

---

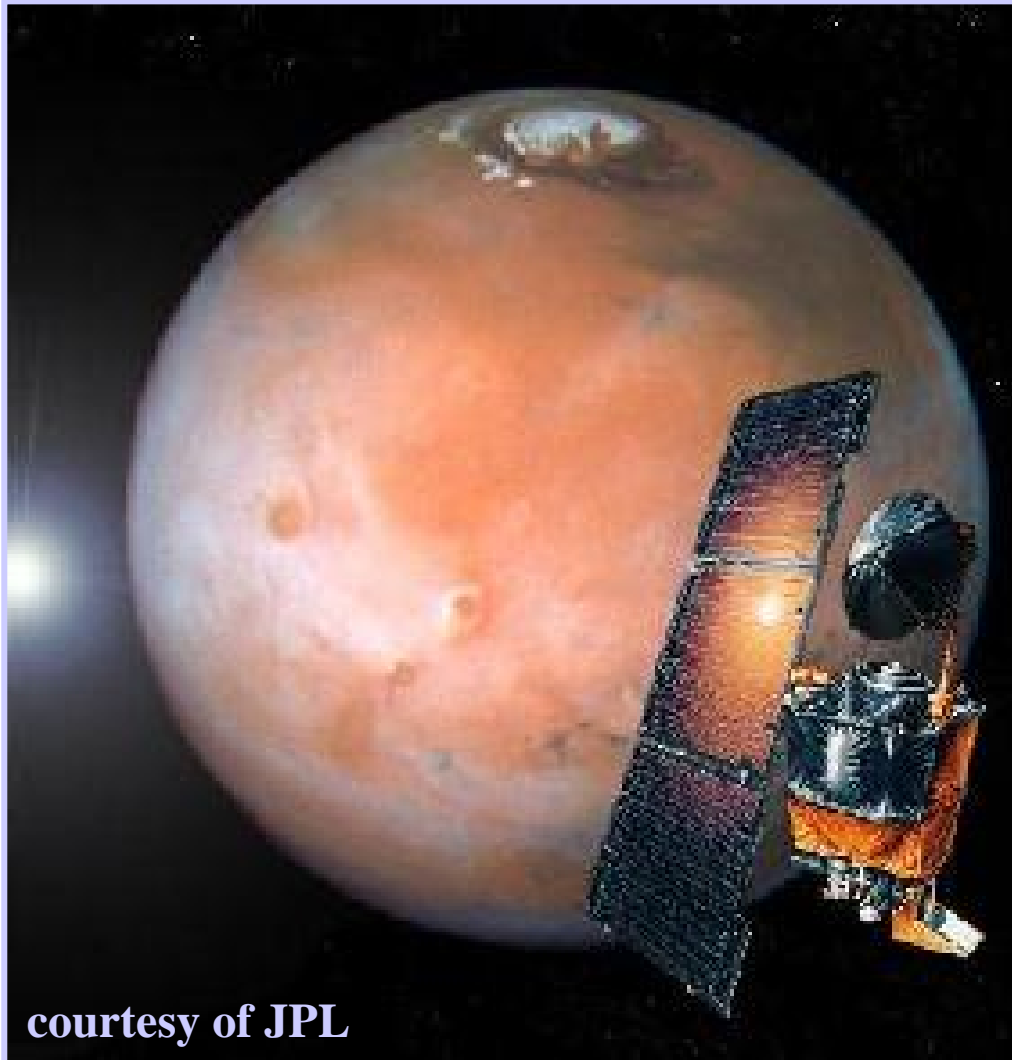
# Model-based Programming: From Embedded Systems To Robotic Space Explorers

