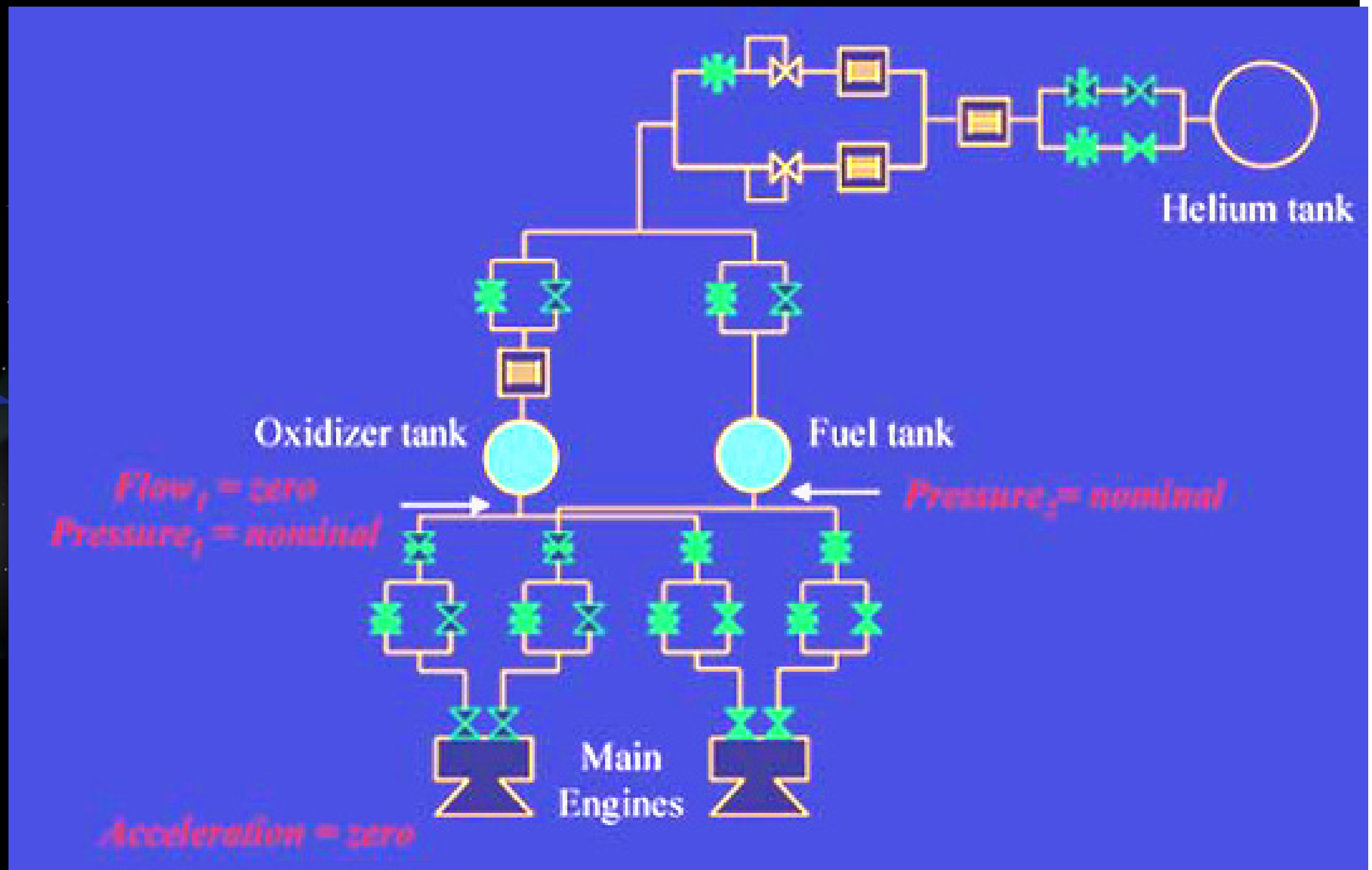
Model-based Programming of Cooperating Explorers

Brian C. Williams
CSAIL
Dept. Aeronautics and Astronautics
Massachusetts Institute of Technology

Programming Long-lived Embedded Systems



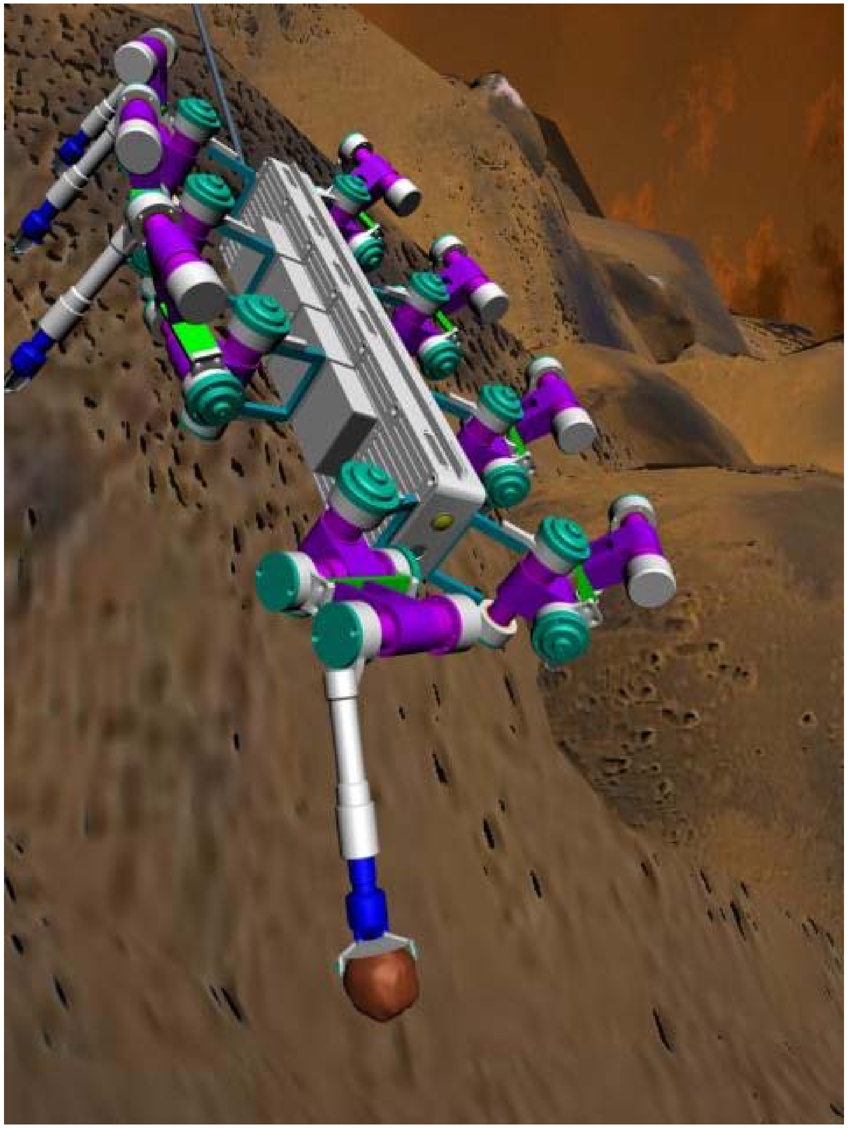
Large collections of devices must work in concert to achieve goals

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

Coordination Recapitulated At The Level of Cooperating Explorers

(Courtesy of Jonathan How. Used with permission.)

Coordination Issues Increase For Dexterous Explorers



(Courtesy of Frank Kirchner. Used with permission.)

Outline

- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy

Approach

Elevate programming and operation to system-level coaching.

➔ Model-based Programming

– State Aware: Coordinates behavior at the level of intended state.

➔ Model-based Execution

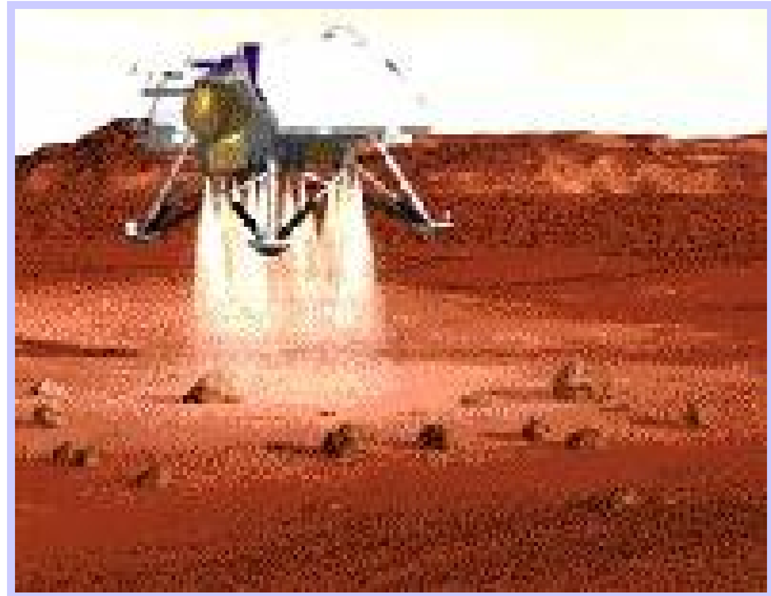
– Fault Aware: Uses models to achieve intended behavior under normal and faulty conditions.

Why Model-based Programming?

Polar Lander Leading Diagnosis:

- Legs deployed during descent.
- Noise spike on leg sensors latched by software monitors.
- Laser altimeter registers 40m.
- Begins polling leg monitors to determine touch down.
- Read latched noise spike as touchdown.
- Engine shutdown at ~40m.

Programmers often make commonsense mistakes when reasoning about hidden state.



Objective: Support programmers with embedded languages that avoid these mistakes, by reasoning about hidden state automatically.

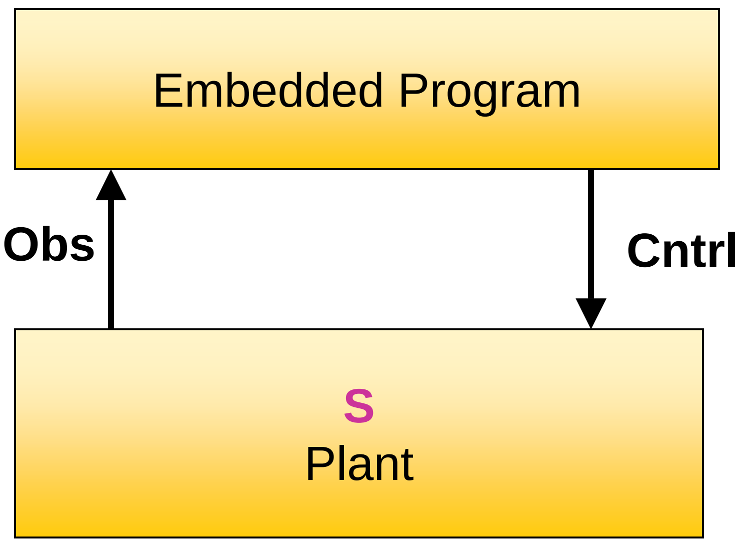
Reactive Model-based Programming Language (RMPL)

Model-based Programs Interact Directly with State

Embedded programs interact with plant sensors and actuators:

• Read sensors

• Set actuators

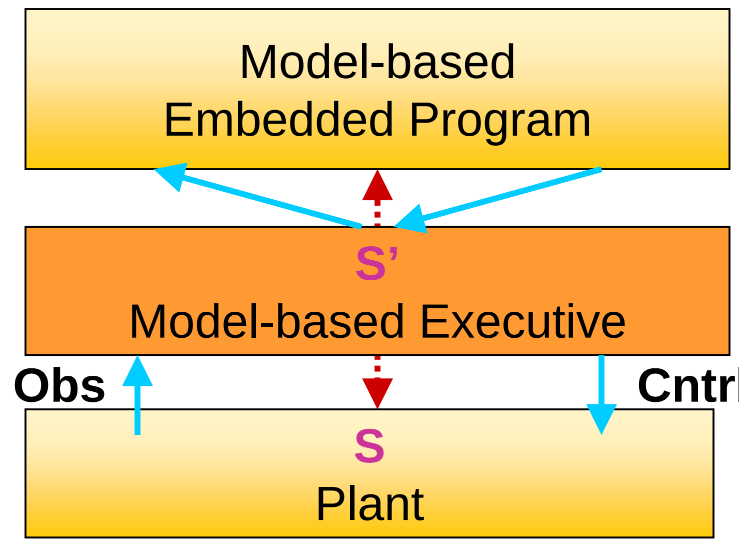


Programmer must map between state and sensors/actuators.

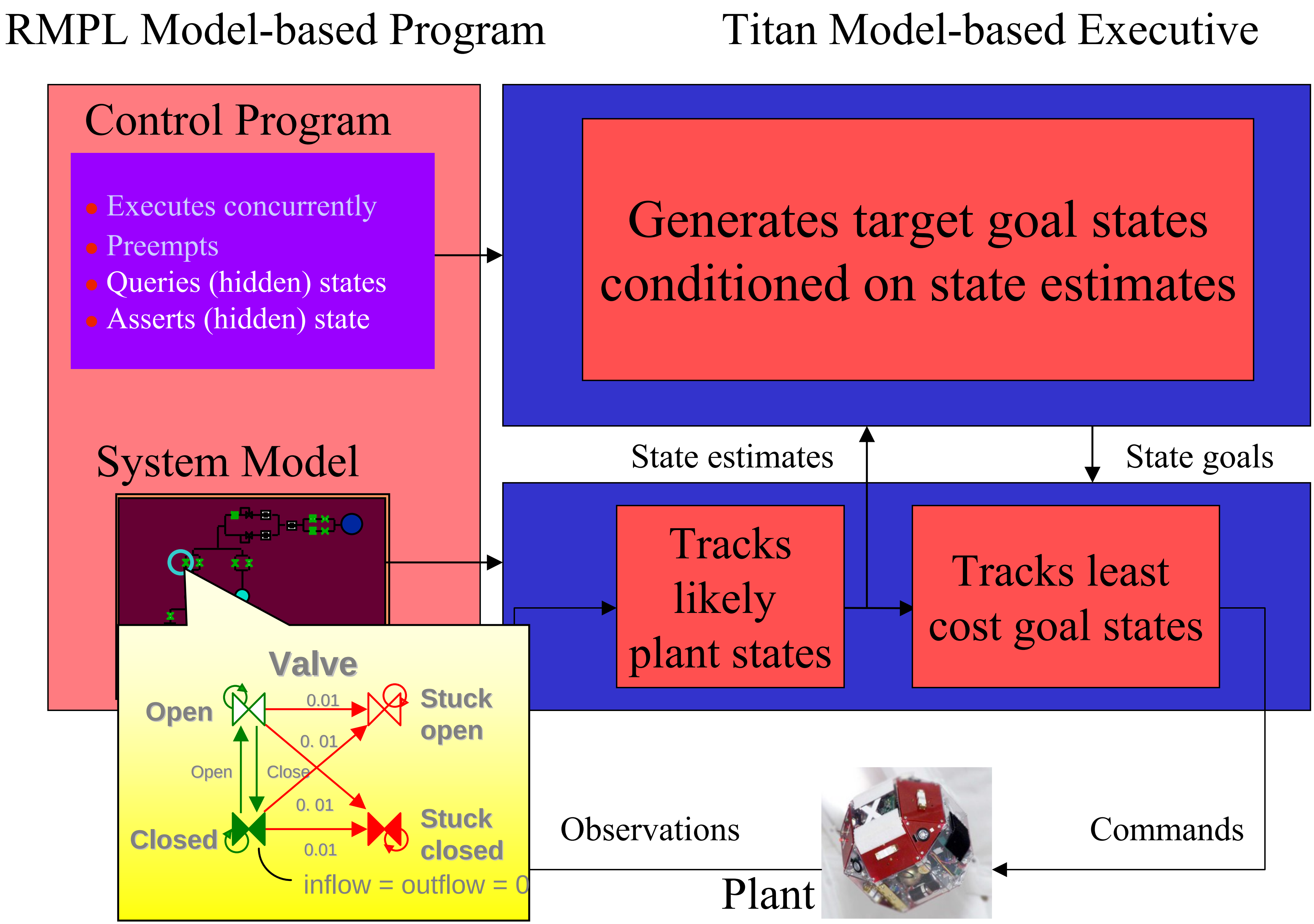
Model-based programs interact with plant state:

• Read state

• Write state



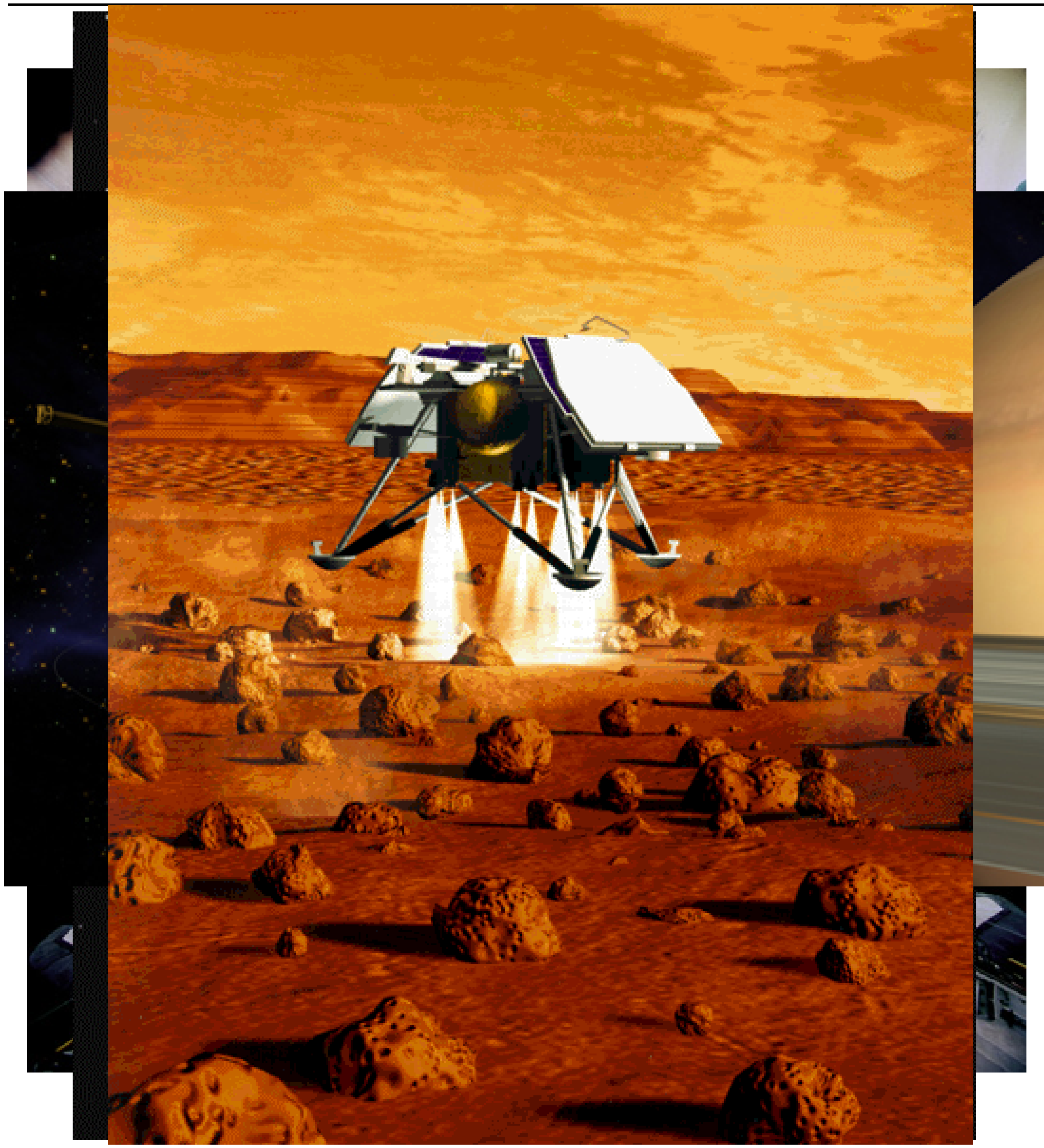
Model-based executive maps between state and sensors/actuators.


