract states through direct interaction:

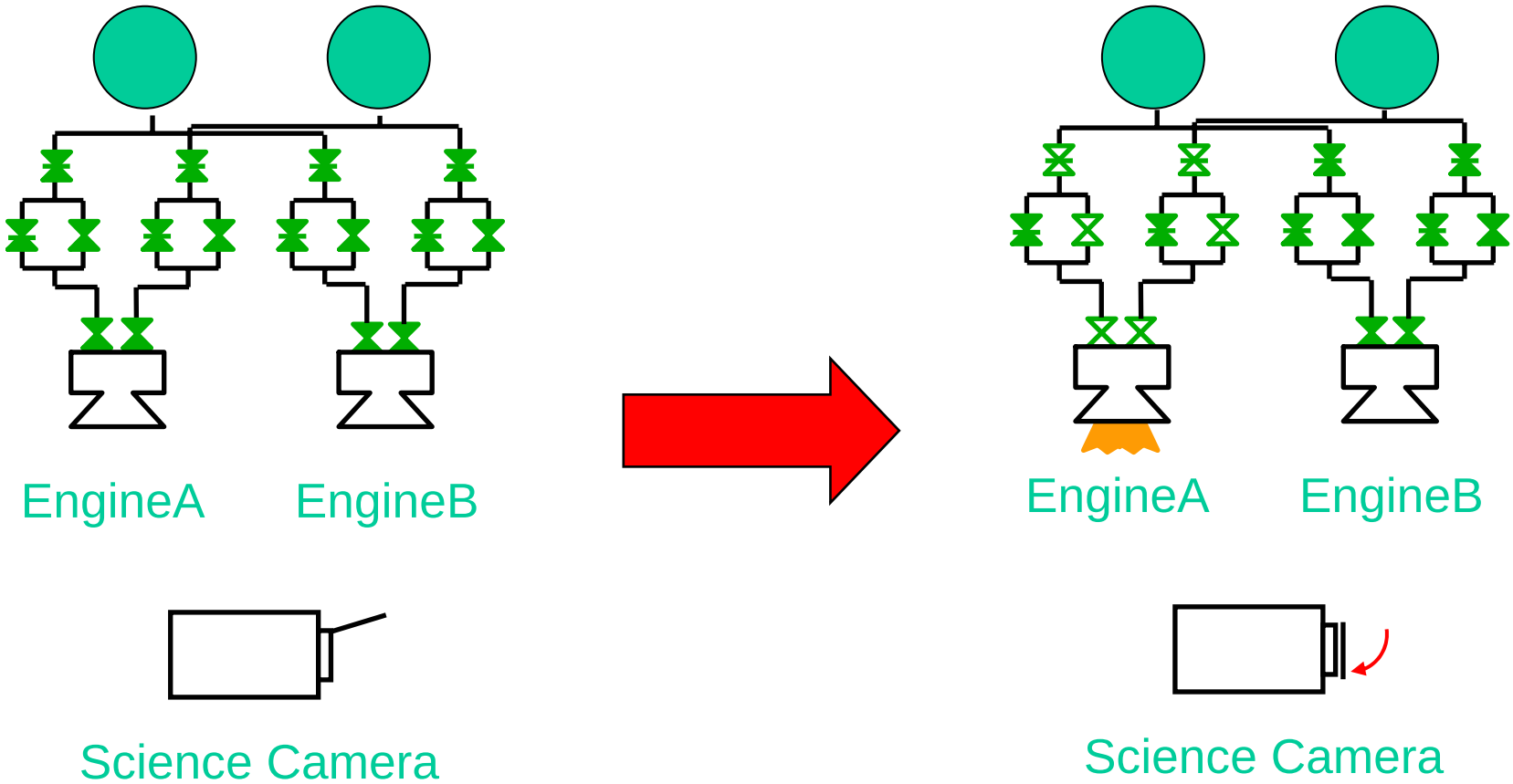
- Read abstract state
- Write abstract state



Model-based executive maps between state and sensors/actuators.