Brian C. Williams

CSAIL

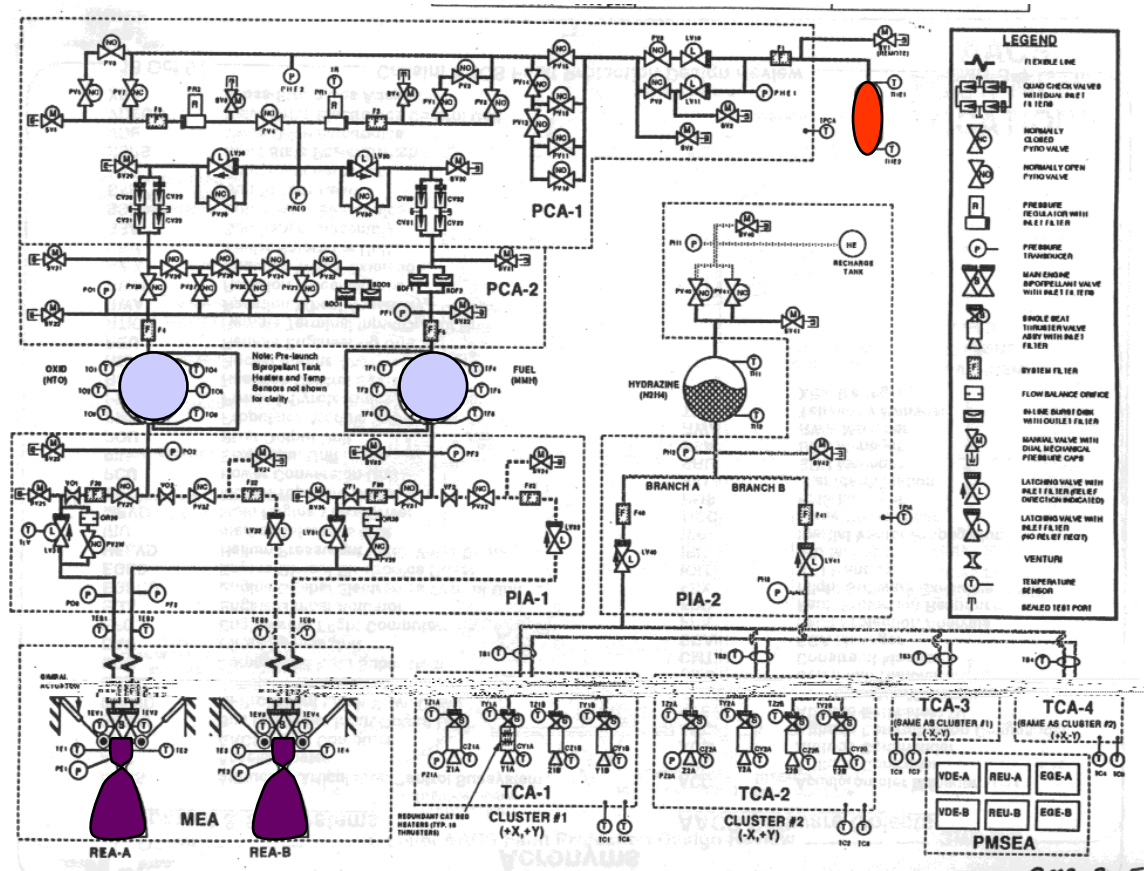
Massachusetts Institute of Technology

# Failures Highlight The Challenge of Robustness



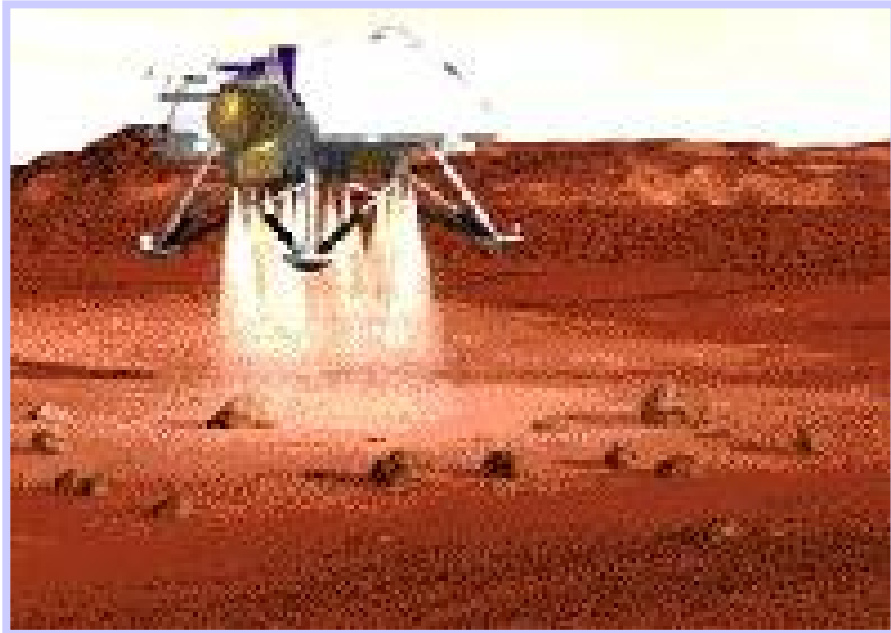
courtesy of JPL

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



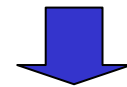
Large collections of devices must work in concert to achieve goals

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



## 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.



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



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

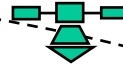
# Mission Design Begins With A Storyboard