RMPL Model-based Program
Titan Model-based Executive
Control Program
Executes concurrently
Preempts
Queries (hidden) states
Asserts (hidden) state
Generates target goal states conditioned on state estimates
System Model
State estimates
State goals
Tracks likely plant states
Tracks least cost goal states
Valve
Open
Stuck open
Open
Close
Closed
Stuck closed
0.01
0. 01
0. 01
0.01
inflow = outflow = 0
Observations
Commands
Plant

Outline

- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy

Motivation



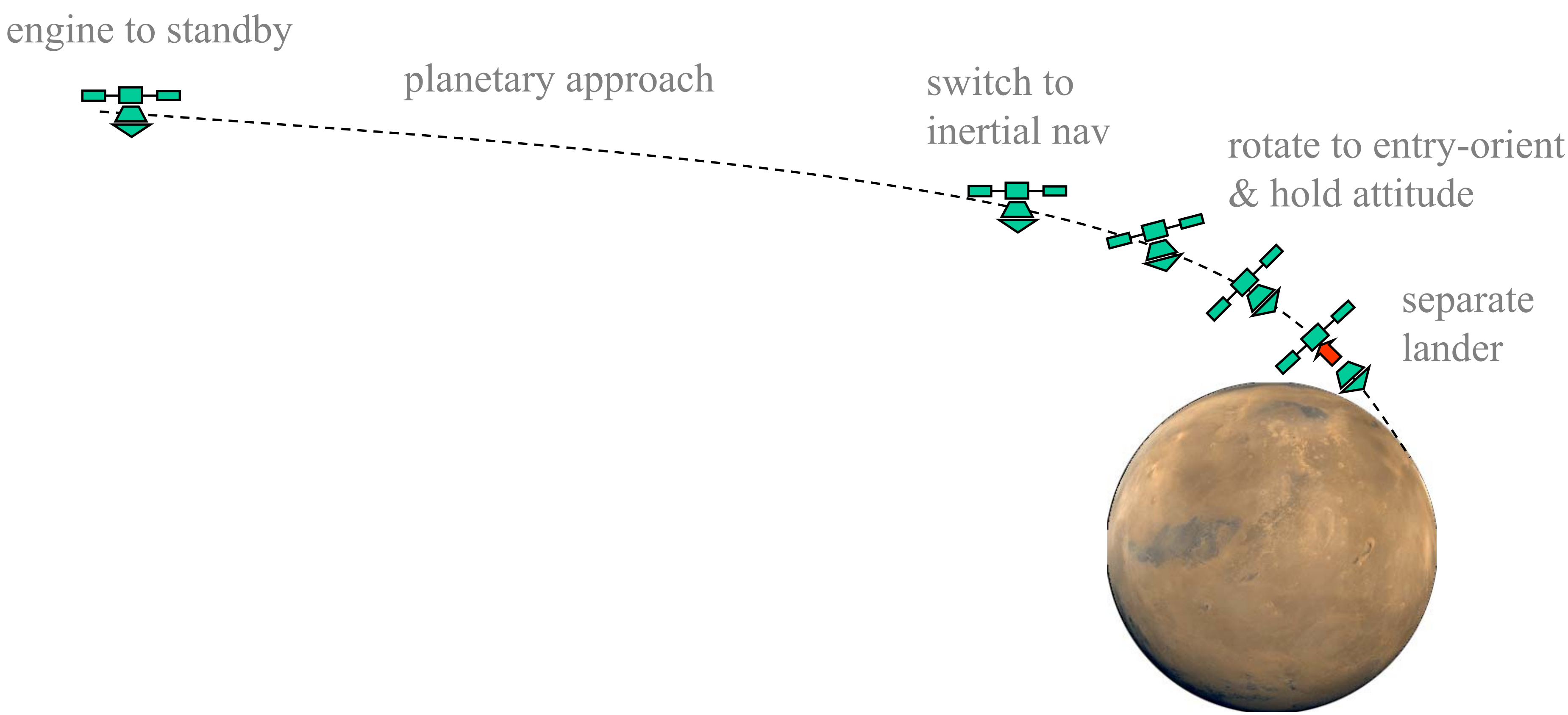
Mission-critical sequences:

- Launch & deployment ☆
- Planetary fly-by ☆
- Orbital insertion ☆
- Entry, descent & landing ☆

images courtesy of NASA

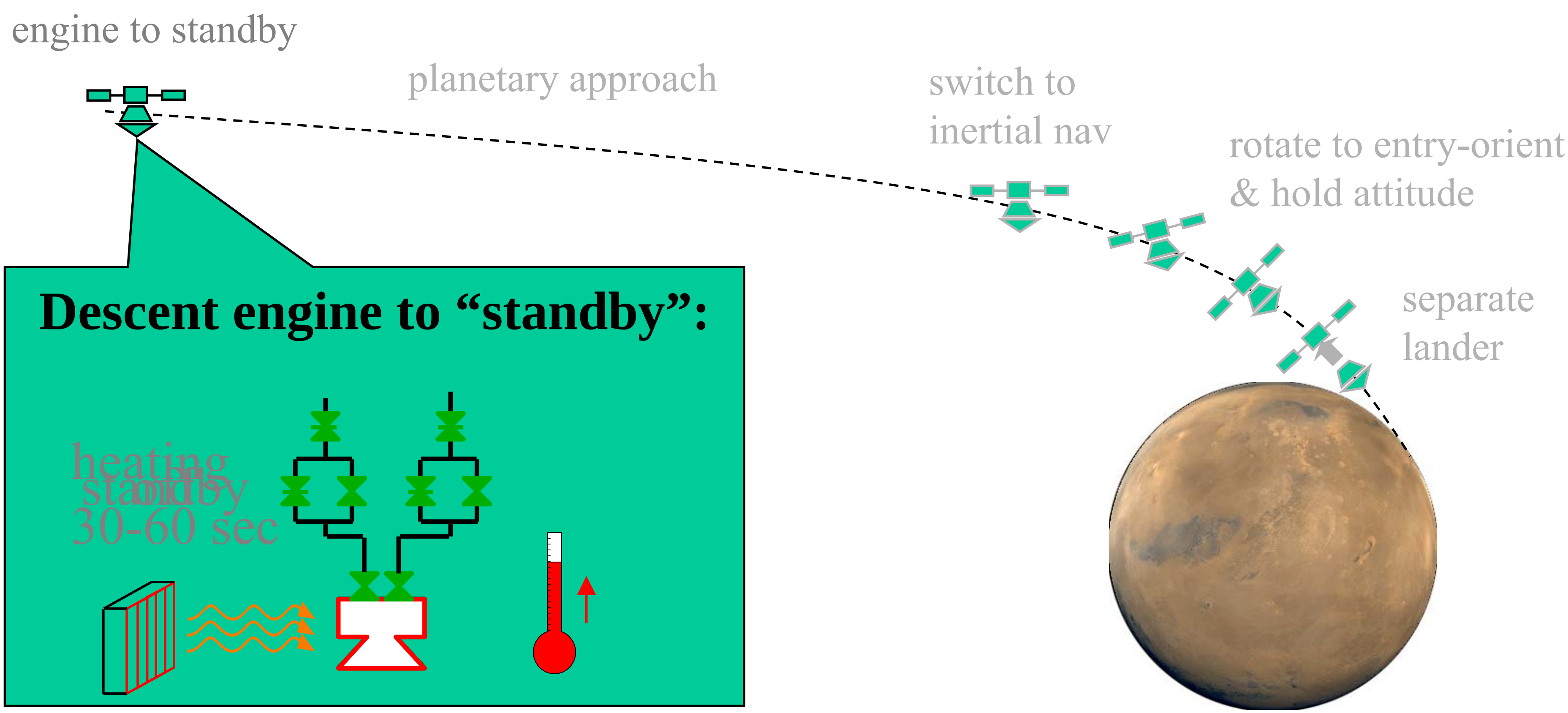
(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example

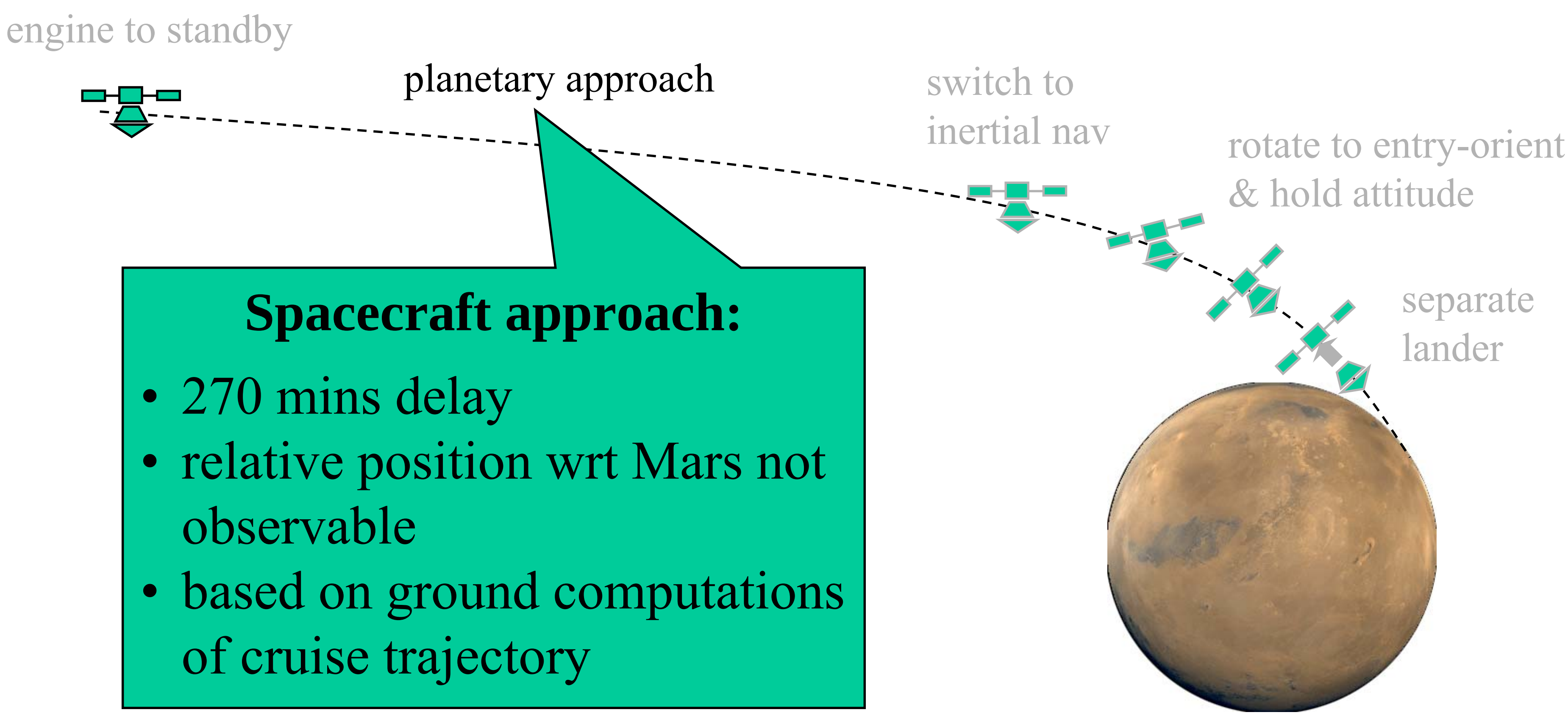


(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example



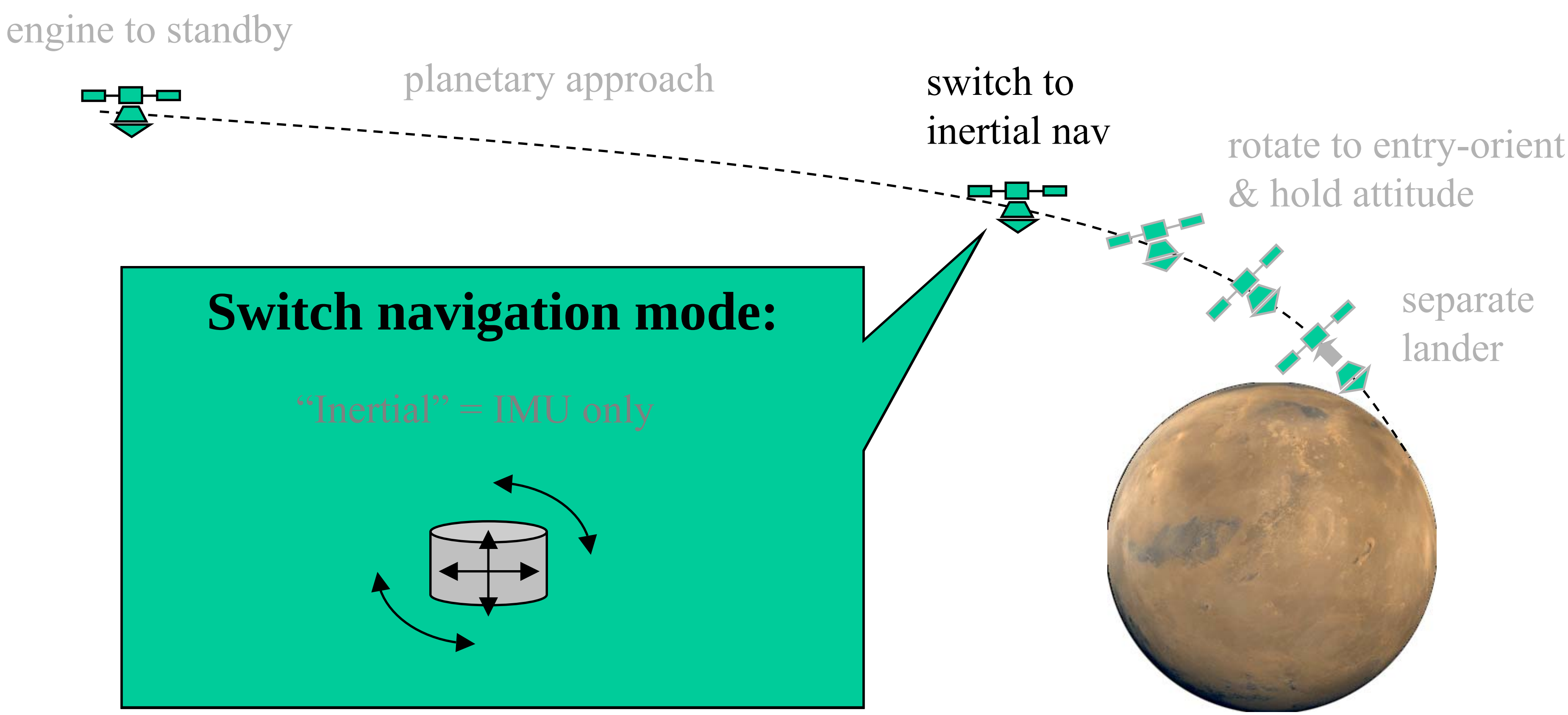
Mars Entry Example



Spacecraft approach:

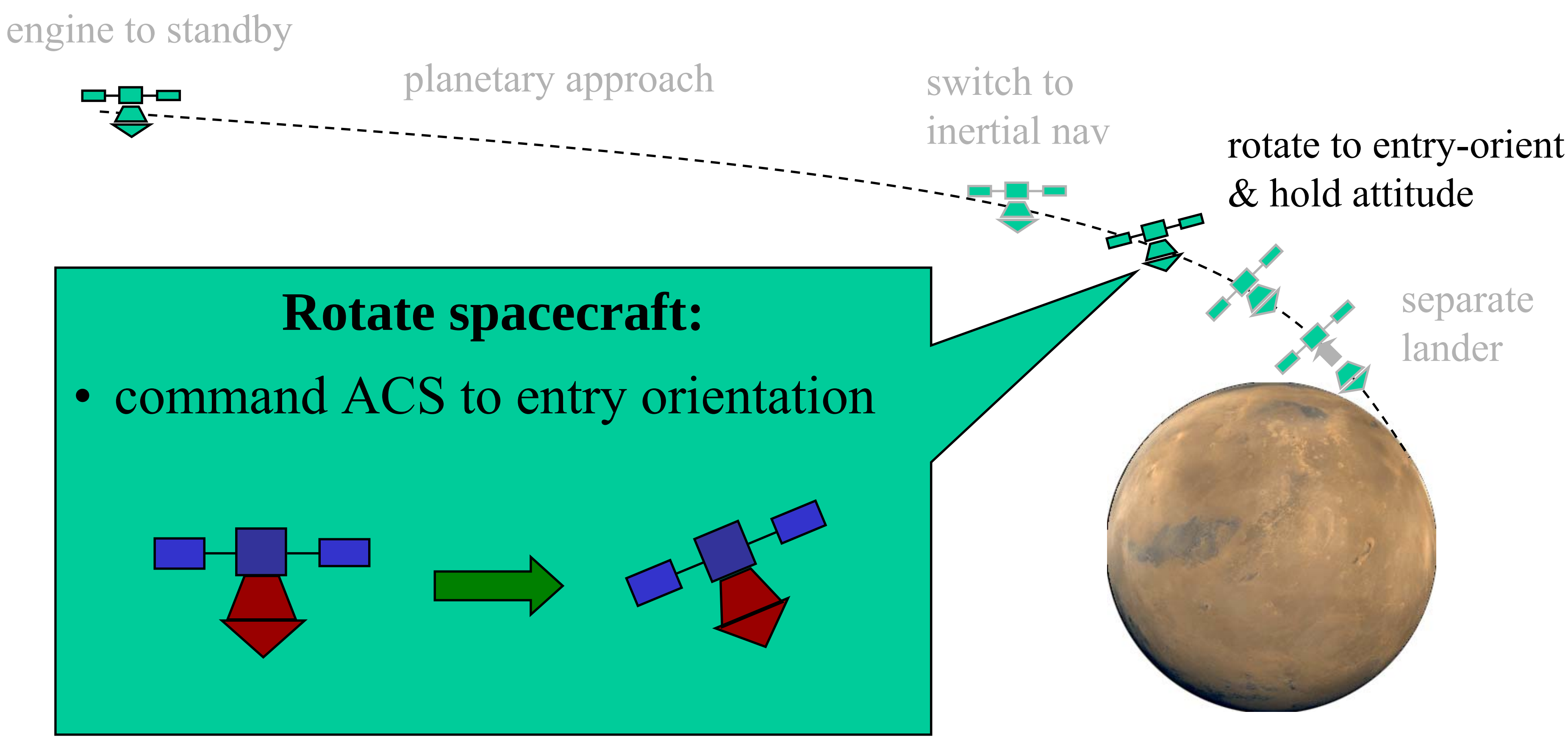
- 270 mins delay
- relative position wrt Mars not observable
- based on ground computations of cruise trajectory

Mars Entry Example



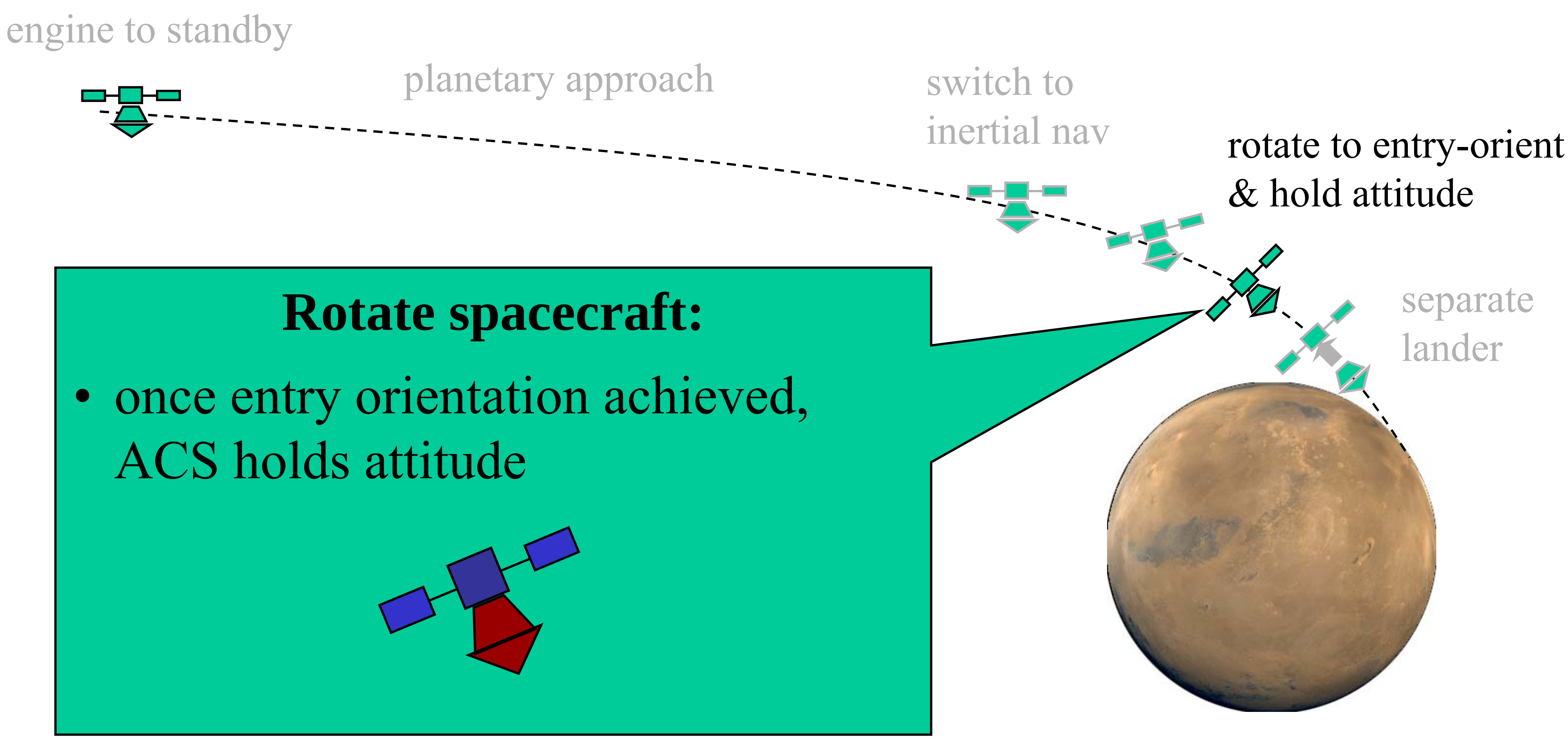
(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example