Descent Example

Turn camera off and engine on



Model-based Programs

Control program specifies state trajectories:

- fires one of two engines
- sets both engines to ‘standby’
- prior to firing engine, camera must be turned off to avoid plume contamination
- in case of primary engine failure, fire backup engine instead

Plant Model describes behavior of each component:

– Nominal and Off nominal

– qualitative constraints

– likelihoods and costs

```
OrbitInsert()::

(do-watching ((EngineA = Thrusting) OR
              (EngineB = Thrusting))
    (parallel
        (EngineA = Standby)
        (EngineB = Standby)
        (Camera = Off)
        (do-watching (EngineA = Failed)
            (when-donext ( (EngineA = Standby) AND
                           (Camera = Off) )
                (EngineA = Thrusting)))
        (when-donext ( (EngineA = Failed) AND
                       (EngineB = Standby) AND
                       (Camera = Off) )
            (EngineB = Thrusting))))
```

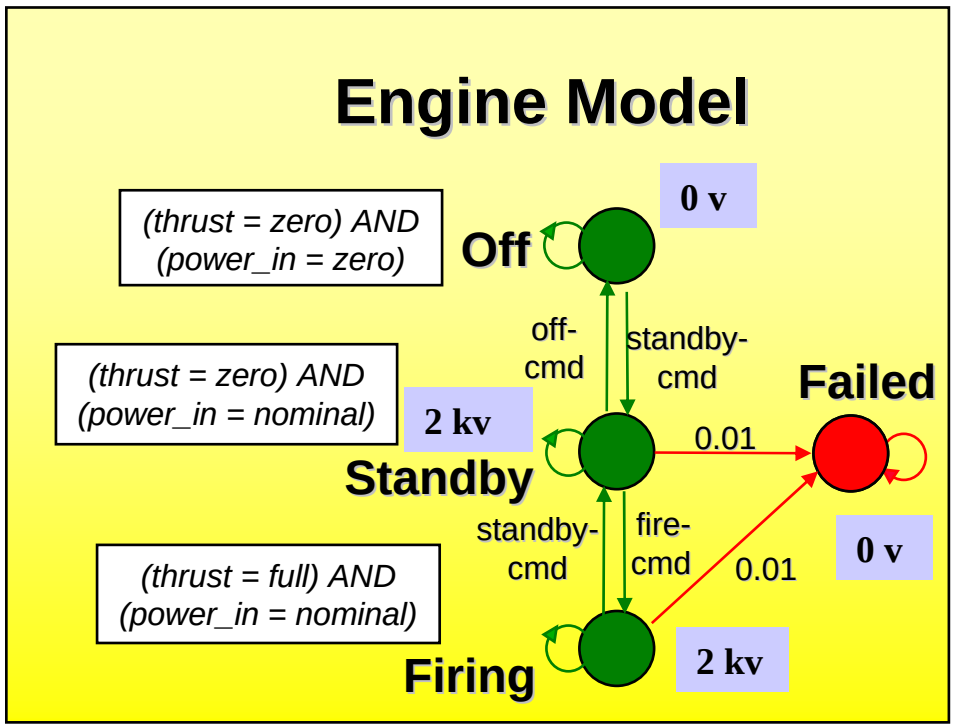
Plant Model

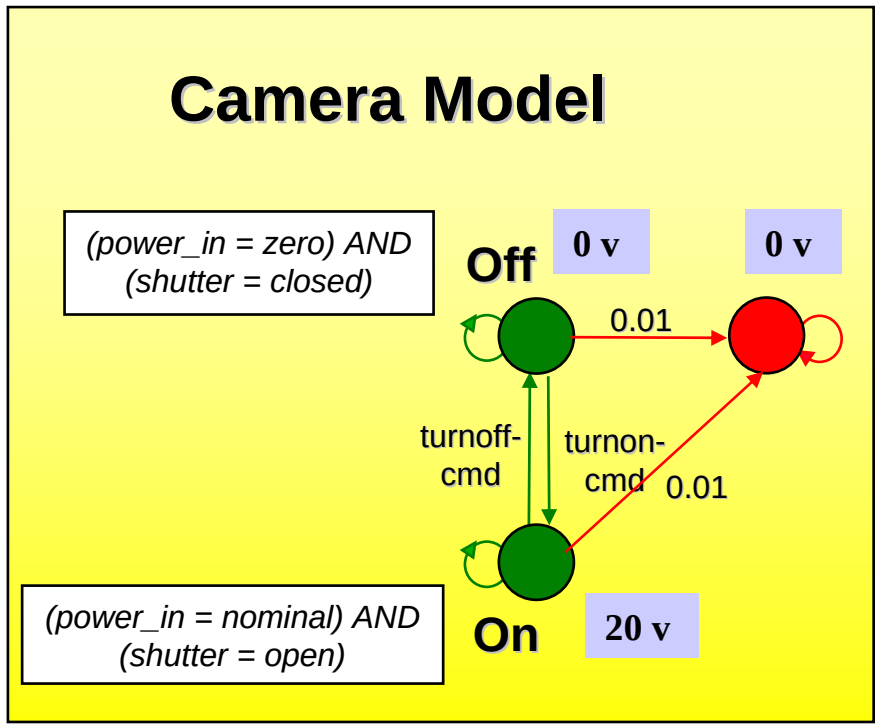
component modes…

described by finite domain constraints on variables…

deterministic and probabilistic transitions

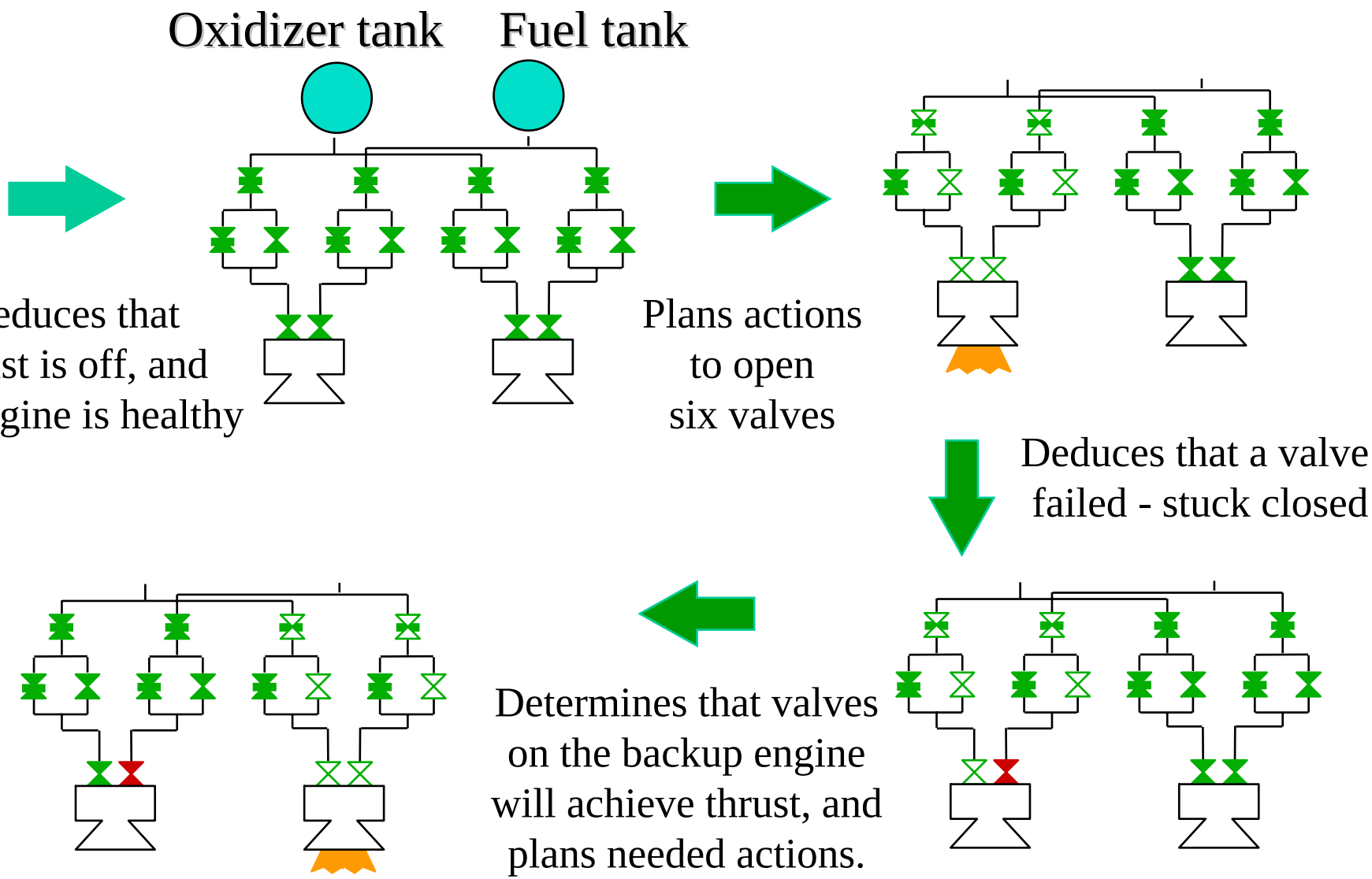
cost/reward





one per component … operating concurrently

State-based Execution: The model-based program sets the state to thrusting, and the deductive controller . . . .



RMI

Titan Model-based Executive

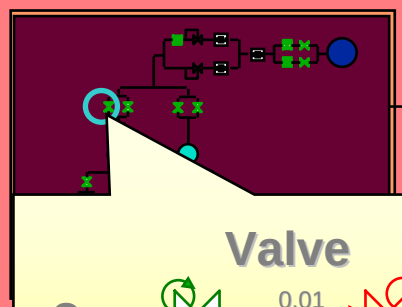
```