engine to standby

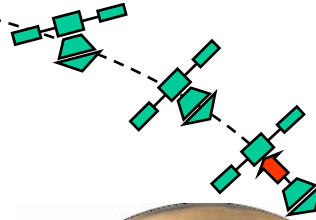


planetary approach

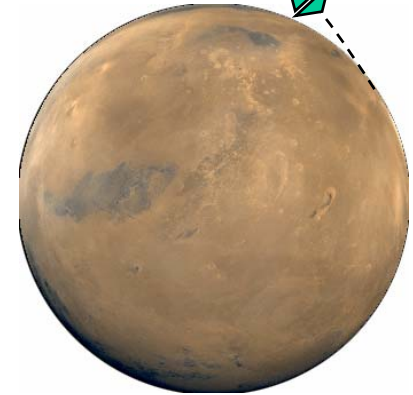
switch to  
inertial nav



rotate to entry-orient  
& hold attitude



separate  
lander



# Mission Storyboards Specify Evolving States

engine to standby

planetary approach

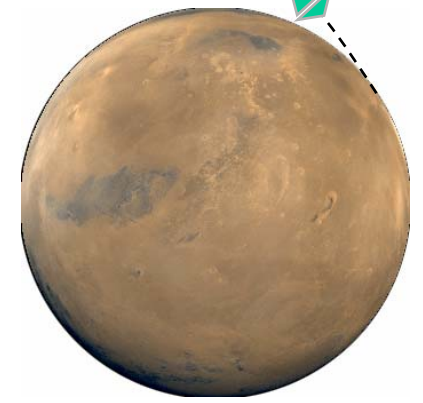
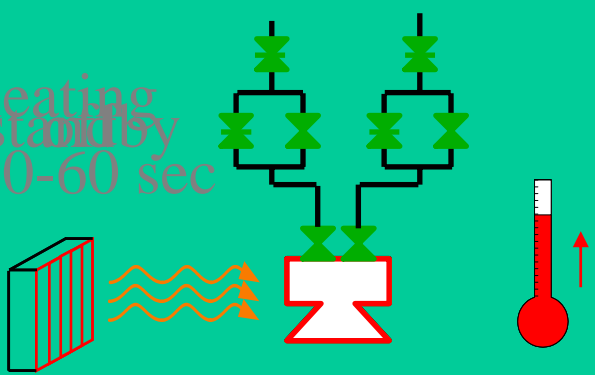
switch to  
inertial nav

rotate to entry-orient  
& hold attitude

separate  
lander

**Descent engine to “standby”:**

heating  
standby  
30-60 sec



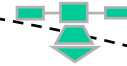
# Mission Storyboards Specify Evolving States

engine to standby

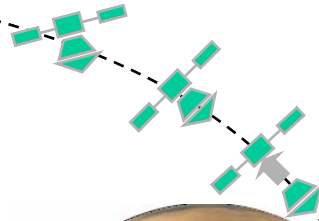


planetary approach

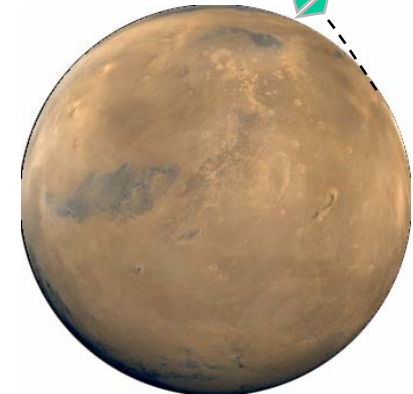
switch to  
inertial nav



rotate to entry-orient  
& hold attitude



separate  
lander



## Spacecraft approach:

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

# Mission Storyboards Specify Evolving States

engine to standby

planetary approach

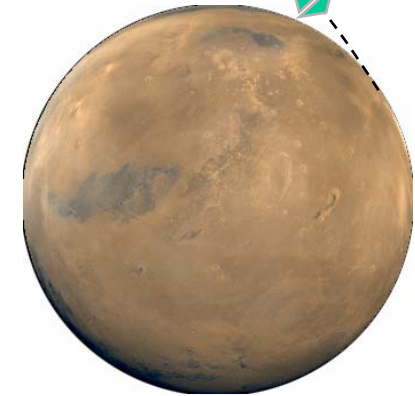
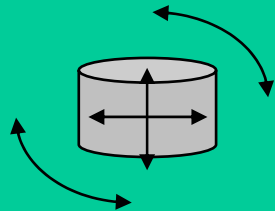
switch to  
inertial nav

rotate to entry-orient  
& hold attitude

separate  
lander

**Switch navigation mode:**

"Inertial" = IMU only





# Mission Storyboards Specify Evolving States

engine to standby

planetary approach

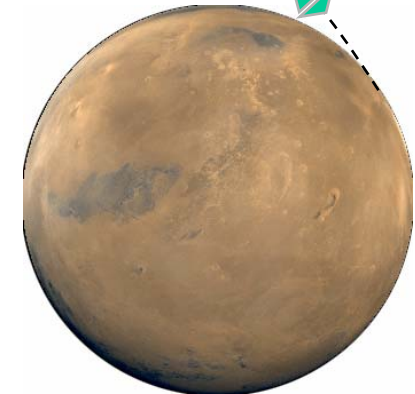
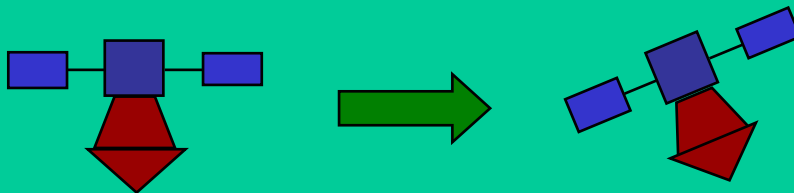
switch to  
inertial nav