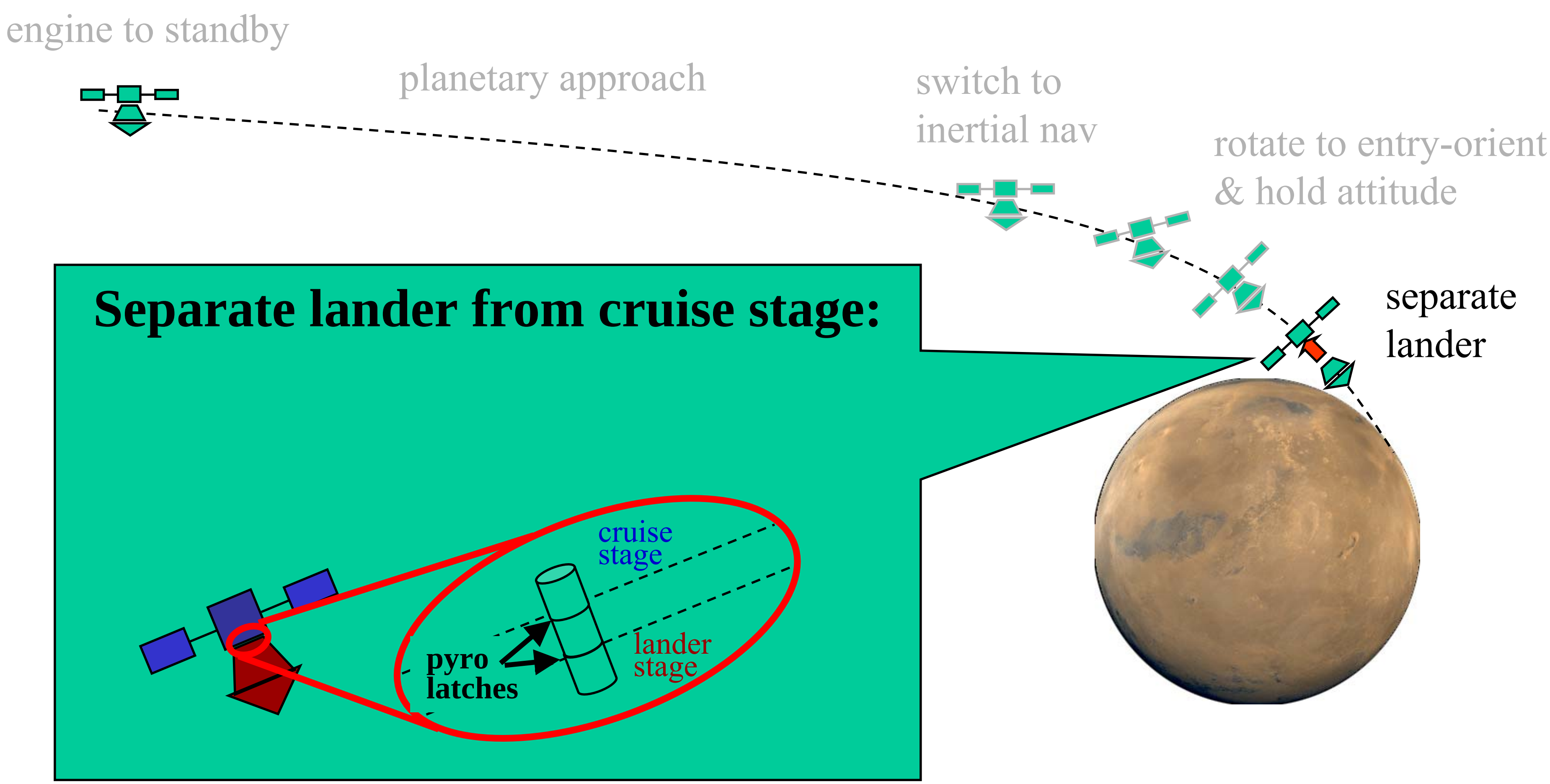
(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example



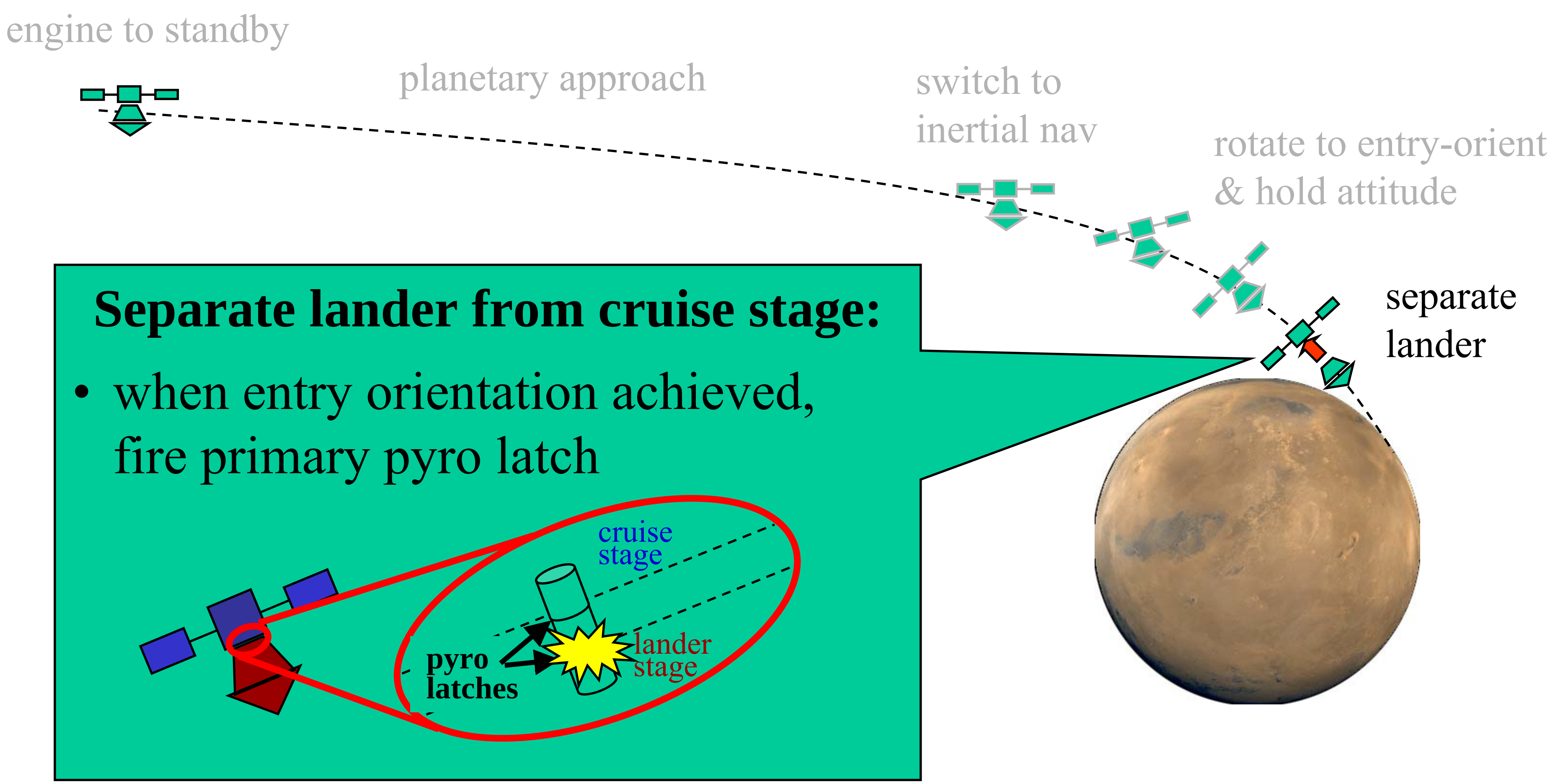
(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example



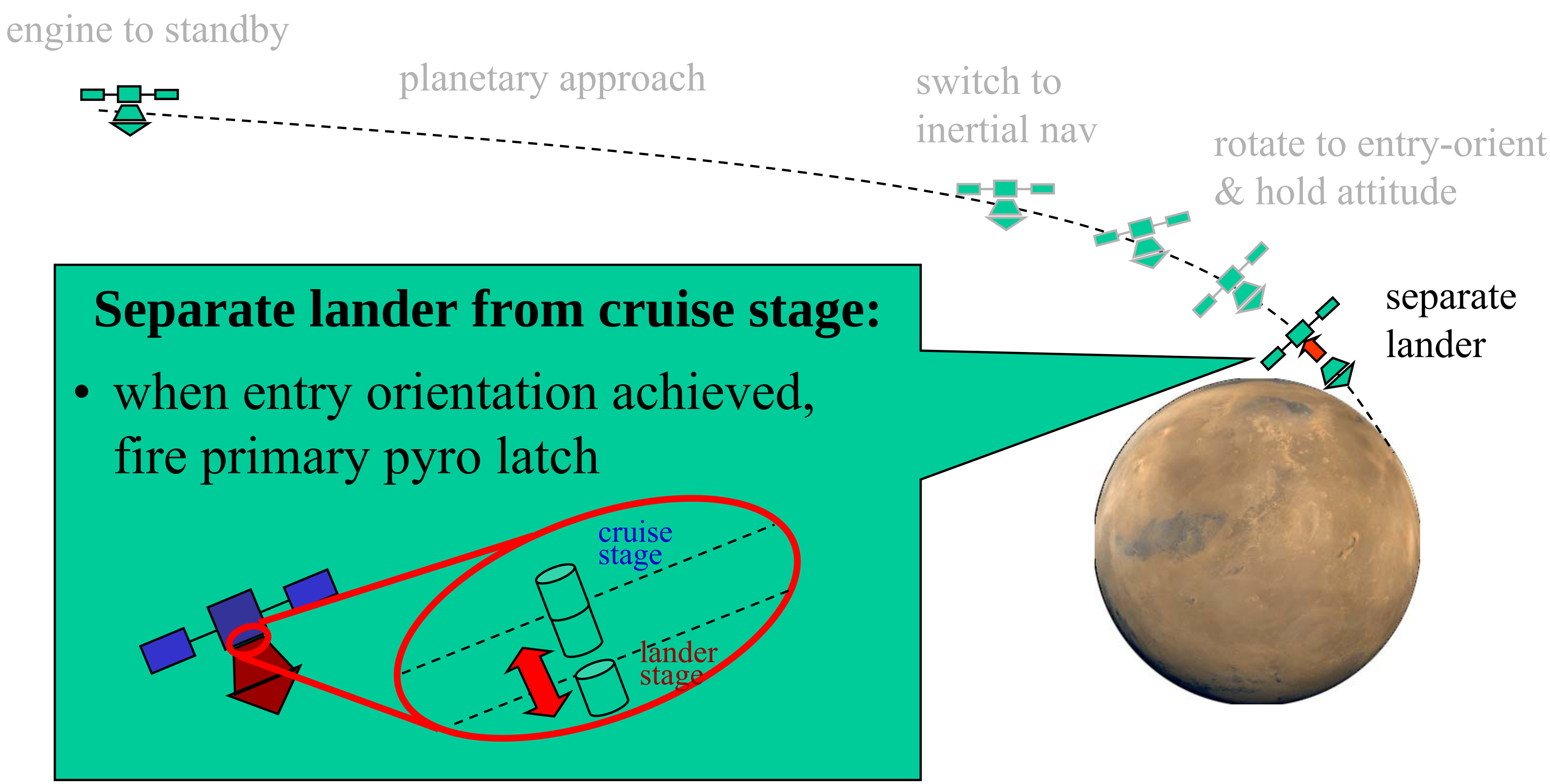
(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example

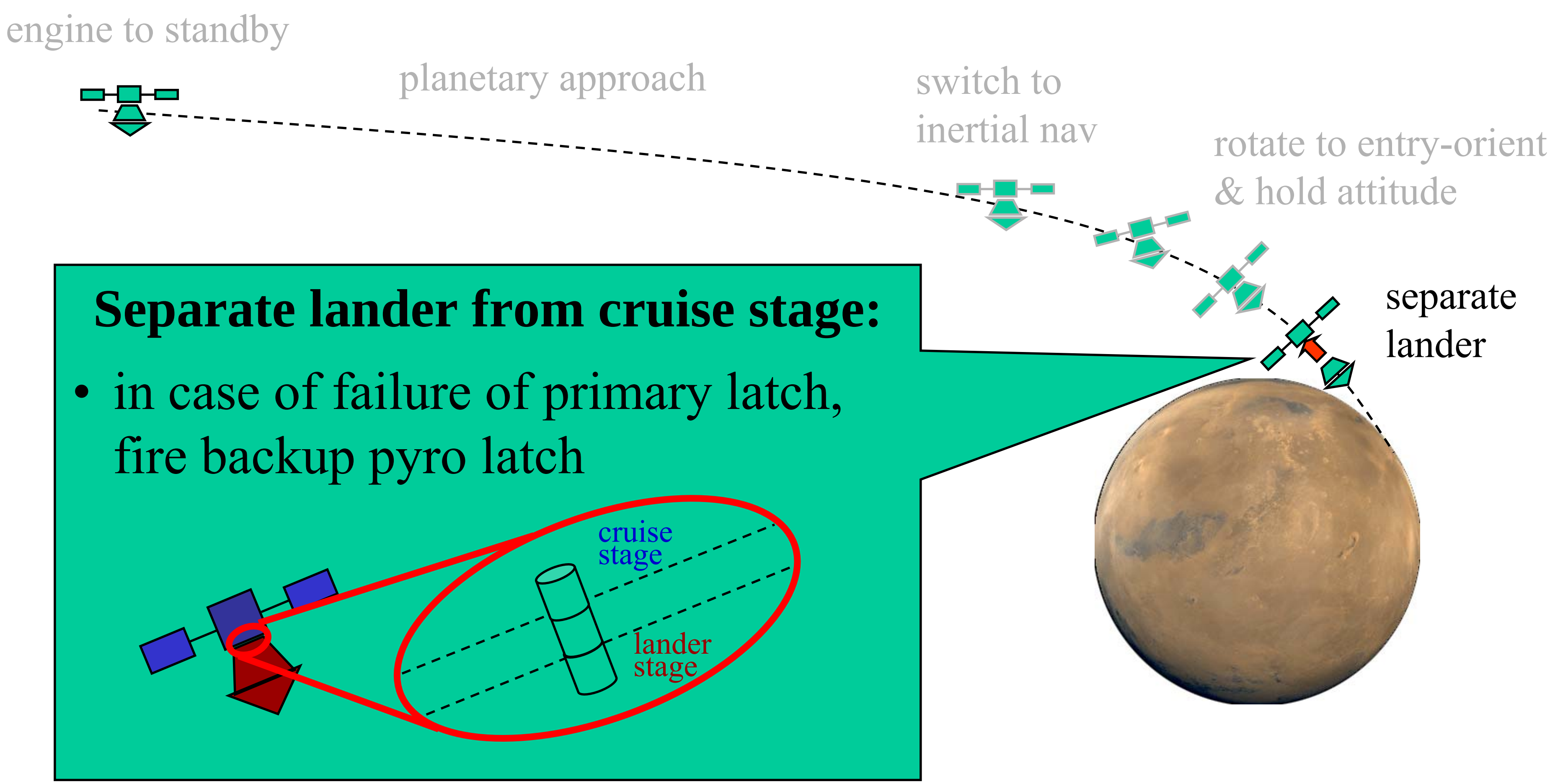


(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example

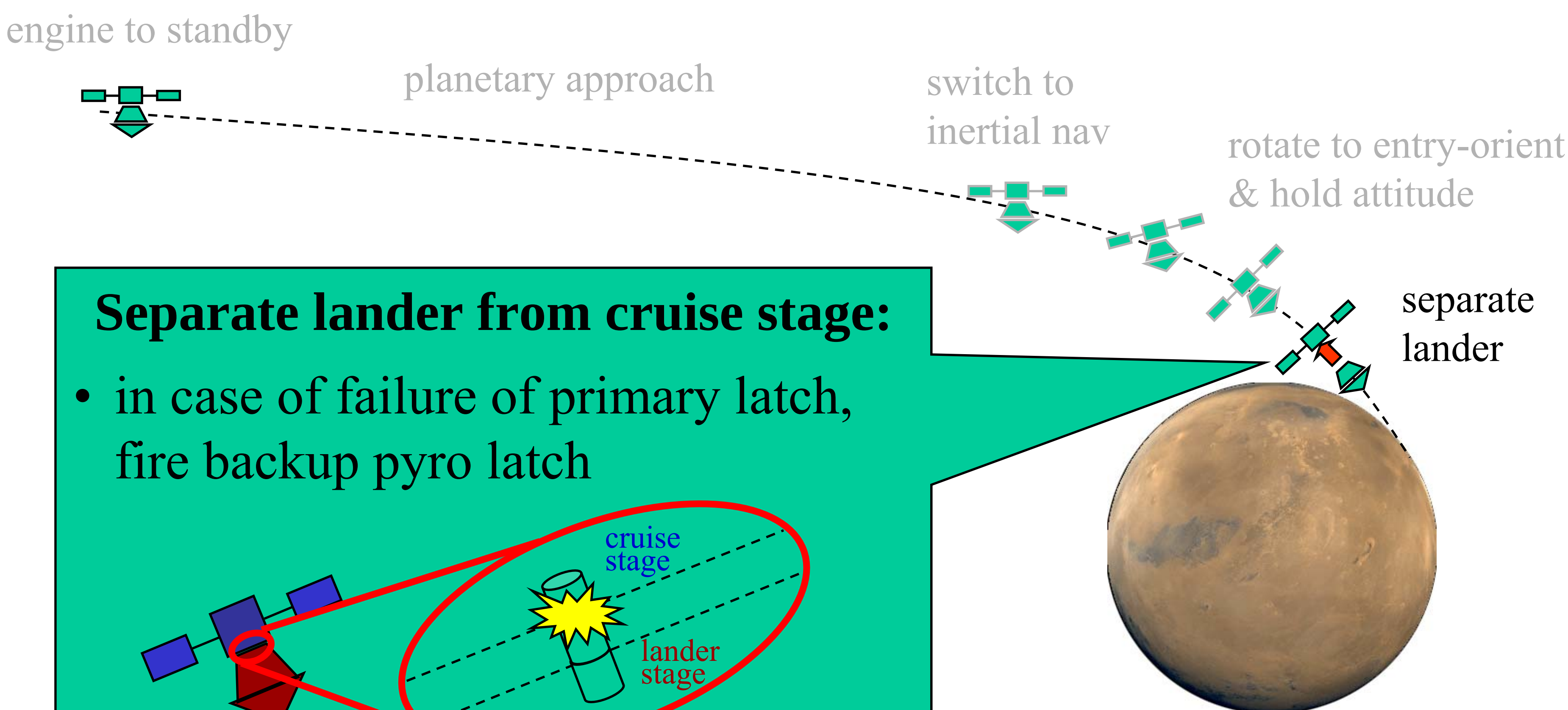


Mars Entry Example



(Courtesy of Mitch Ingham. Used with permission.)

Mars Entry Example

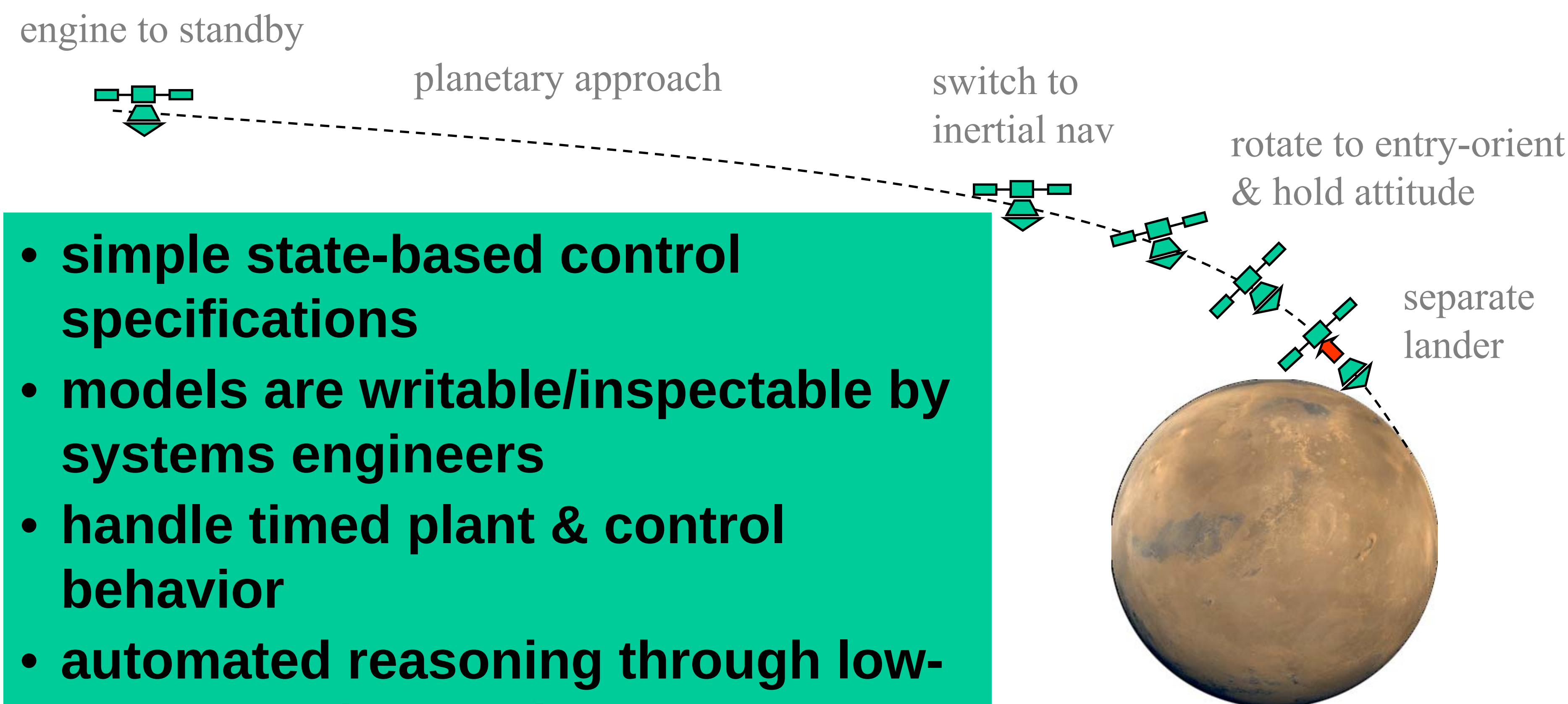


Separate lander from cruise stage:

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

(Courtesy of Mitch Ingham. Used with permission.)

What is Required to Program at This Level?