OrbitInsert()::
(do-watching ((EngineA = Firing) OR
              (EngineB = Firing))
    (parallel
        (EngineA = Standby)
        (EngineB = Standby)
        (Camera = Off)
        (do-watching (EngineA = Failed)
            (when-donext ( (EngineA = Standby) AND
                           (Camera = Off) )
                (EngineA = Firing)))
        (when-donext ( (EngineA = Failed) AND
                       (EngineB = Standby) AND
                       (Camera = Off) )
            (EngineB = Firing))))
```

Generates target goal states conditioned on state estimates

System Model

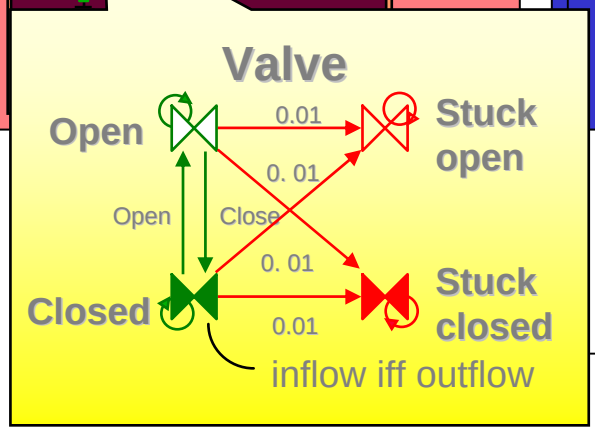
State estimates

State goals



Tracks likely plant states

Tracks least cost goal states



Observations

Commands

Plant