rotate to entry-orient  
& hold attitude

separate  
lander

## Rotate spacecraft:

- command ACS to entry orientation



# Mission Storyboards Specify Evolving States

engine to standby

planetary approach

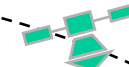
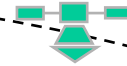
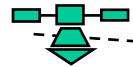
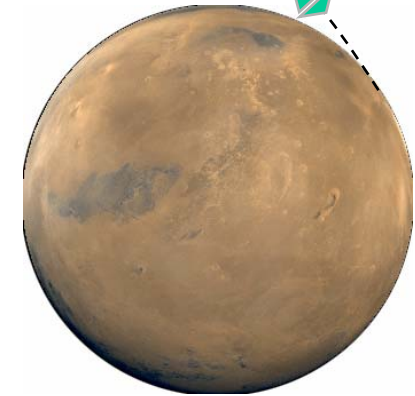
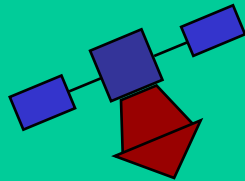
switch to  
inertial nav

rotate to entry-orient  
& hold attitude

separate  
lander

## Rotate spacecraft:

- once entry orientation achieved, ACS holds attitude



# Mission Storyboards Specify Evolving States

engine to standby

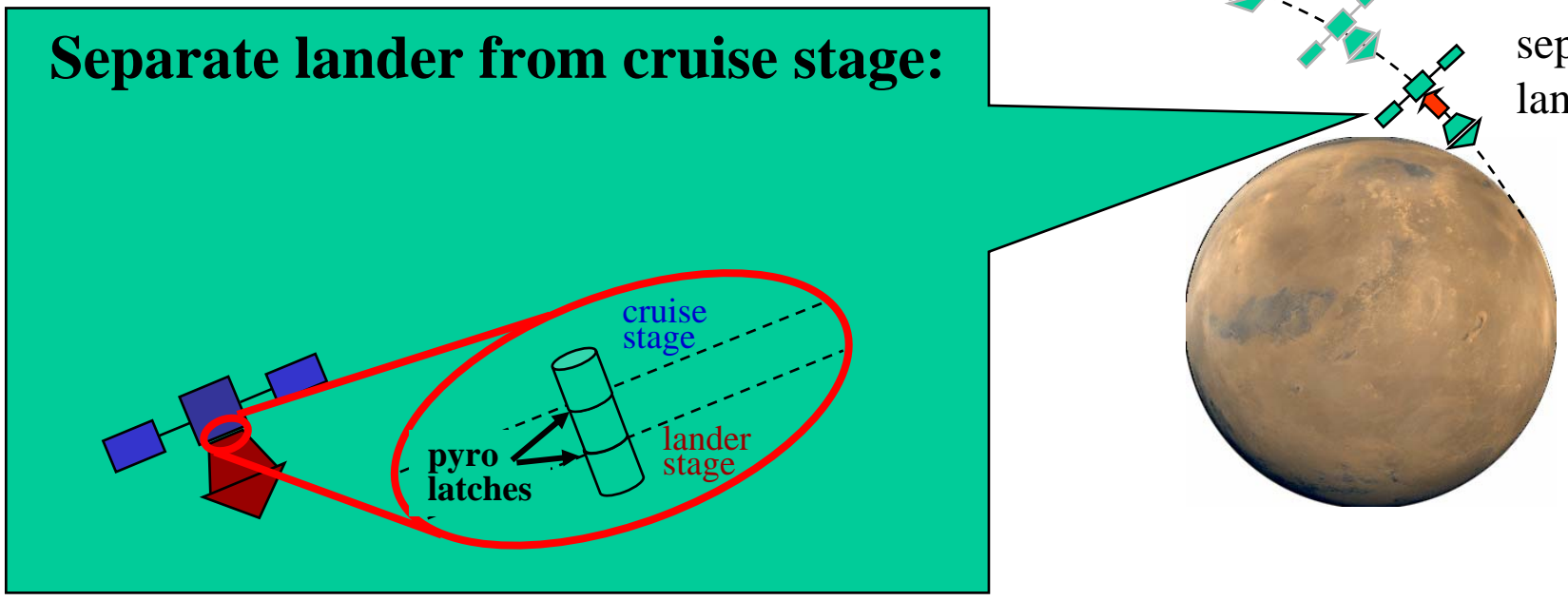
planetary approach

switch to  
inertial nav

rotate to entry-orient  
& hold attitude

separate  
lander

**Separate lander from cruise stage:**



# Mission Storyboards

## Specify Evolving States

engine to standby

planetary approach

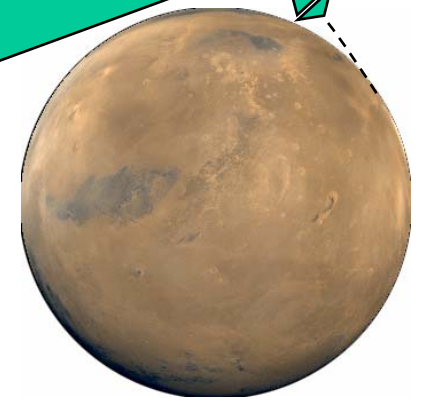
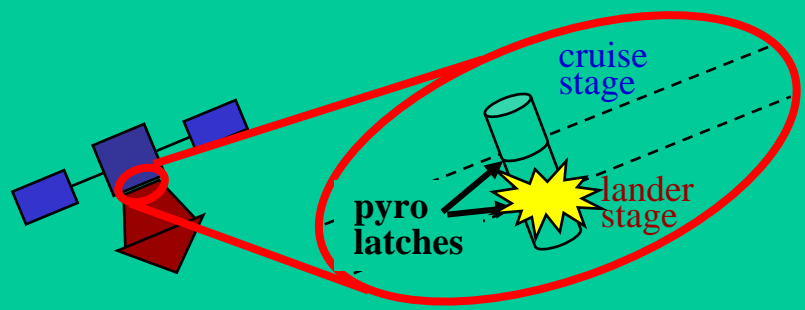