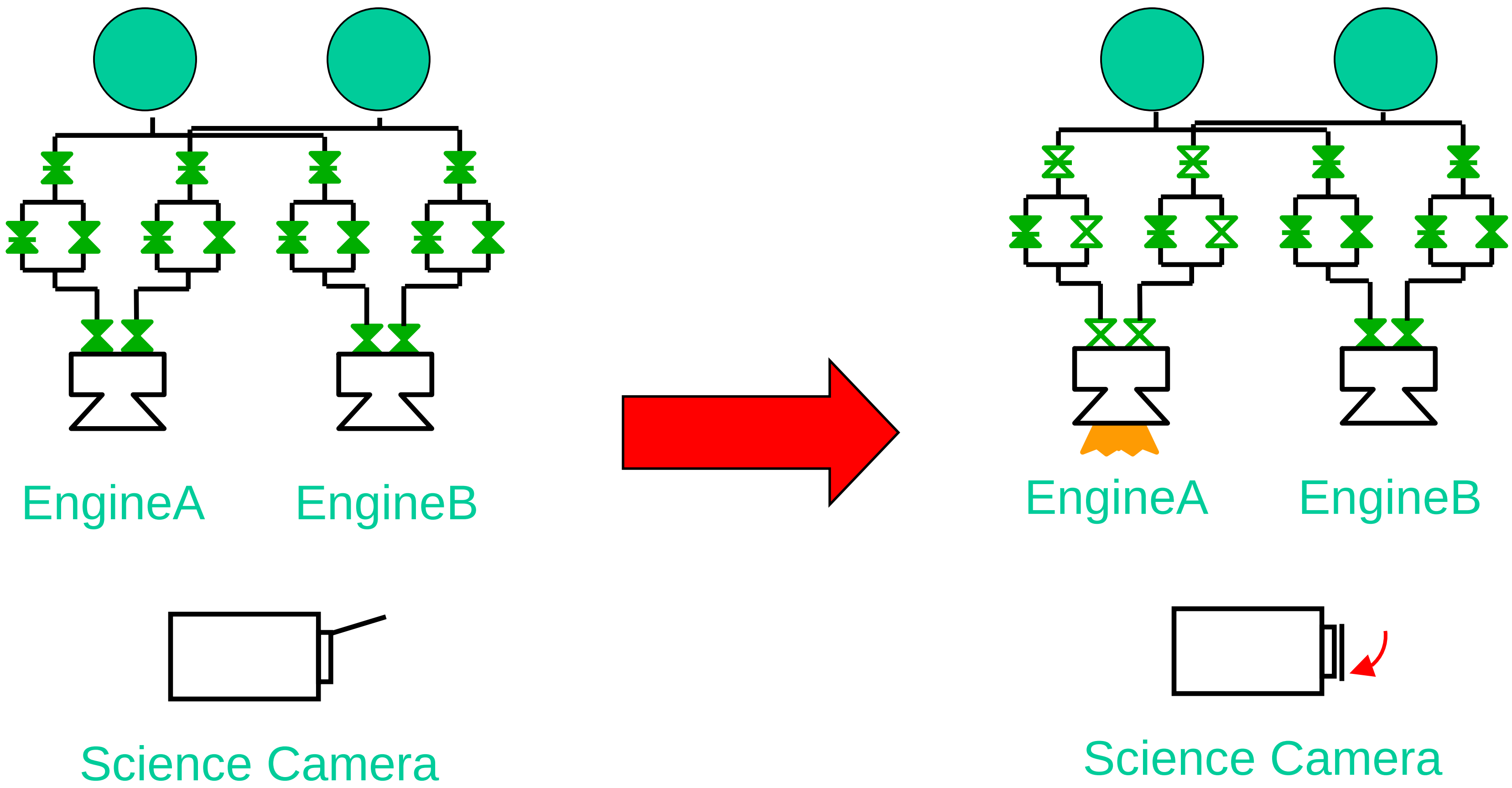


- **simple state-based control specifications**
- **models are writable/inspectable by systems engineers**
- **handle timed plant & control behavior**
- **automated reasoning through low-level plant interactions**
- **fault-aware (in-the-loop recoveries)**

(Courtesy of Mitch Ingham. Used with permission.)

Descent Example

Turn camera off and engine on



Model-based Program

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))))
```

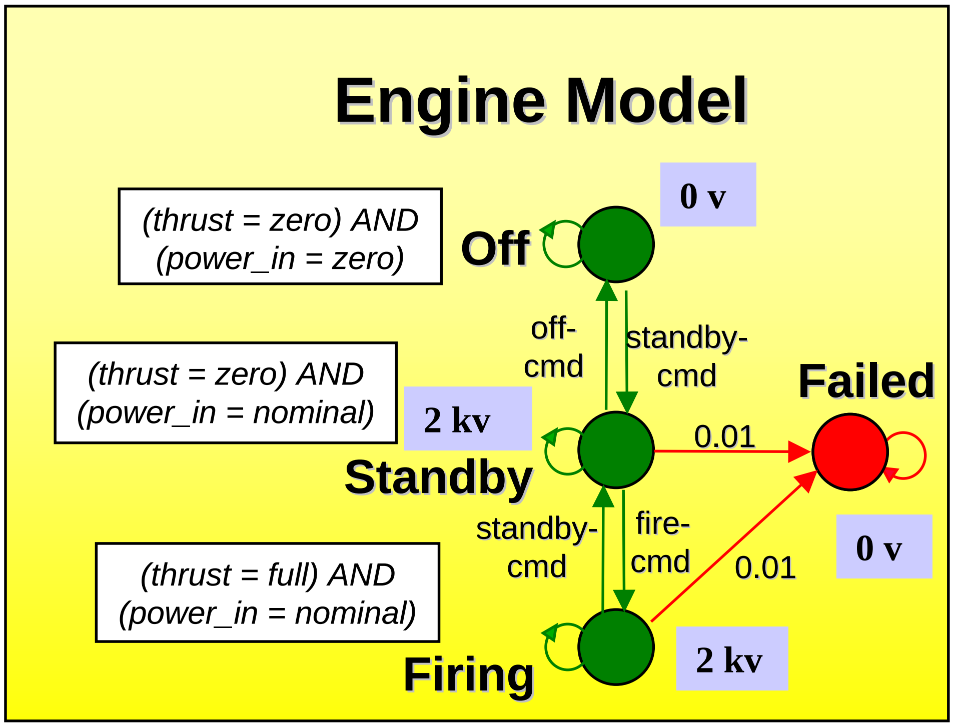
Plant Model

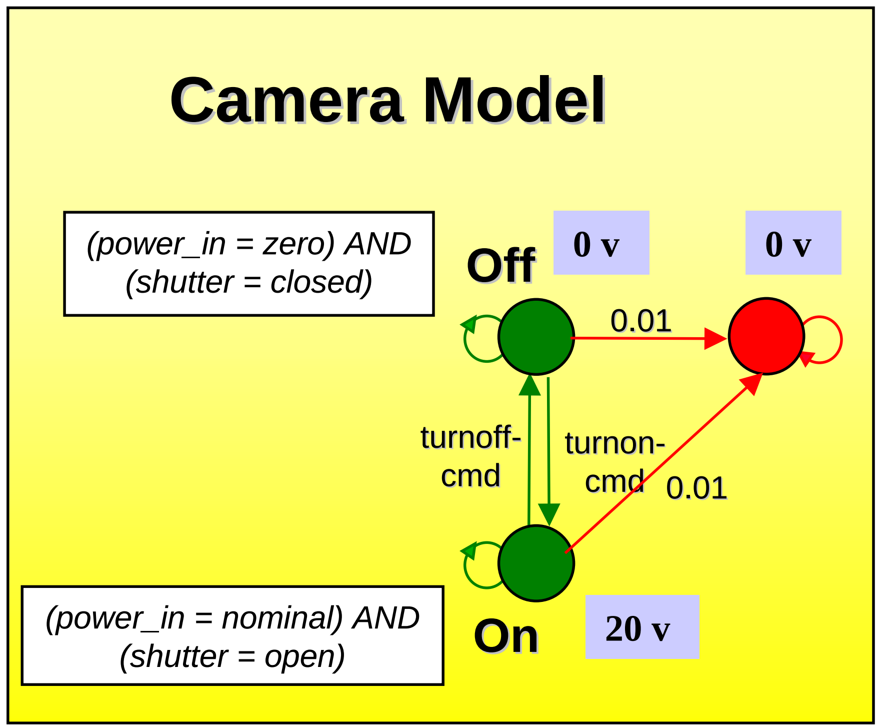
component modes…

described by finite domain constraints on variables…

deterministic and probabilistic transitions

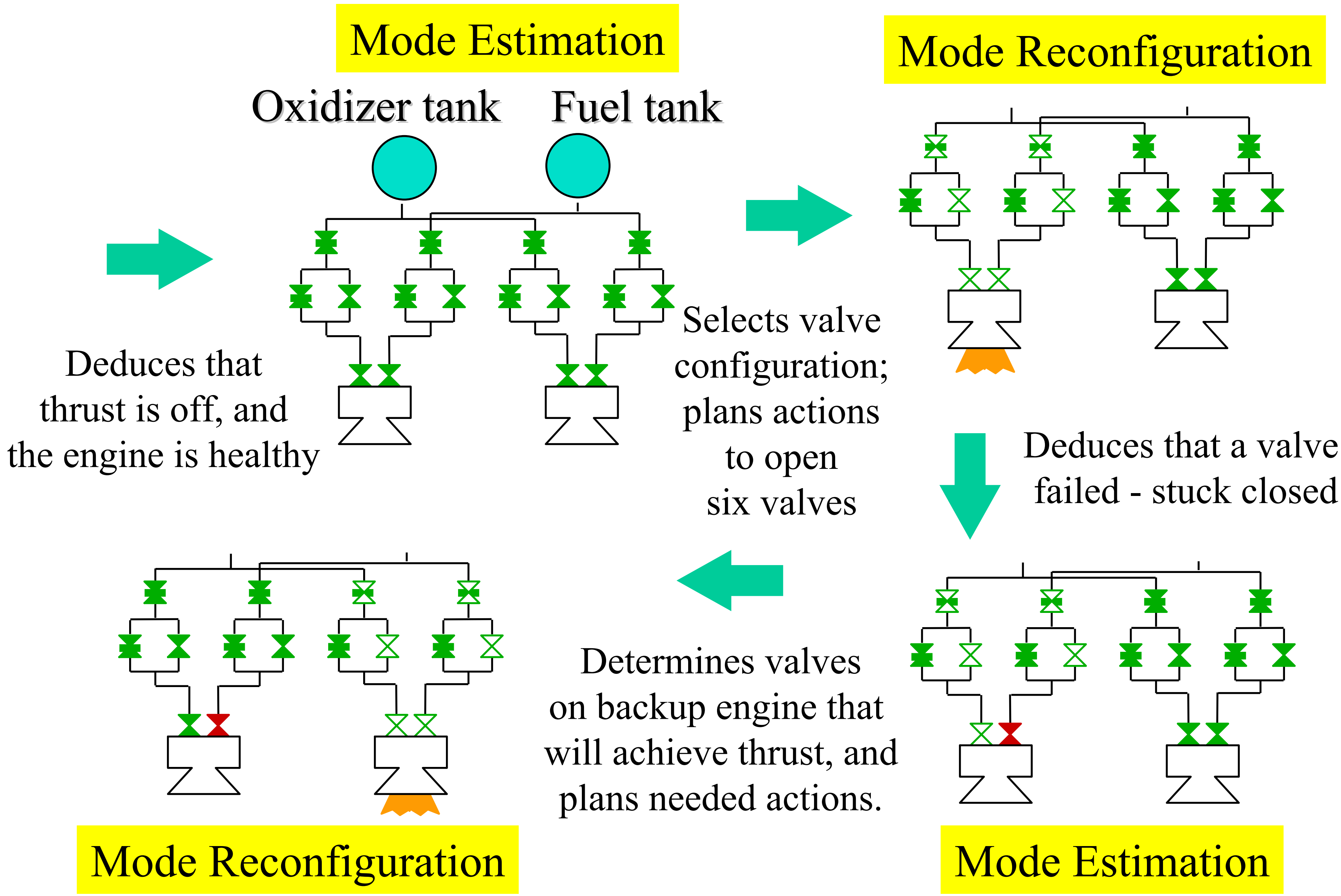
cost/reward





one per component … operating concurrently

Example: The model-based program sets engine = thrusting, and the deductive controller . . . .



Outline

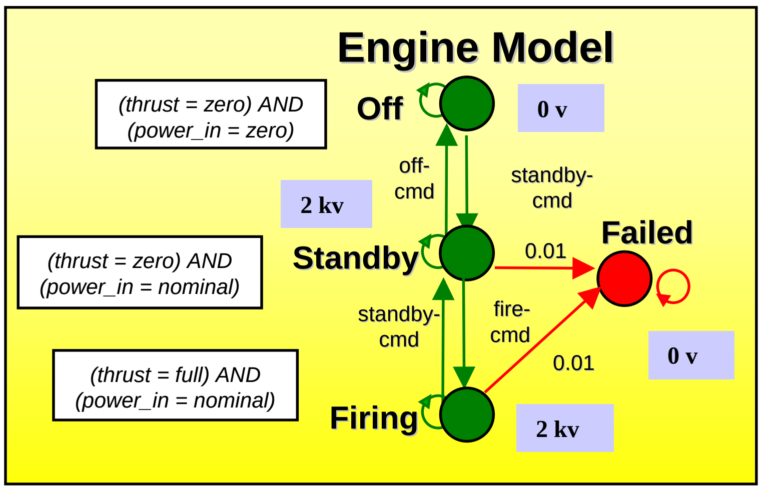
- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy

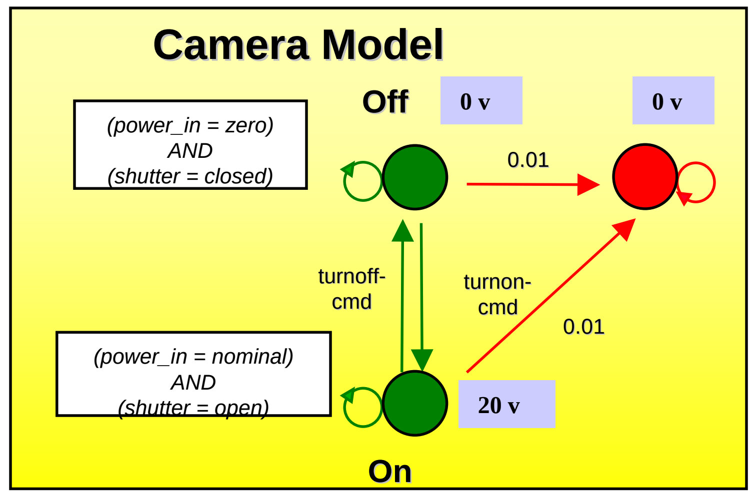
Modeling Plant Dynamics using Probabilistic Concurrent, Constraint Automata (PCCA)

Compact Encoding:

– Concurrent probabilistic transitions

– State constraints between variables



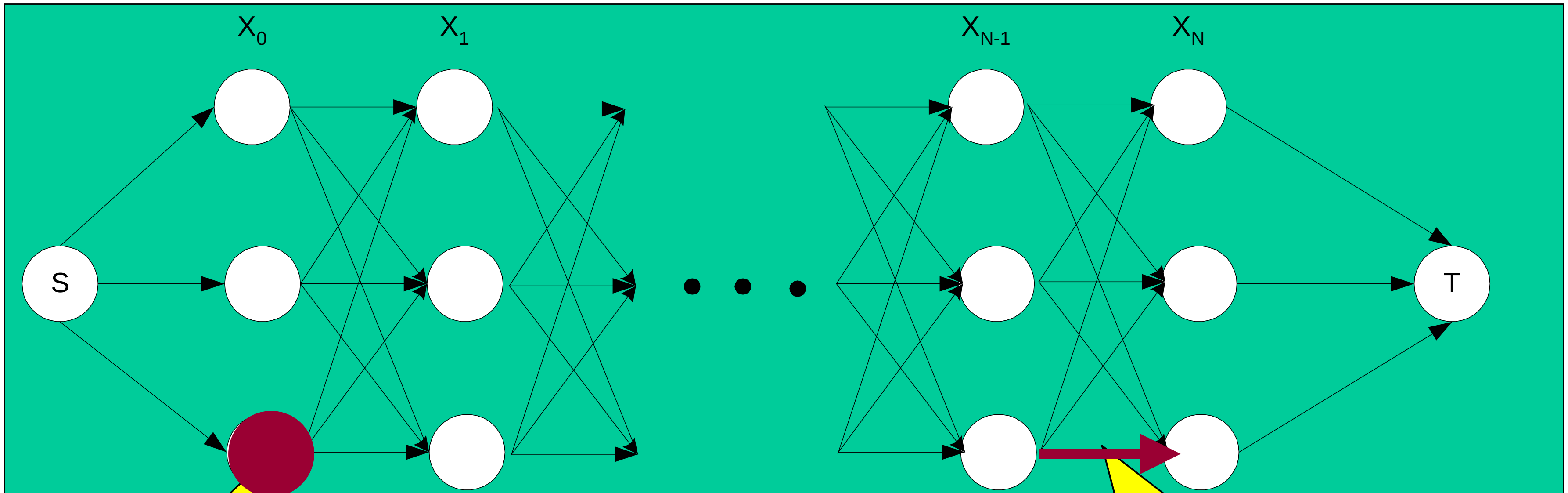


Typical Example (DS1 spacecraft):

– 80 Automata, 5 modes on average

– 3000 propositional variables, 12,000 propositional clauses

The Plant's Behavior



- Assigns a value to each variable (e.g.,3,000 vars).
- Consistent with all state constraints (e.g., 12,000).

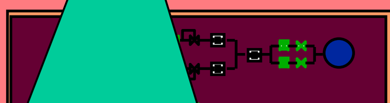
- A set of concurrent transitions, one per automata (e.g., 80).
- Previous & Next states consistent with source & target of transitions

RMPL Model-based Program

Control Program

- Executes concurrently
- Preempts
- Asserts and queries states
- Chooses based on reward

System Model