switch to inertial nav

rotate to entry-orient & hold attitude

separate lander

### Separate lander from cruise stage:

- when entry orientation achieved, fire primary pyro latch



engine to standby

planetary approach

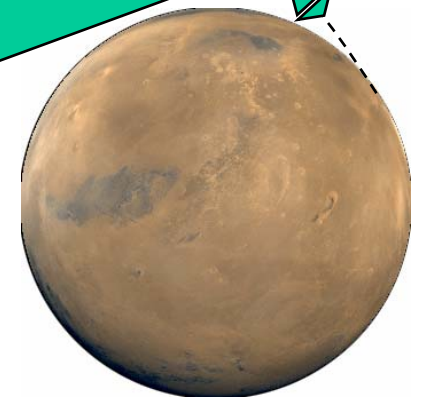
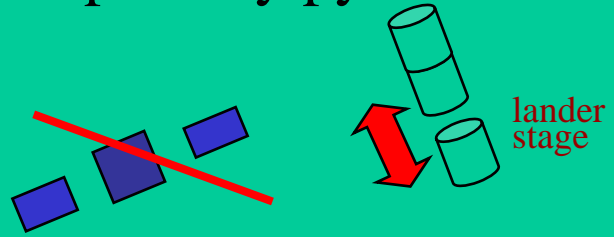
switch to inertial nav

rotate to entry-orient & hold attitude

separate lander

## Separate lander from cruise stage:

- when entry orientation achieved, fire primary pyro latch



engine to standby

planetary approach

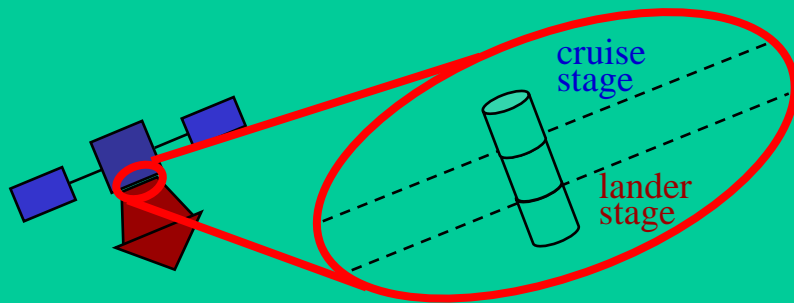
switch to  
inertial nav

rotate to entry-orient  
& hold attitude

separate  
lander

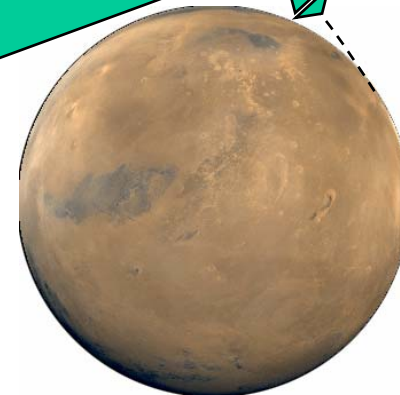
## Separate lander from cruise stage:

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



cruise  
stage

lander  
stage



engine to standby

planetary approach

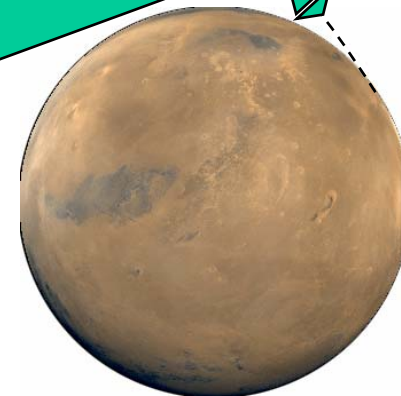
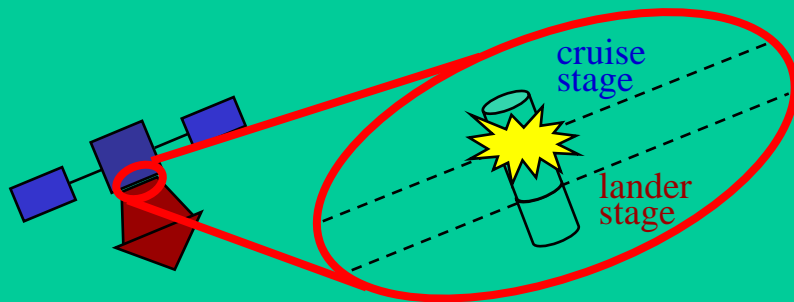
switch to  
inertial nav