```
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))))
```

Titan Model-based Executive

Control Sequencer:

Generates goal states conditioned on state estimates

State estimates

State goals

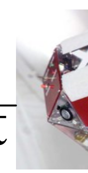
Mode Estimation:

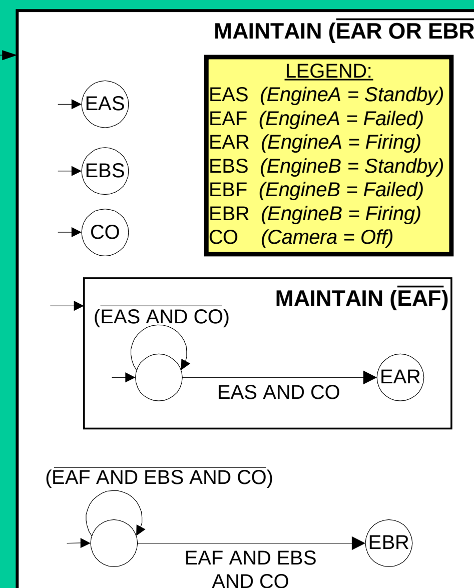
Tracks likely States

Mode

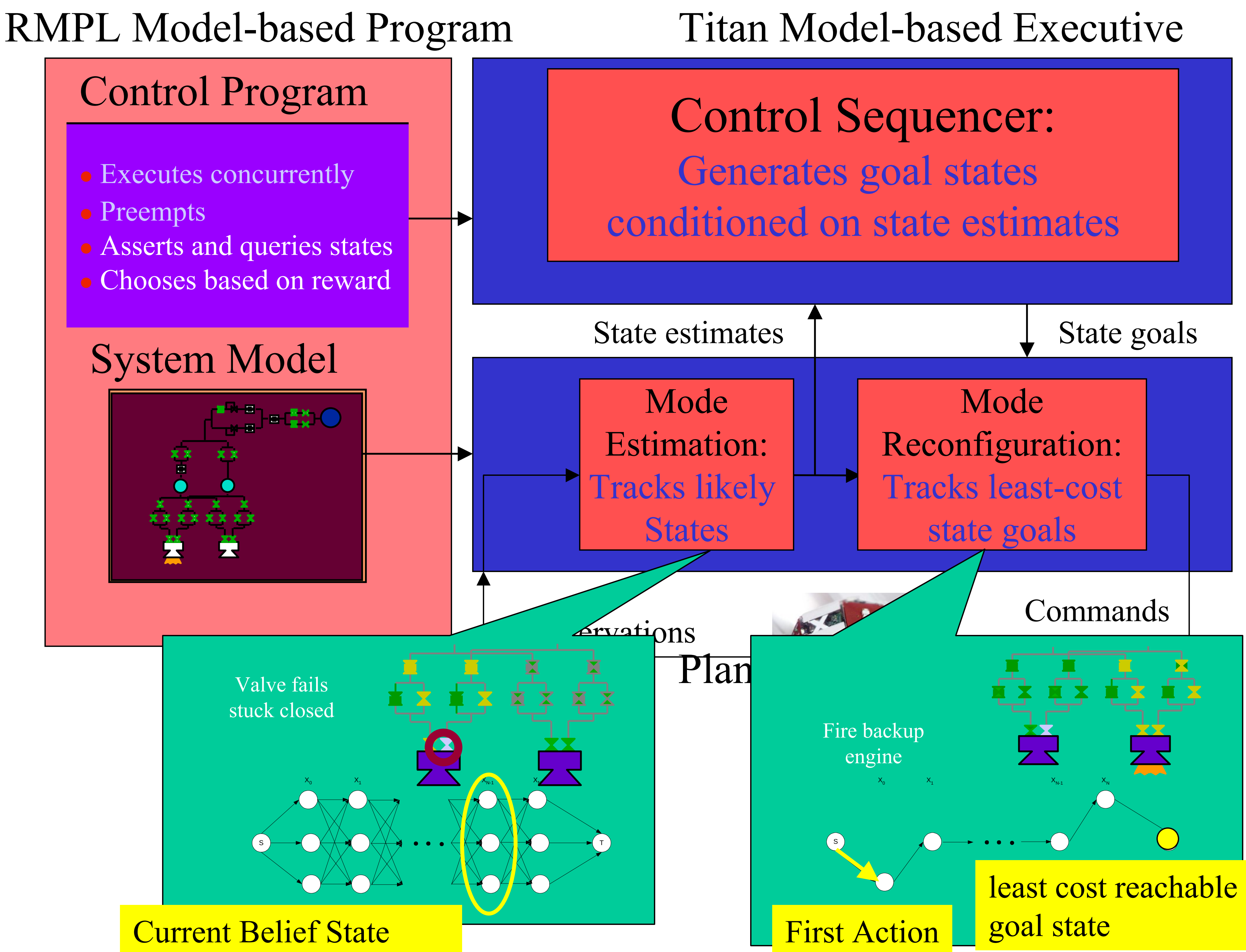
Observations

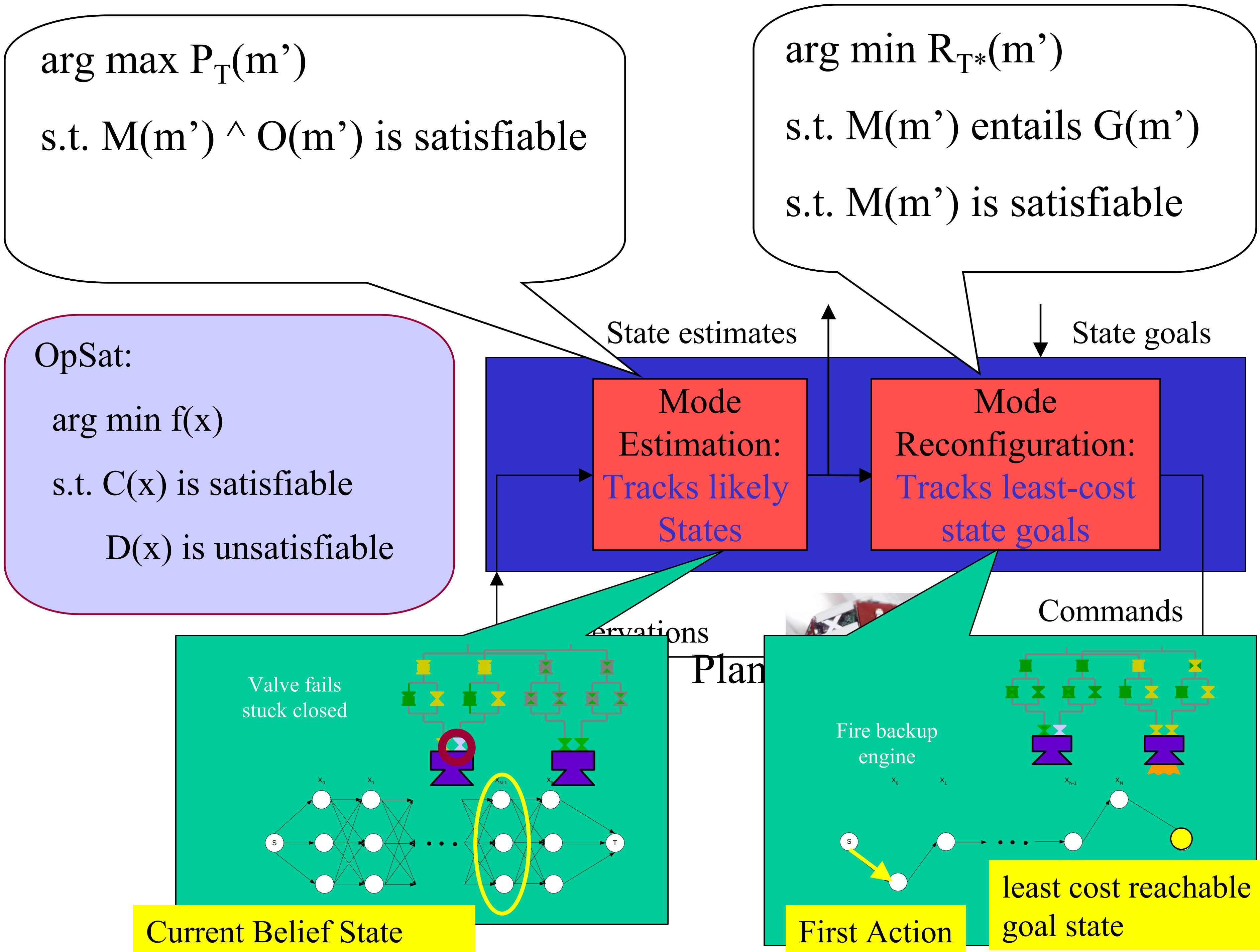
Plant





hierarchical constraint automata on state s


RMPL Model-based Program
Titan Model-based Executive
Control Program
Executes concurrently
Preempts
Asserts and queries states
Chooses based on reward
System Model
Control Sequencer:
Generates goal states
conditioned on state estimates
State estimates
State goals
Mode
Estimation:
Tracks likely
States
Mode
Reconfiguration:
Tracks least-cost
state goals
Commands
Plan
Valve fails
stuck closed
Fire backup
engine
Current Belief State
First Action
least cost reachable
goal state



arg max $P_T(m')$
s.t. $M(m')$ ^ $O(m')$ is satisfiable
arg min $R_{T*}(m')$
s.t. $M(m')$ entails $G(m')$
s.t. $M(m')$ is satisfiable
OpSat:
arg min f(x)
s.t. C(x) is satisfiable
D(x) is unsatisfiable
State estimates
State goals
Mode Estimation: Tracks likely States
Mode Reconfiguration: Tracks least-cost state goals
ervations
Plan
Commands
Valve fails stuck closed
Fire backup engine
Current Belief State
First Action
least cost reachable goal state

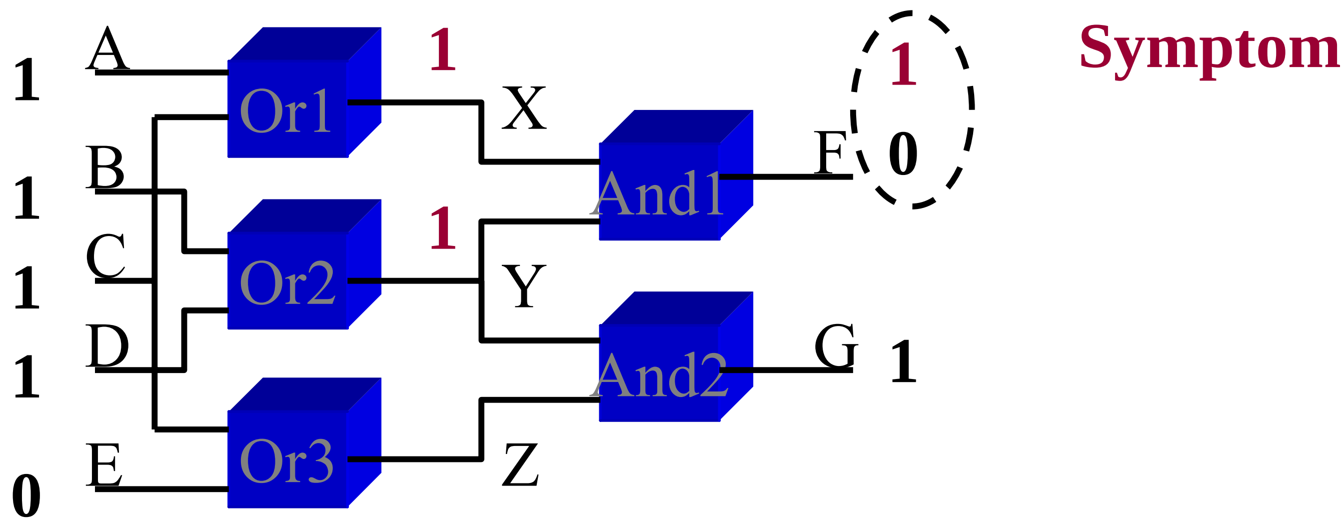
Outline

- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy

Diagnosis Formulation

Consistency-based Diagnosis: Given symptoms, find diagnoses that are consistent with symptoms.

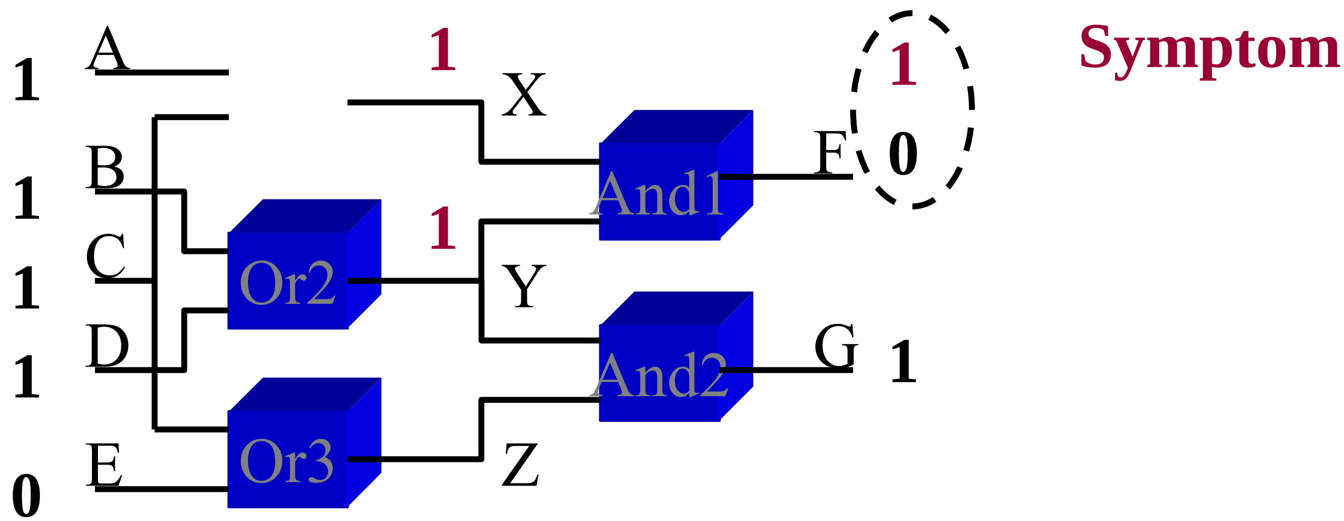
Handle Novel Failures by Suspending Constraints: Make no presumptions about faulty component behavior.



Diagnosis Formulation

Consistency-based Diagnosis: Given symptoms, find diagnoses that are consistent with symptoms.

Handle Novel Failures by Suspending Constraints: Make no presumptions about faulty component behavior.



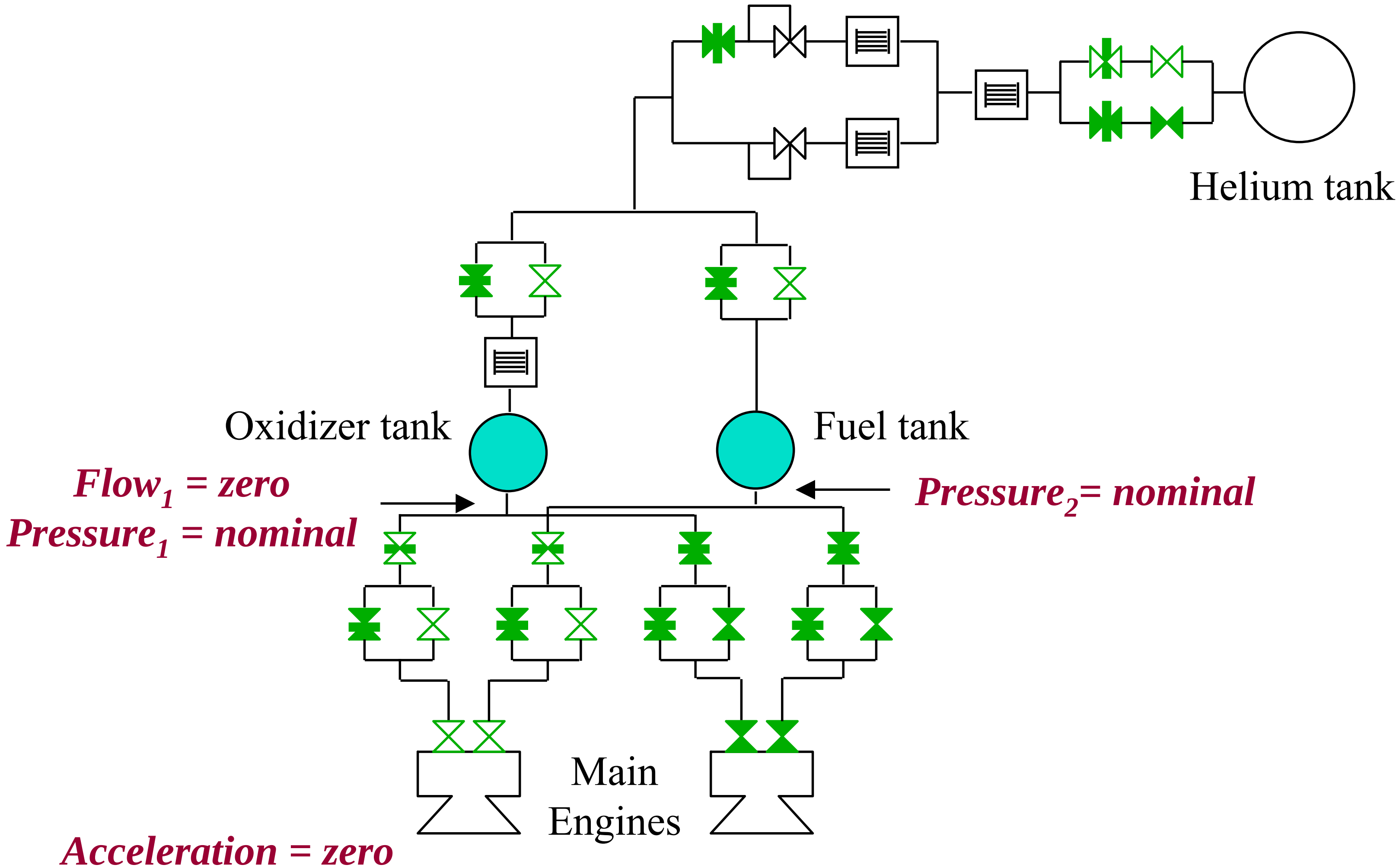
Fast Reasoning Through Conflict

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

- Sherlock Holmes. The Sign of the Four.

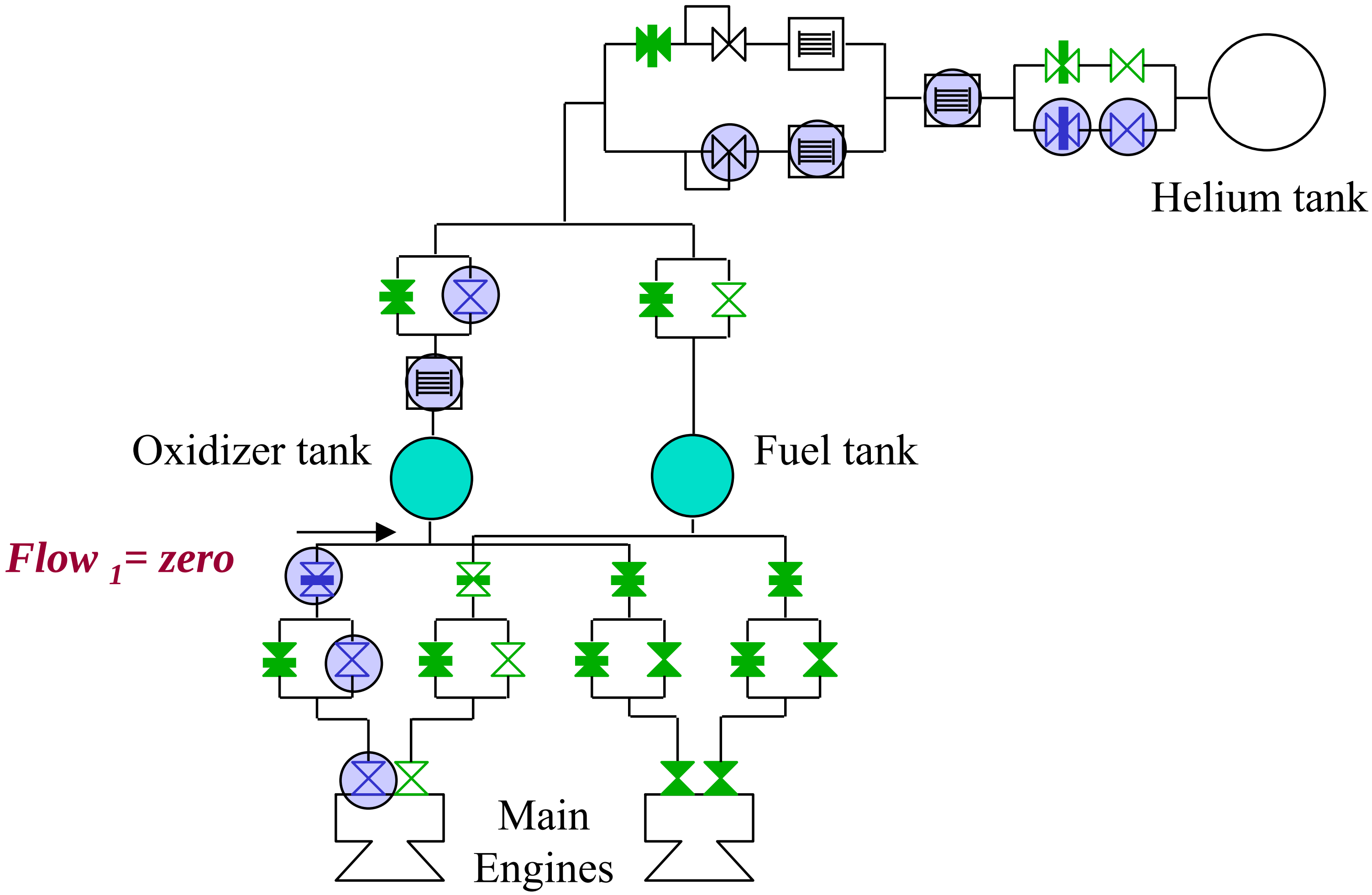
1. Test Hypothesis
2. If inconsistent, learn reason for inconsistency (a Conflict).
3. Use conflicts to leap over similarly infeasible options to next best hypothesis.

Compare Most Likely Hypothesis to Observations



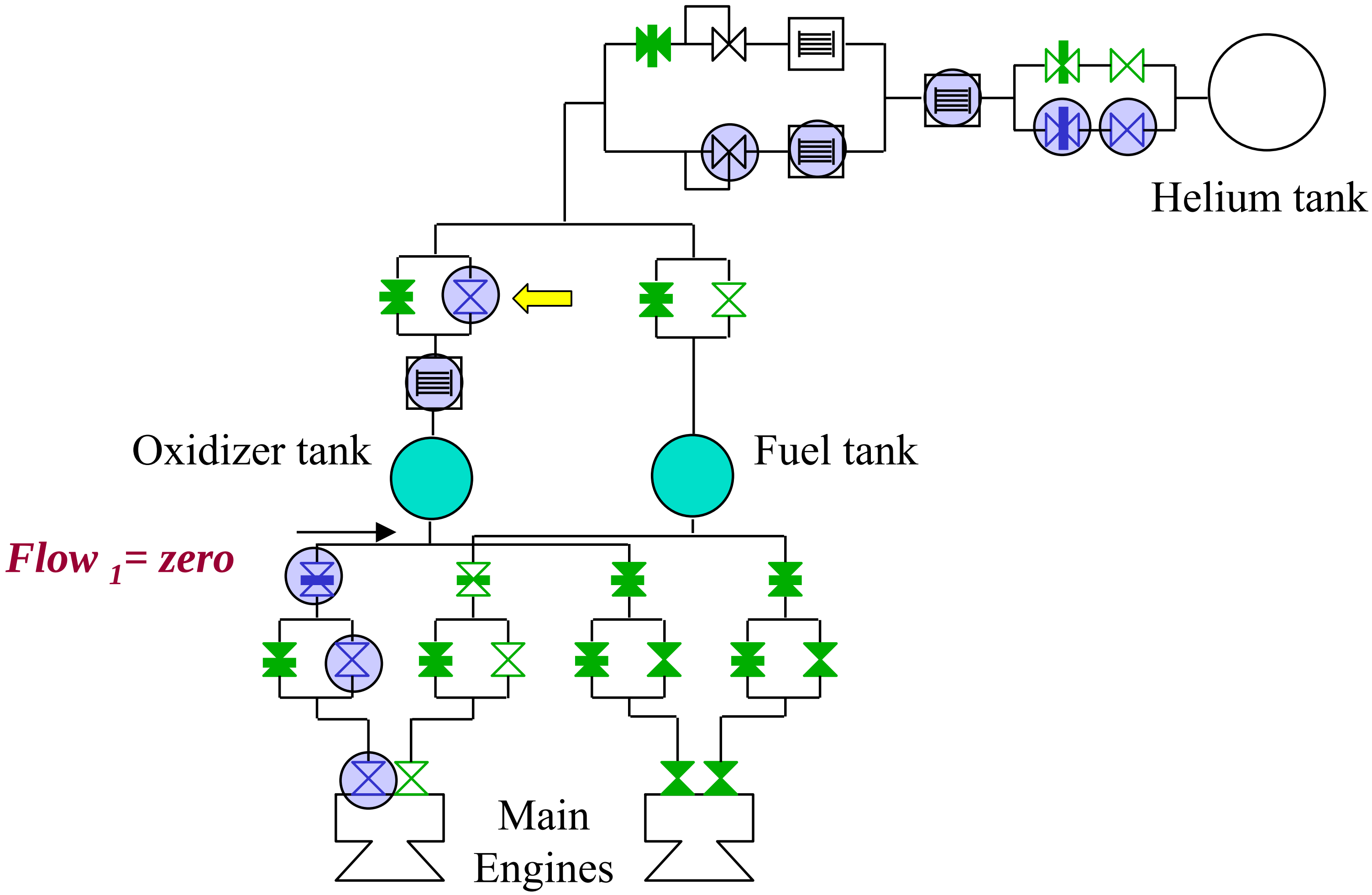
It is most likely that all components are okay.

Isolate Conflicting Information



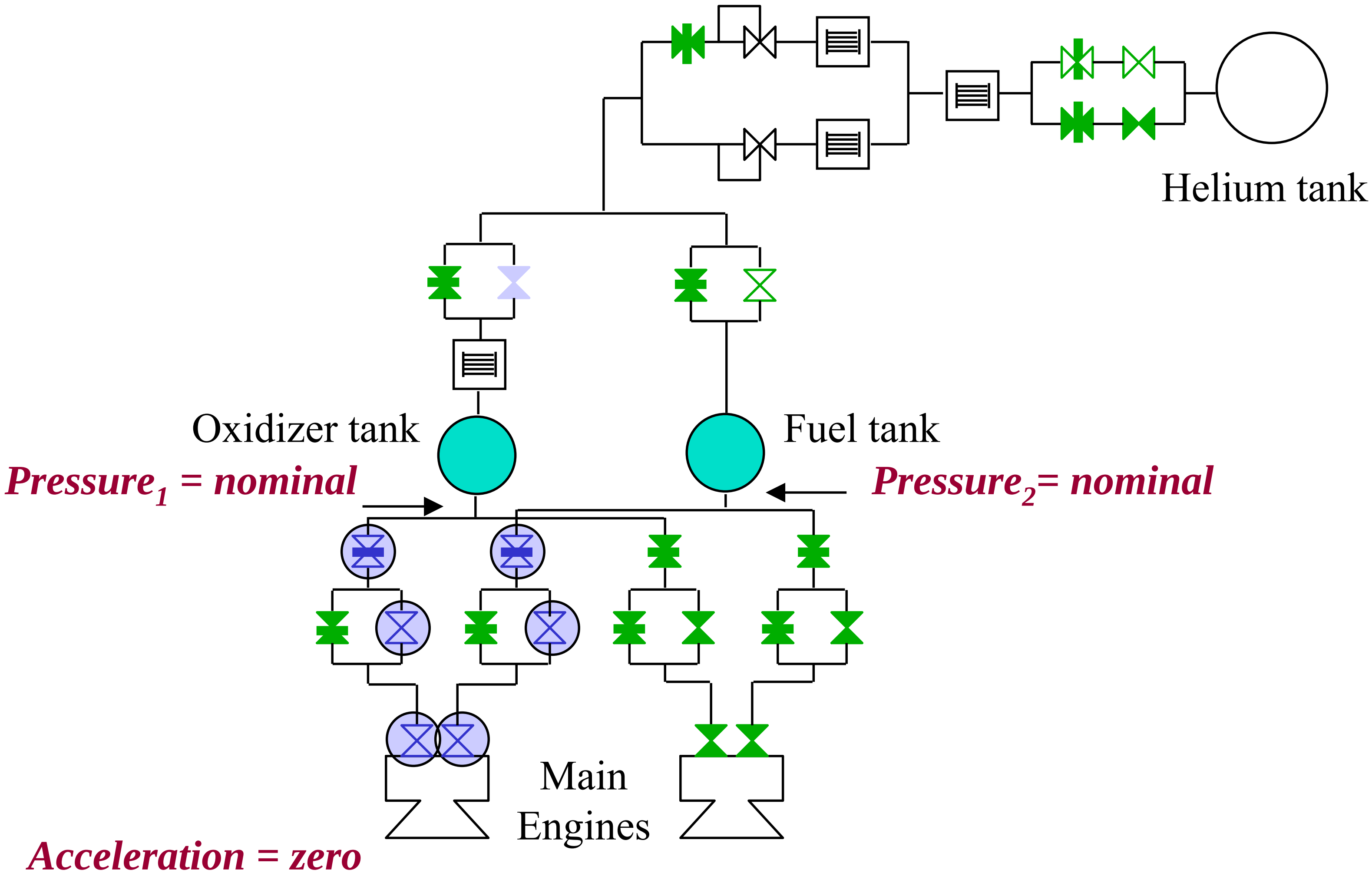
The red component modes *conflict* with the model and observations.

Leap to the Next Most Likely Hypothesis that Resolves the Conflict



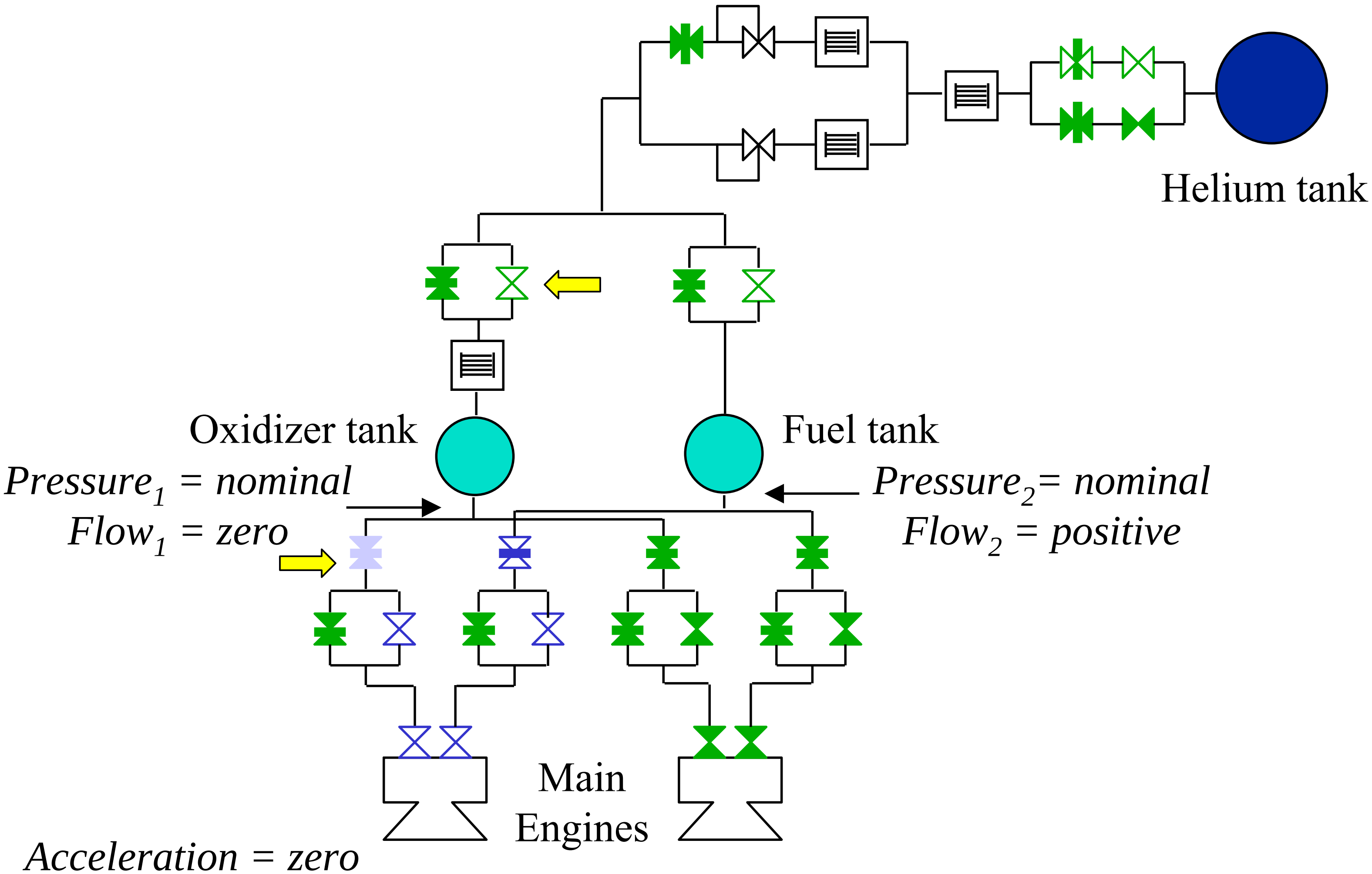