rotate to entry-orient  
& hold attitude

separate  
lander

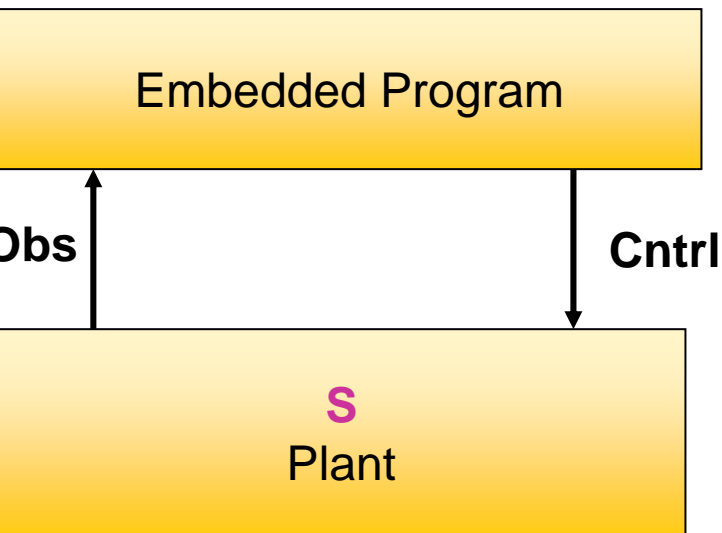
## Separate lander from cruise stage:

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



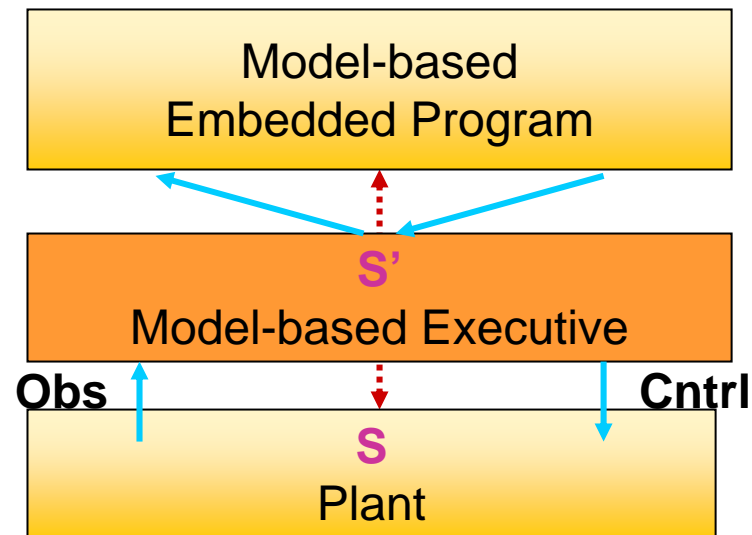
Embedded programs evolve actions by interacting with plant sensors and actuators:

- Read sensors
- Set actuators



Model-based programs evolve abstract states through direct interaction:

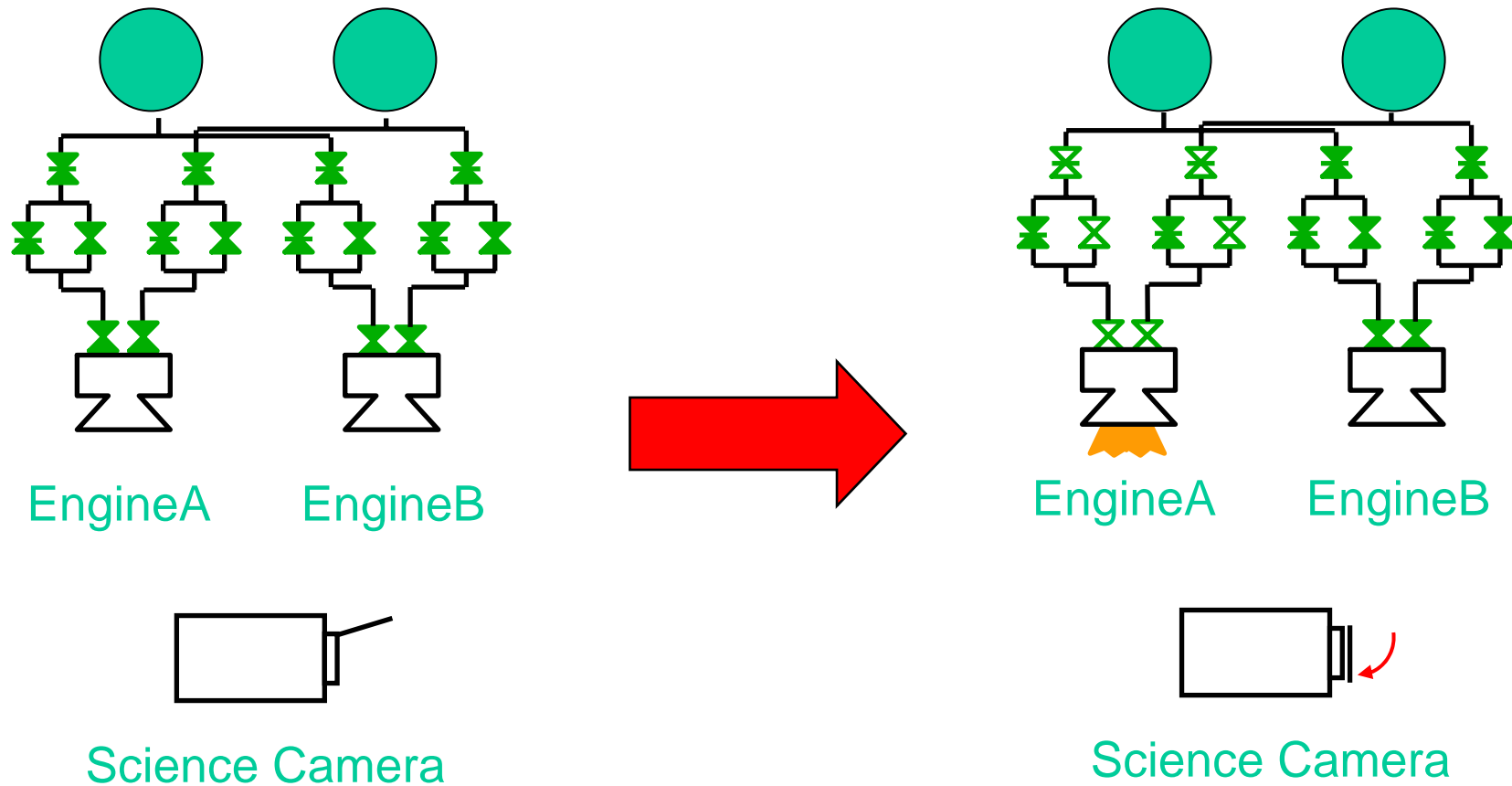
- Read abstract state
- Write abstract state