The next hypothesis must remove the conflict

New Hypothesis Exposes Additional Conflicts



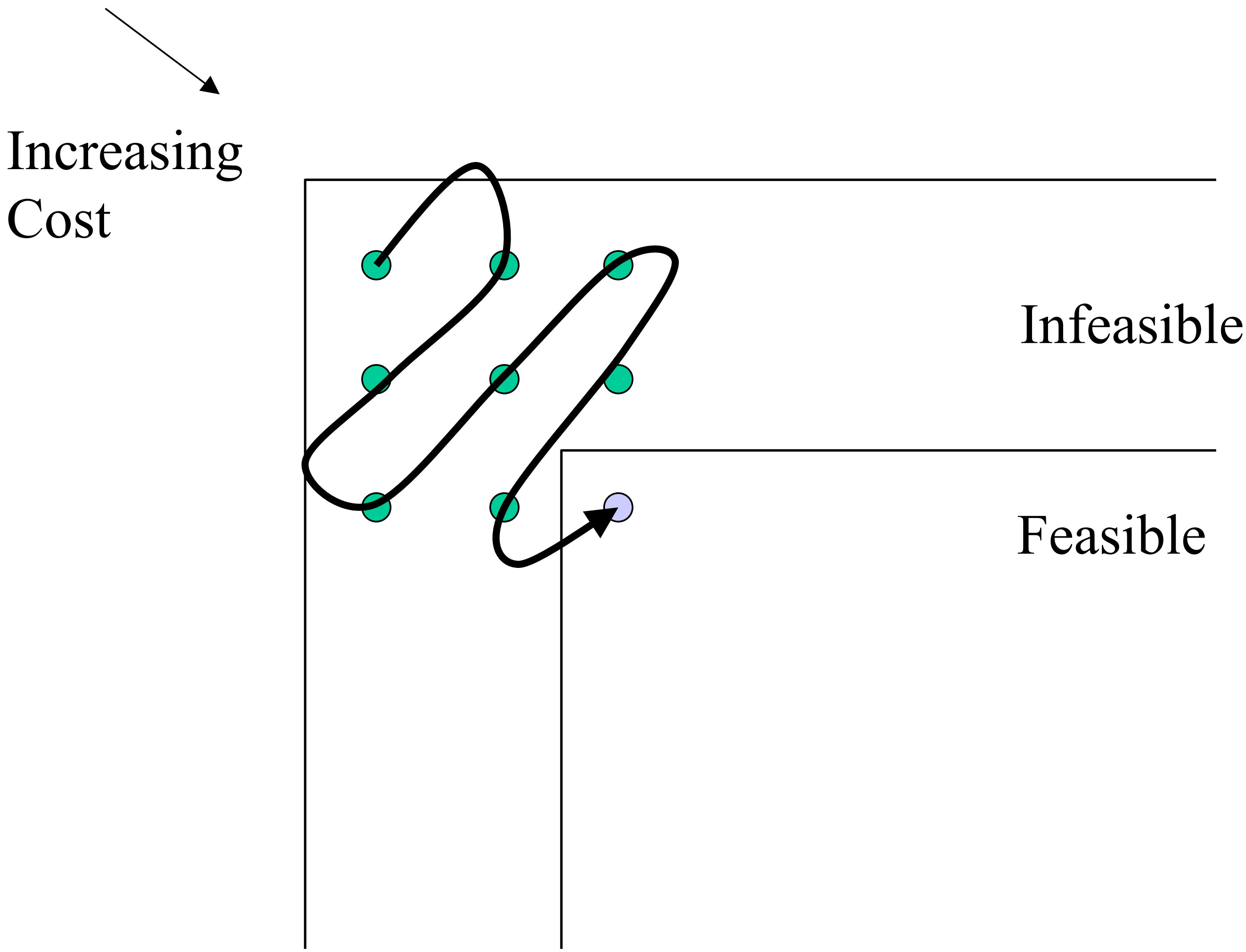
Another conflict, try removing both

Final Hypothesis Resolves all Conflicts

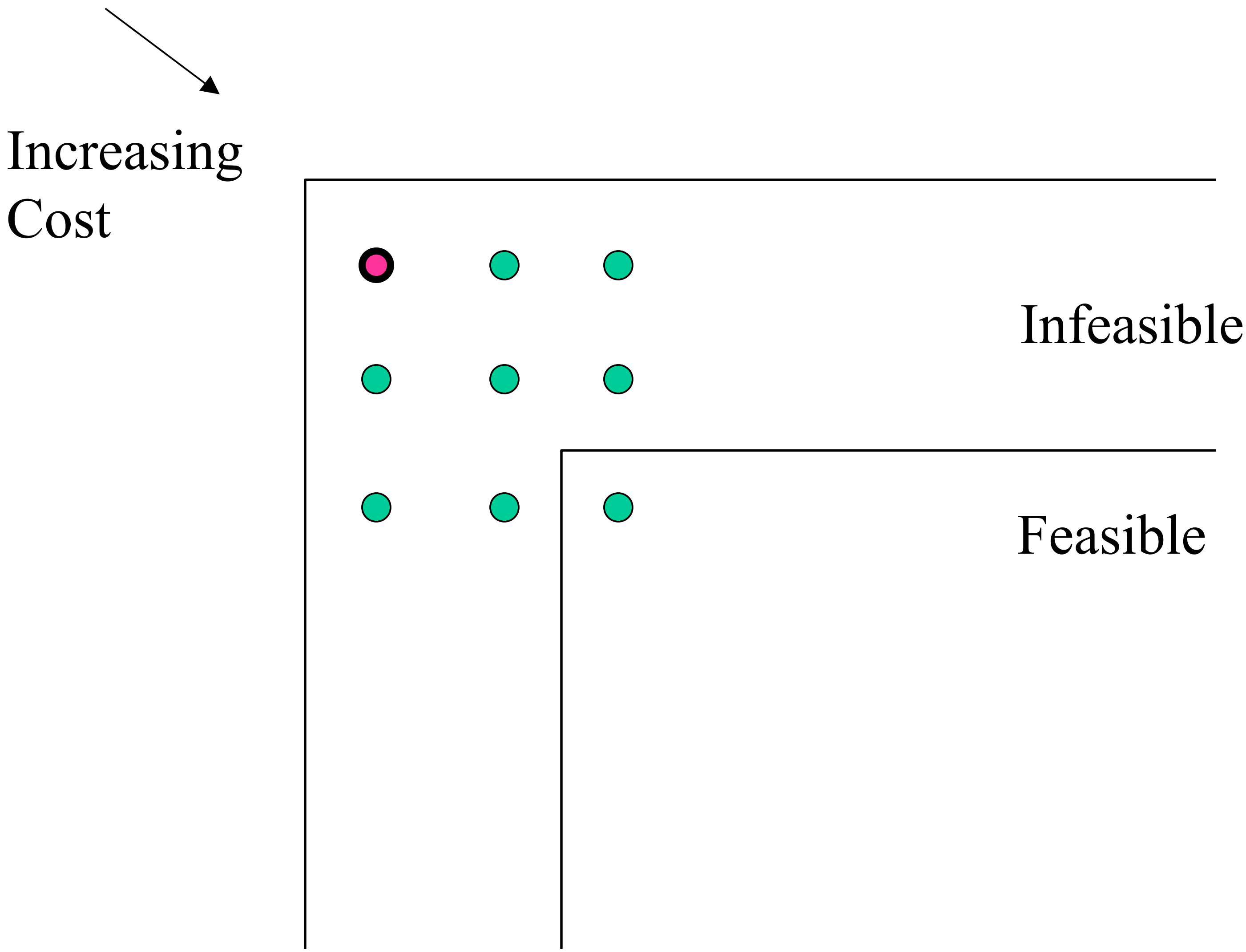


Implementation: Conflict-directed A* search.

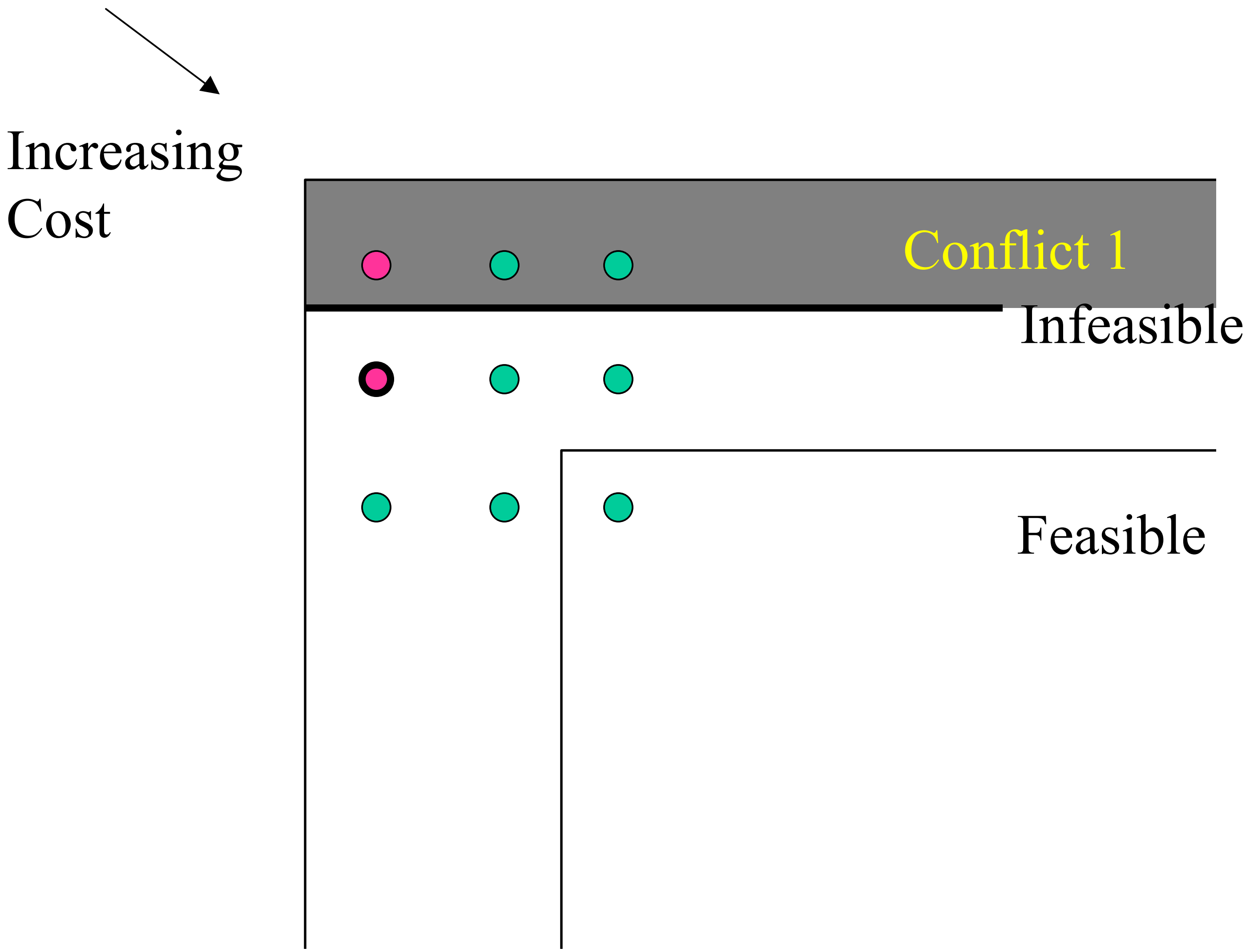
A*



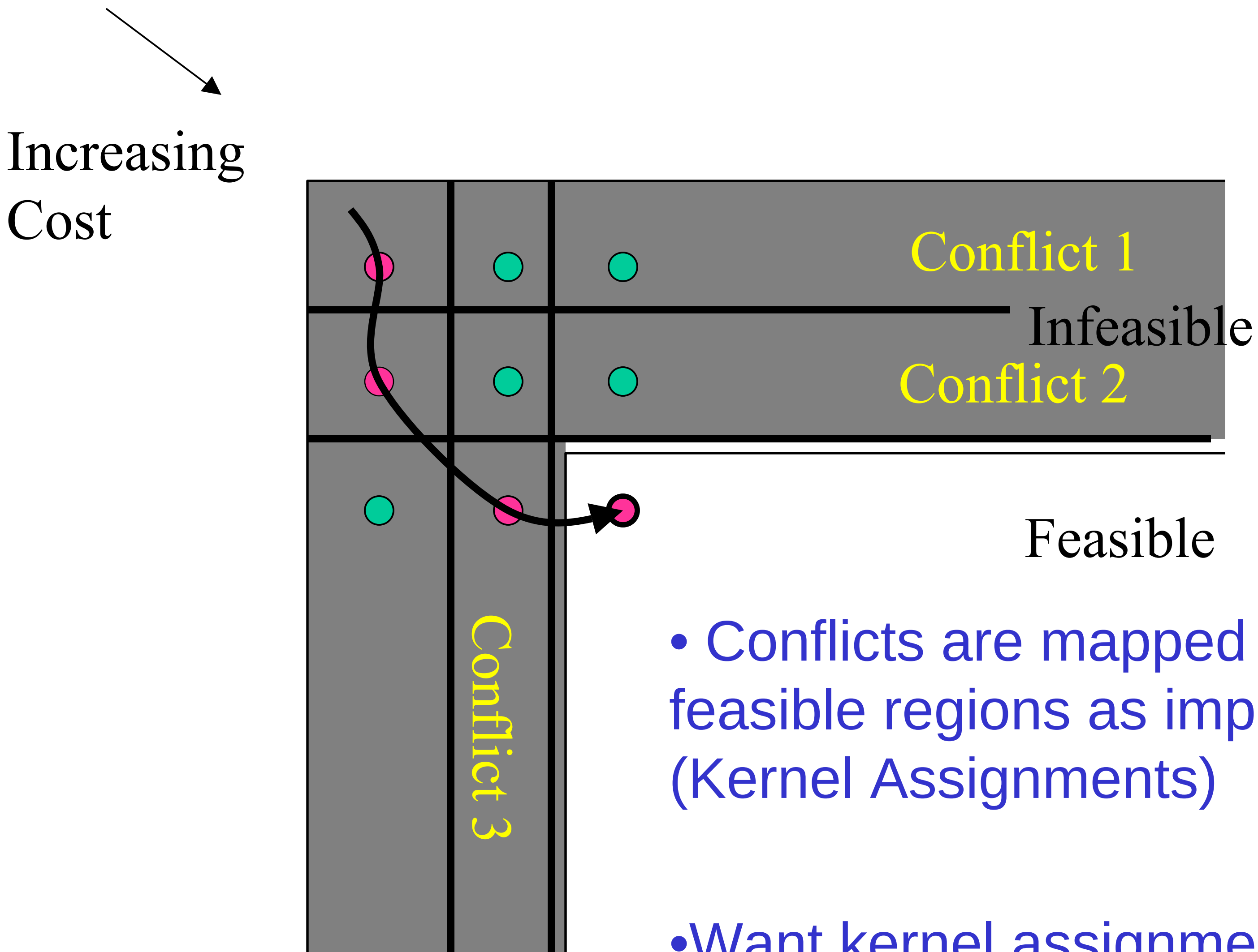
Conflict-directed A*



Conflict-directed A*



Conflict-directed A*



- Conflicts are mapped to feasible regions as implicants (Kernel Assignments)

- Want kernel assignment containing the best cost state.

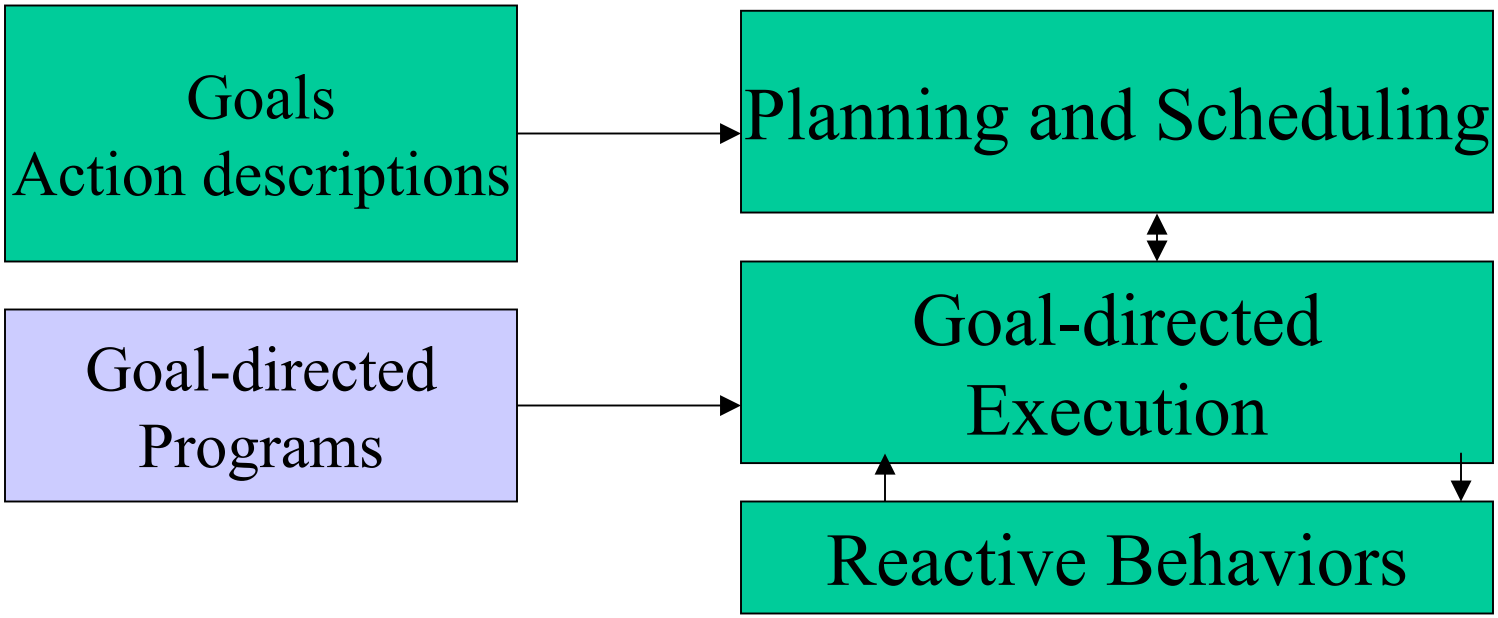
Outline

- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy

Coordination is Recapitulated at the Level of Cooperating Explorers

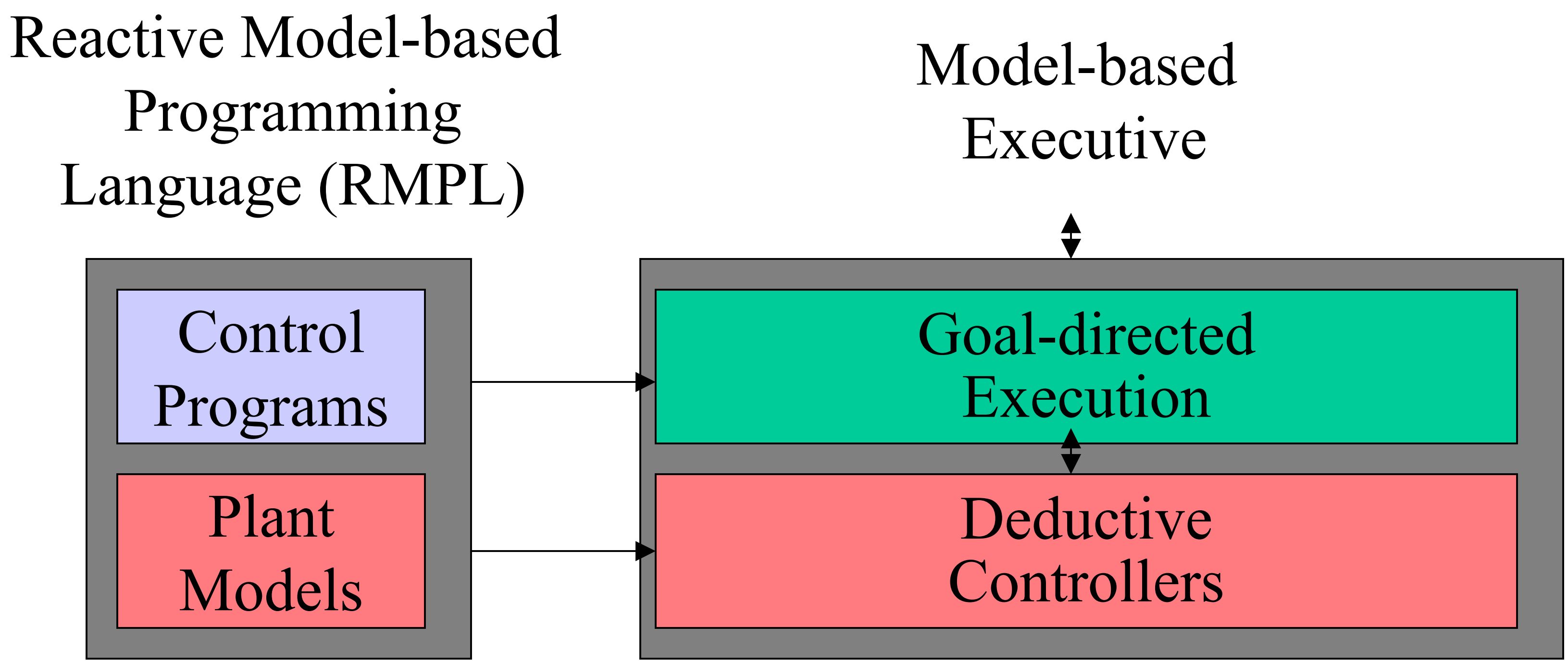
(Courtesy of Jonathan How. Used with permission.)

Traditional Robot Architectures



- Explicit human guidance is at the lowest levels

RMPL for Robotics



What types of reasoning should the programmer/operator guide?

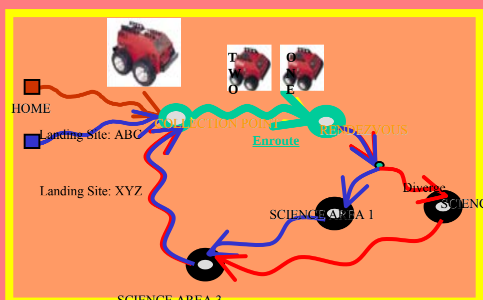
- State/mode inference
- Machine control
- Scheduling
- Method selection
- Roadmap path planning
- Optimal trajectory planning
- Generative temporal planning

RMPL Model-based Program

Control Program

- Executes concurrently
- Preempts
- non-deterministic choice
- A[l,u] timing
- A at l location

Environment Model



Kirk Model-based Executive

Control Sequencer

Predictive Strategy Selection

Dynamic Scheduling

Ensures Safe Execution

location estimates

location goals

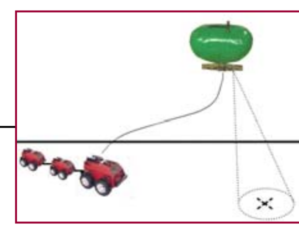
Deductive Controller

Achieves State via Path Planning

Estimates using Localization

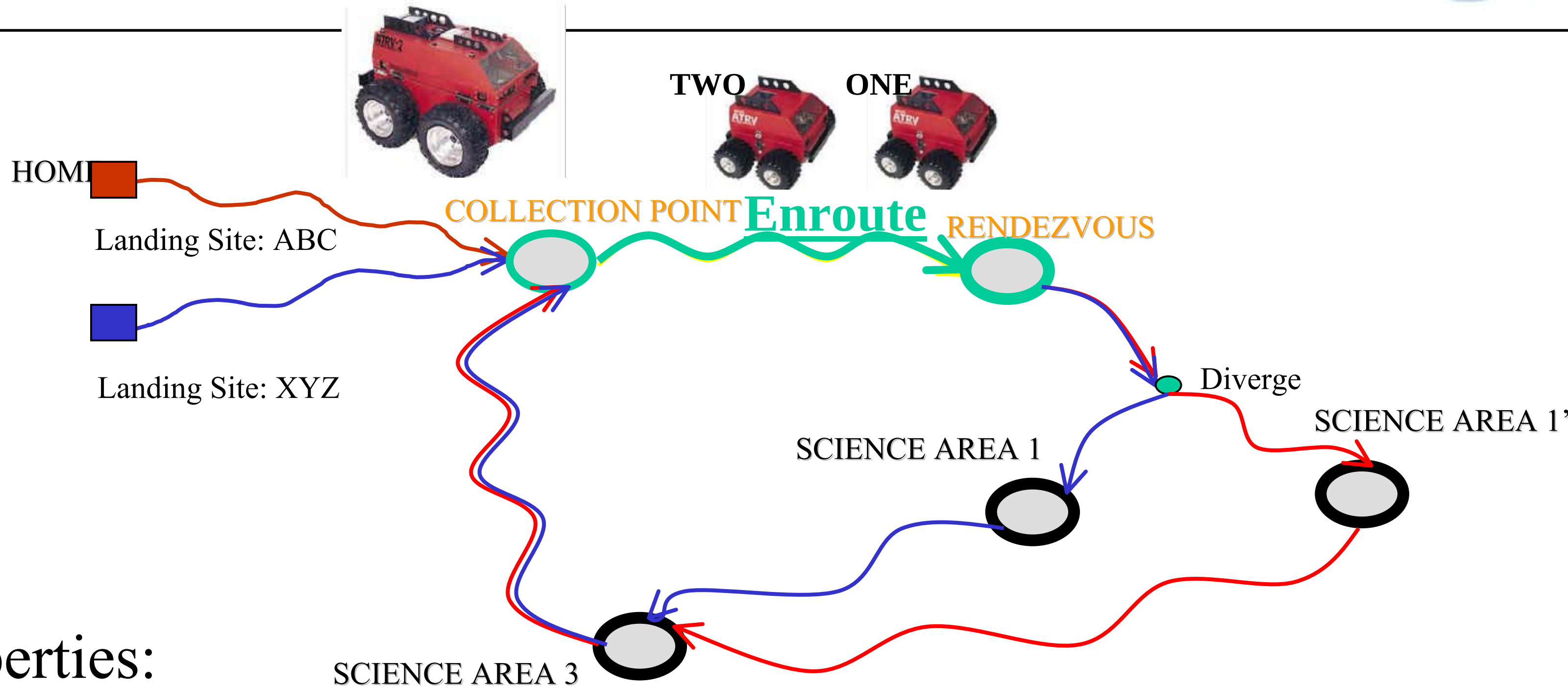
Observations

Commands



Plant

Example Scenario



Properties:

- Mars rover operators have been leery of generative planners.
- Are more comfortable with specifying contingencies.
- Want strong guarantees of safety and robust to uncertainty.
- Global path planning is on the edge

Extend RMPL with planner-like capabilities ..except planning

Reactive Model-based Programming

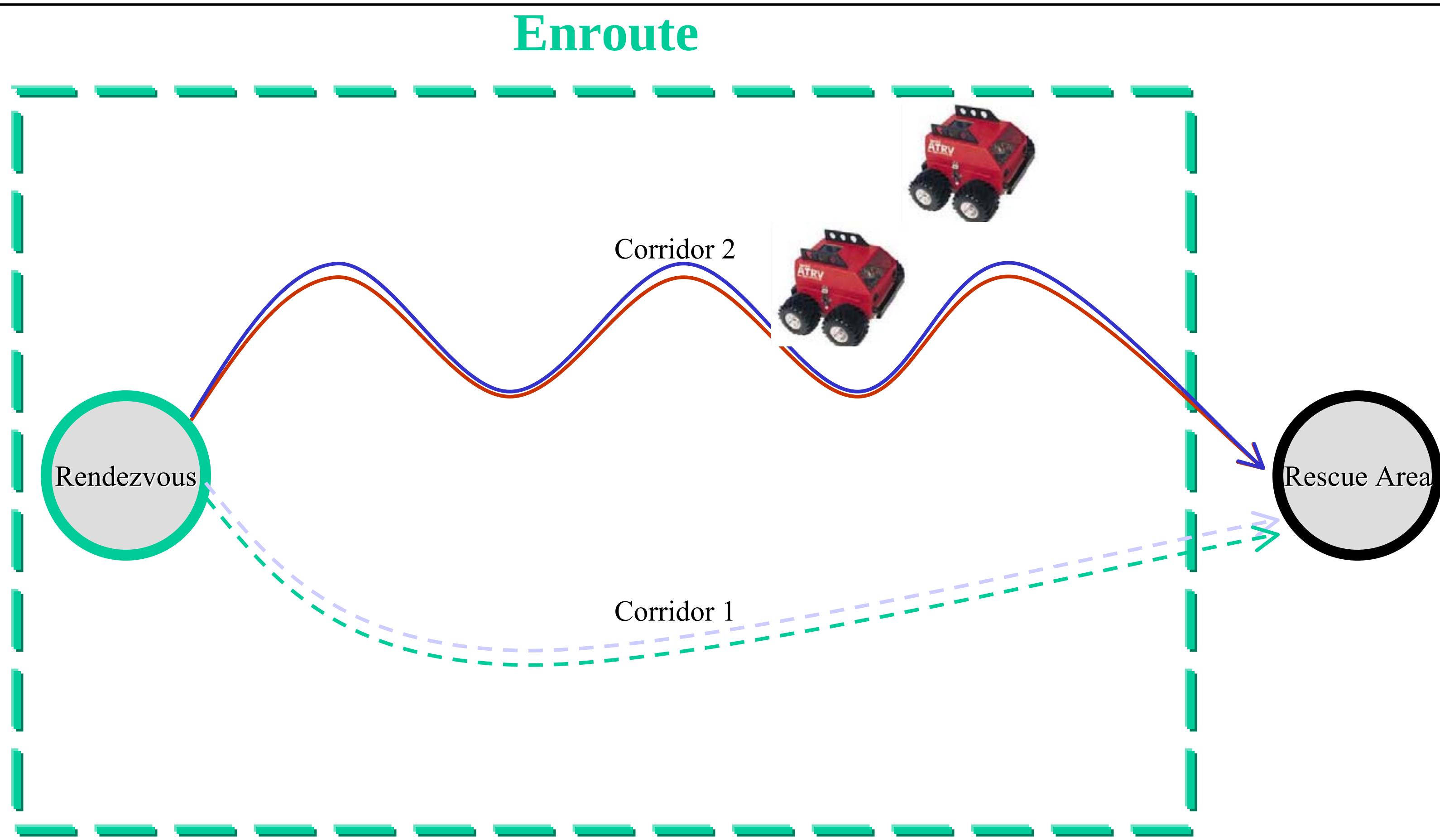
Idea: To describe group behaviors, start with concurrent language:

- p
- If c next A
- Unless c next A
- A, B
- Always A

- Primitive activities
- Conditional execution
- Preemption
- Full concurrency
- Iteration

- Add temporal constraints:
  - A [l,u]
  - Timing
- Add choice (non-deterministic or decision-theoretic):
  - Choose {A, B}
  - Contingency
- Parameterize by location:
  - A at [l]

Example Enroute Activity:



RMPL for Group-Enroute

Temporal Constraints:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Fly-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Fly-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   } at RE_POS
}
```