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



Turn camera off and engine on



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))))
```

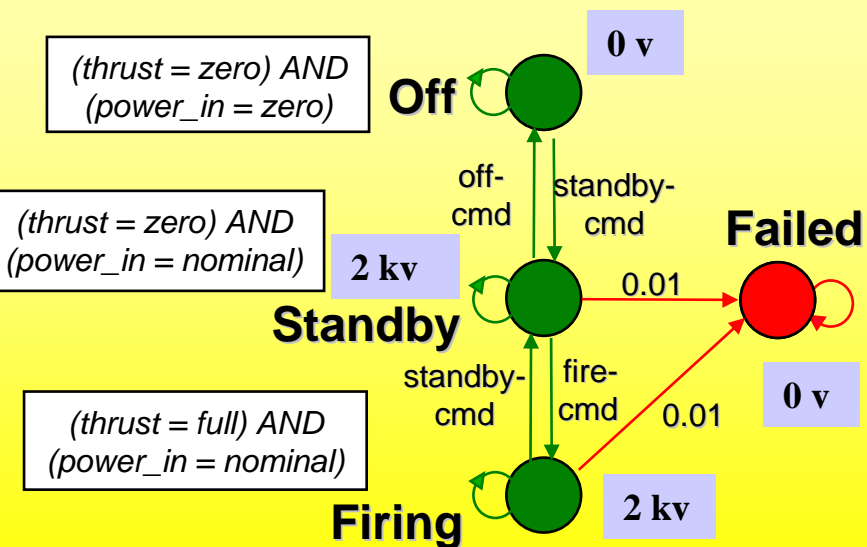
component modes...

described by finite domain constraints on variables...

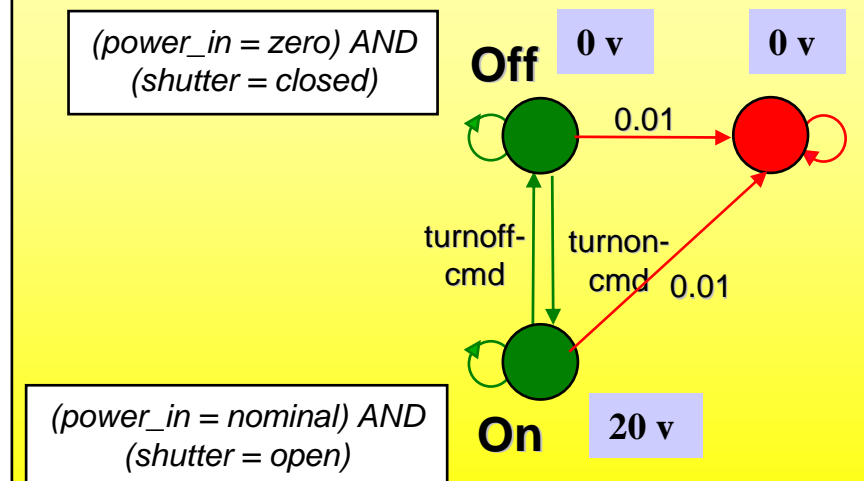
deterministic and probabilistic transitions

cost/reward

## Engine Model



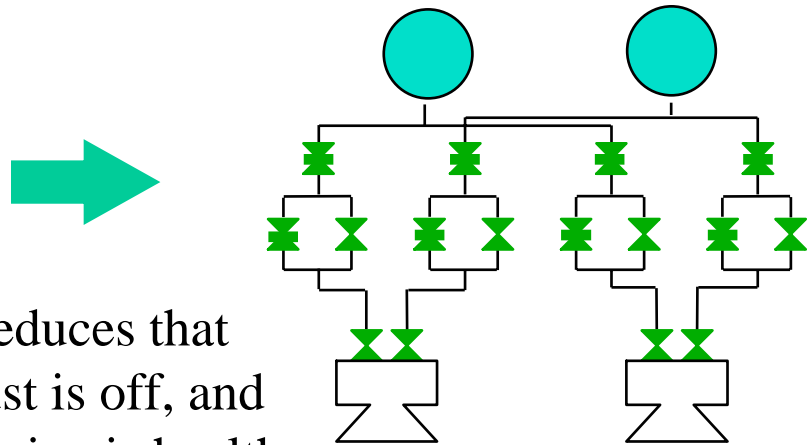
## Camera Model



one per component ... operating concurrently

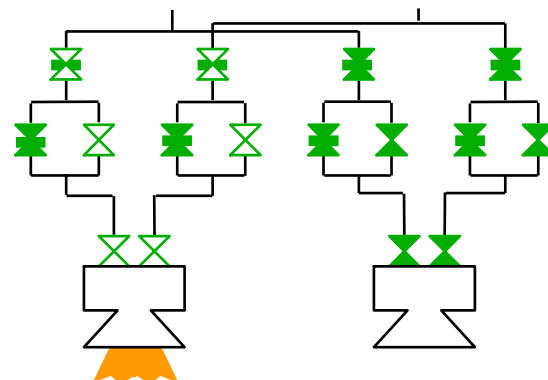
ate to thrusting, and the deductive controller . . . .

Oxidizer tank      Fuel tank

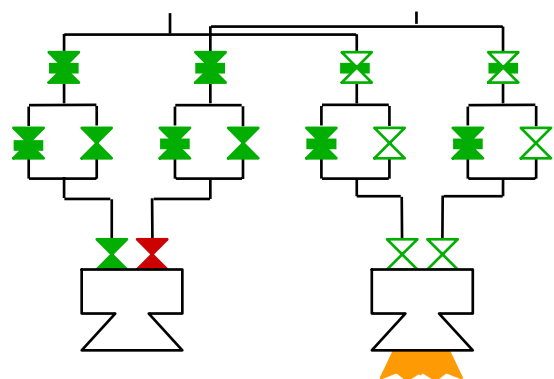


Deduces that thrust is off, and engine is healthy

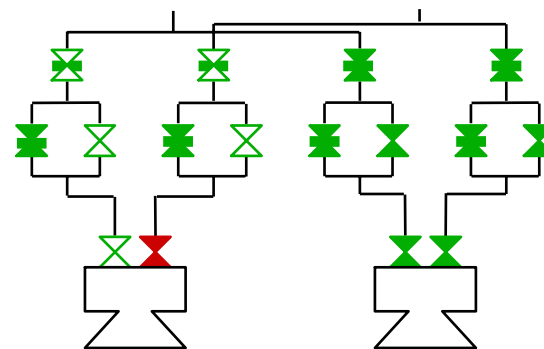
Plans actions to open six valves



Deduces that a valve failed - stuck closed



Determines that valves on the backup engine will achieve thrust, and plans needed actions.



# Titan Model-based Executive

(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) )

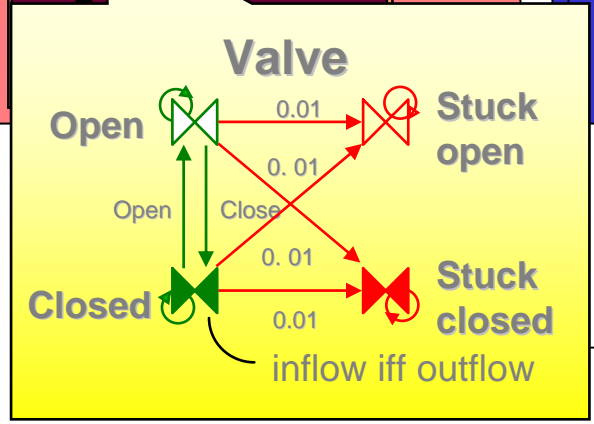
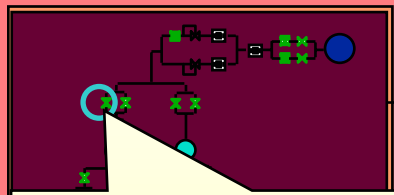
(EngineA = Firing)))

(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