RMPL for Group-Enroute

Location Constraints:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Fly-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Fly-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   } at RE_POS
}
```

RMPL for Group-Enroute

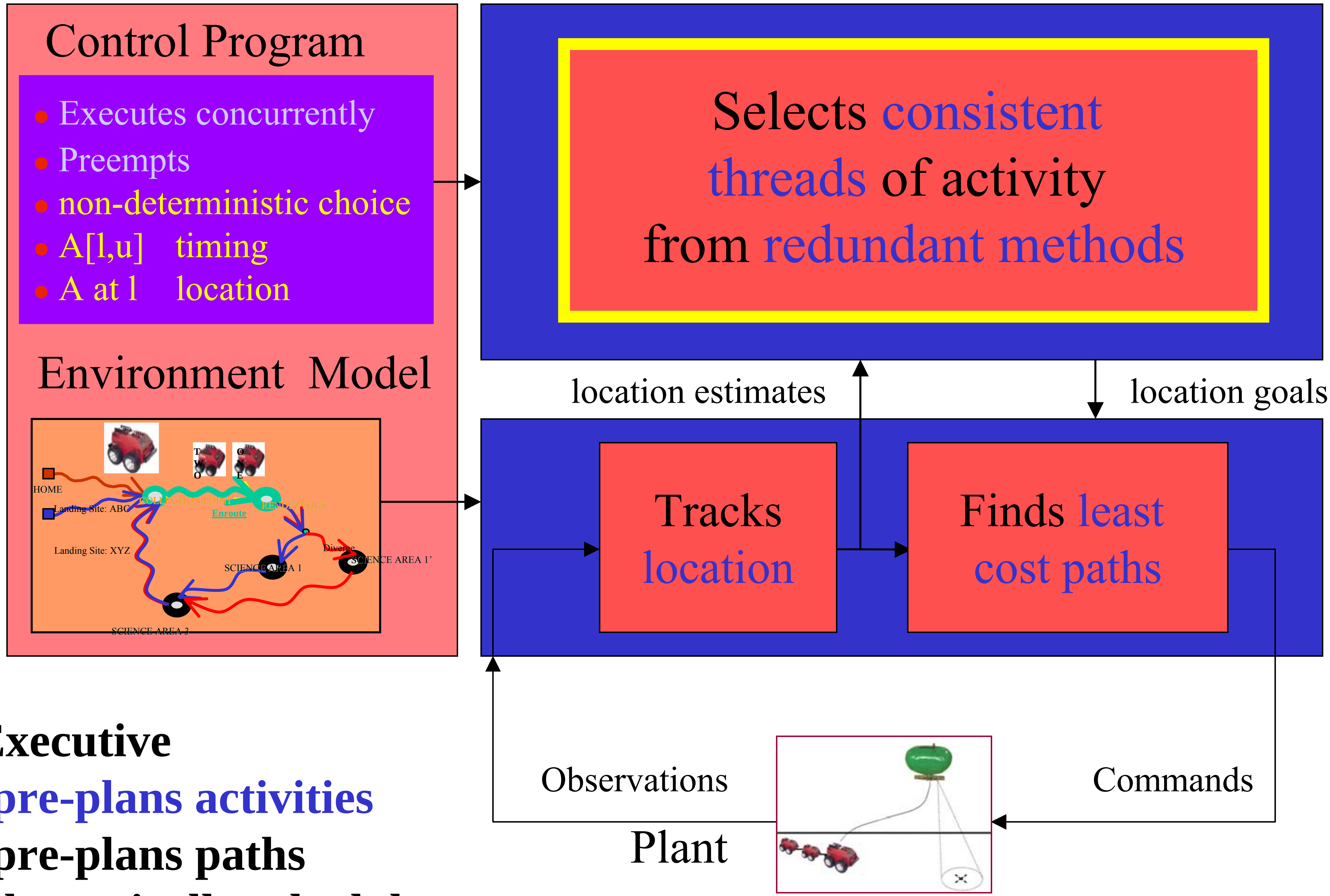
Non-deterministic choice:

```
Group-Enroute()[l,u] = {
   choose {
      do {
         Group-Traverse-
   Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH1_OK,
      do {
         Group-Traverse-
   Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)[l*90%,u*90%];
      } maintaining PATH2_OK
   };
   {
      Group-Transmit(OPS,ARRIVED)[0,2],
      do {
         Group-Wait(HOLD1,HOLD2)[0,u*10%]
      } watching PROCEED
   } at RE_POS
}
```

Outline

- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy

RMPL Model-based Program   Titan Model-based Executive

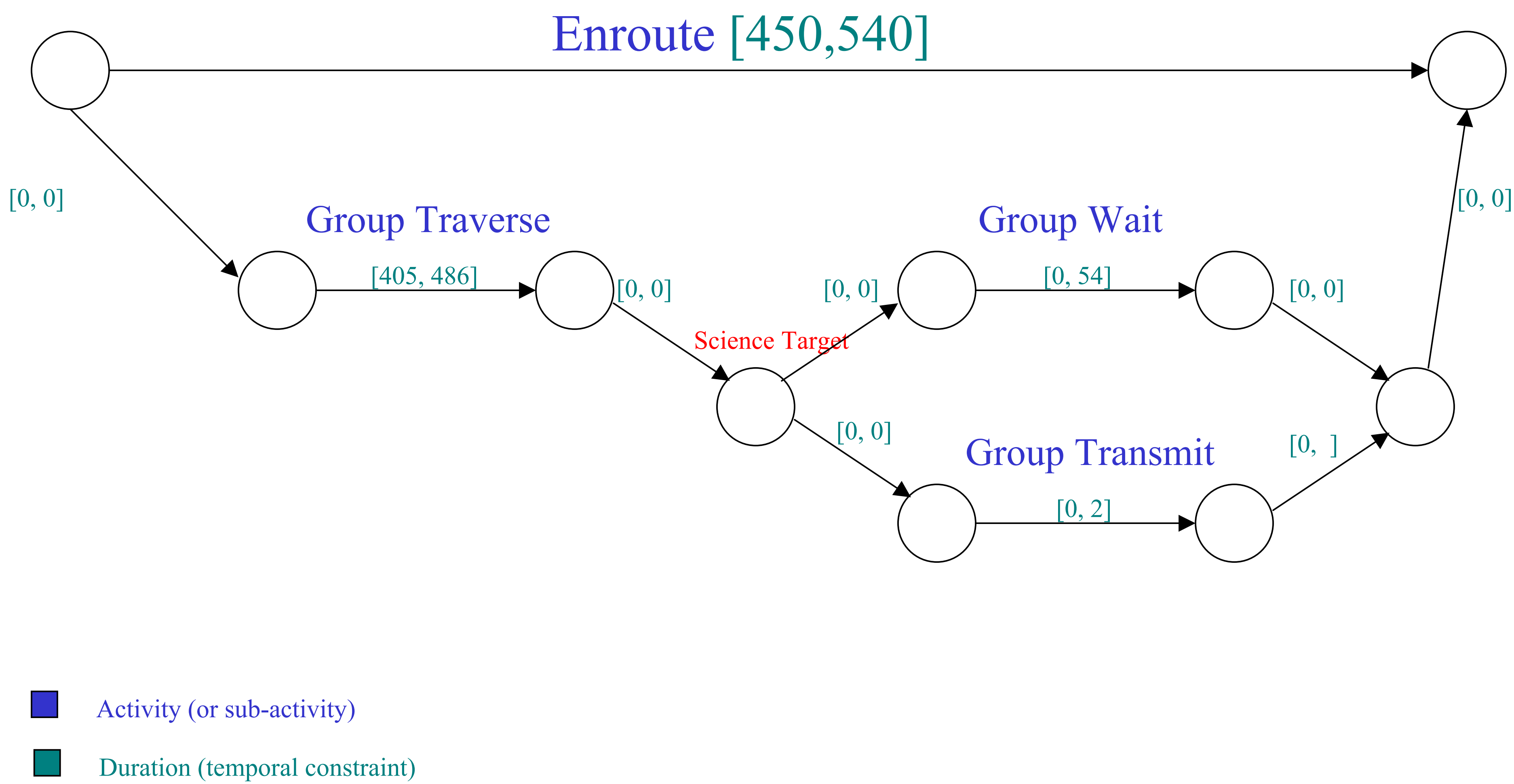


Executive

- **pre-plans activities**
- **pre-plans paths**
- **dynamically schedules [Tsmardinos et al.]**

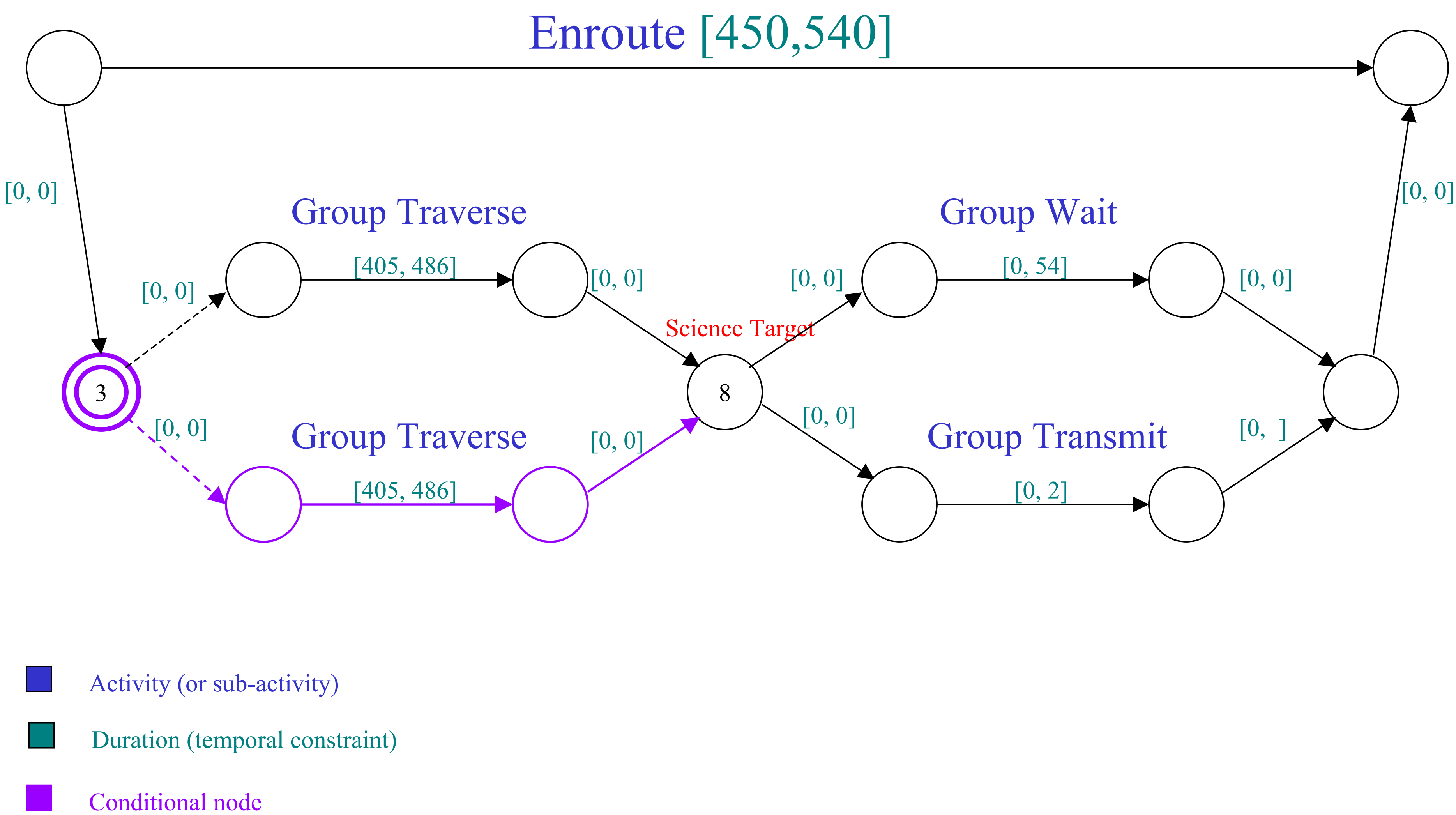
Enroute Activity Encoded as a Temporal Plan Network

- Start with flexible plan representation



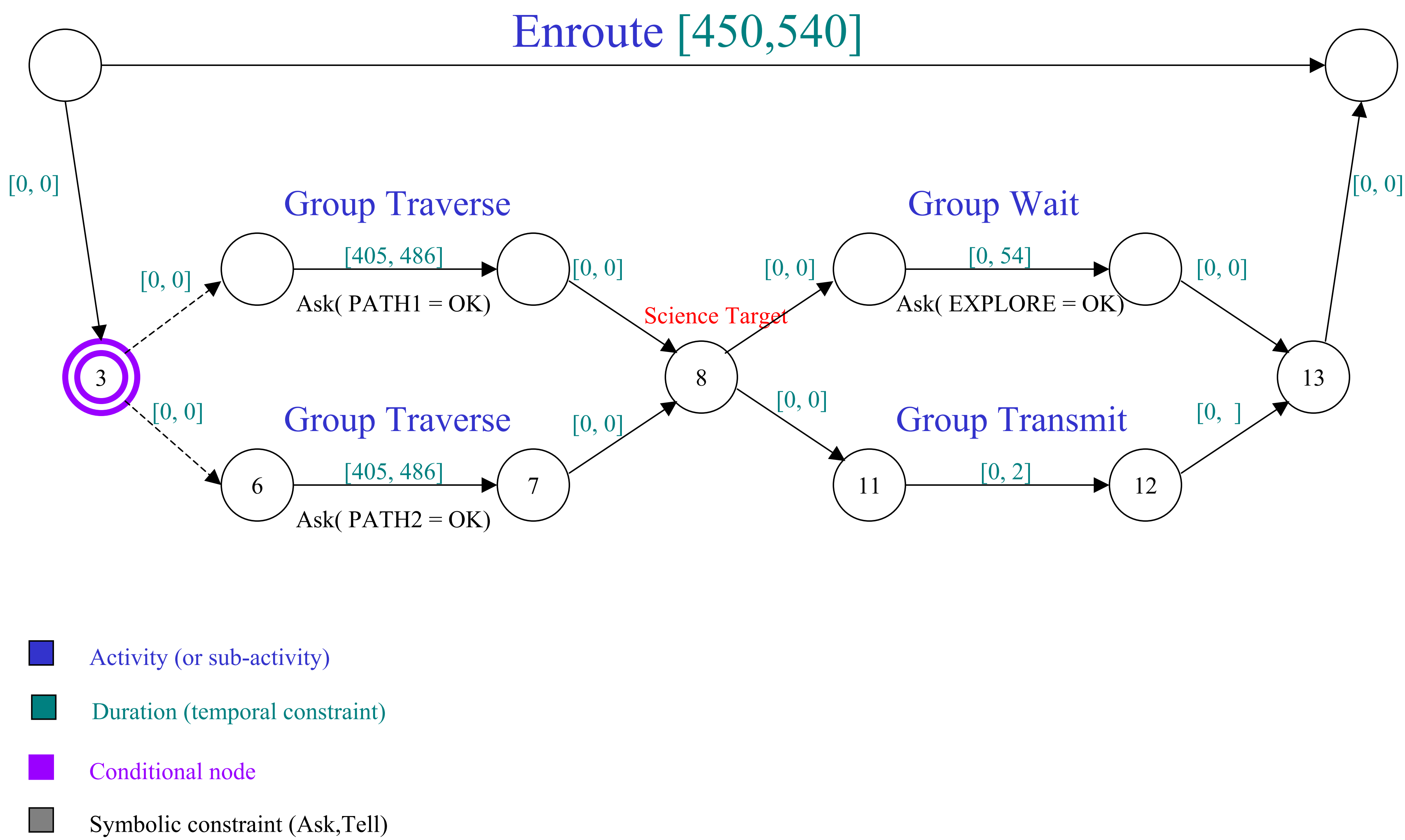
Enroute Activity Encoded as a Temporal Plan Network

- Add conditional nodes



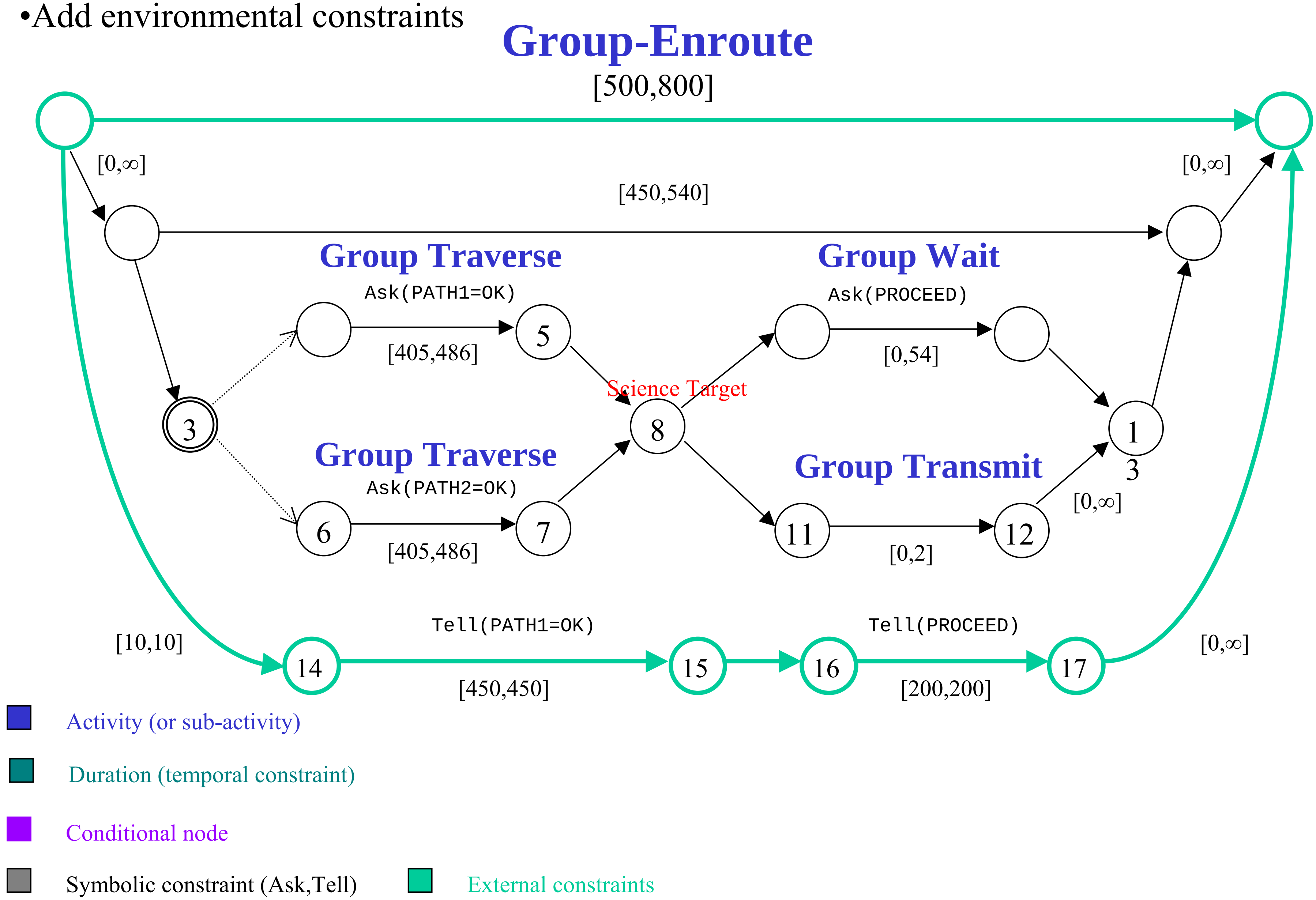
Enroute Activity Encoded as a Temporal Plan Network

- Add temporally extended, symbolic constraints

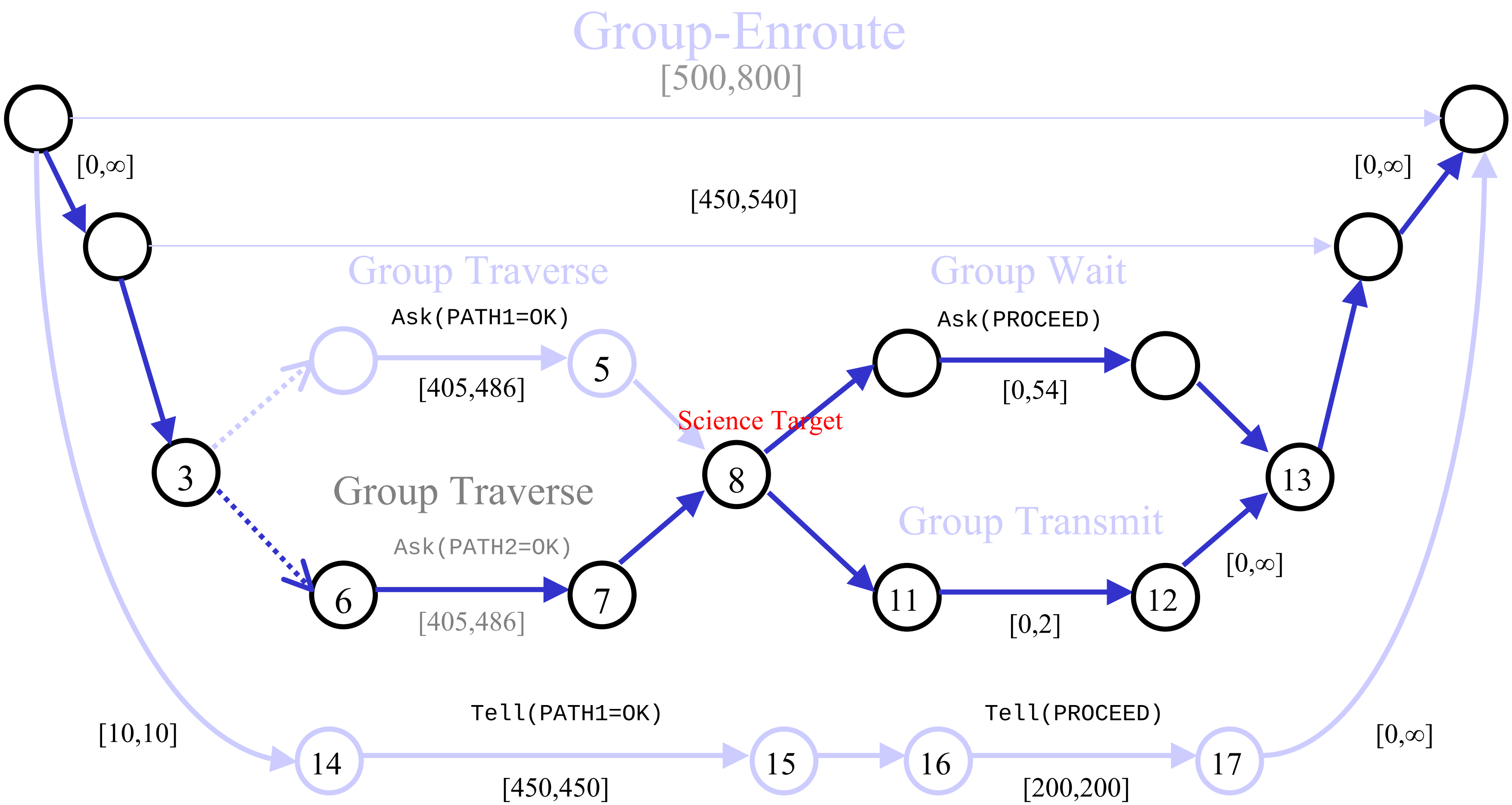


Instantiated Enroute Activity

- Add environmental constraints



Generates Schedulable Plan

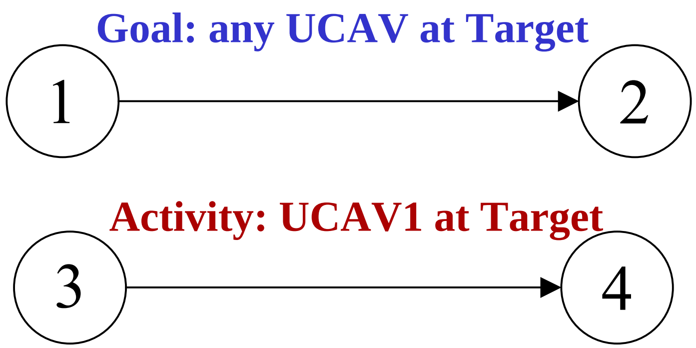


To Plan, . . . perform the following hierarchically:

- **Trace trajectories**
- **Check schedulability**
    - **Supporting and protecting goals (Asks)**

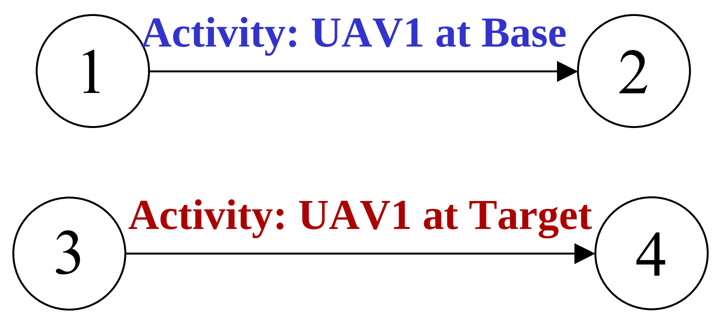
Supporting and Protecting Goals

Unsupported Subgoal



Close open goals

Threatened Activities



Activities can't co-occur

Resolving Unsupported Subgoals:

- Scan plan graph,identifying **activities** that **support open sub-goals;** force to **co-occur**.

Resolving Threatened Subgoals:

- Search for **inconsistent activities** that **co-occur**, and **impose ordering**.

Key computation is bound time of occurrence:

- Used Floyd-Warshall APSP algorithm **$O(V^3)$.**

Randomized Experiments for Assessing Scaling and Robustness

Randomized Experiments:

- Randomly generated range of scenarios with **1-50 vehicles**.
- Each vehicle has **two scenario options**, each with **five actions** and **2 waypoints**:

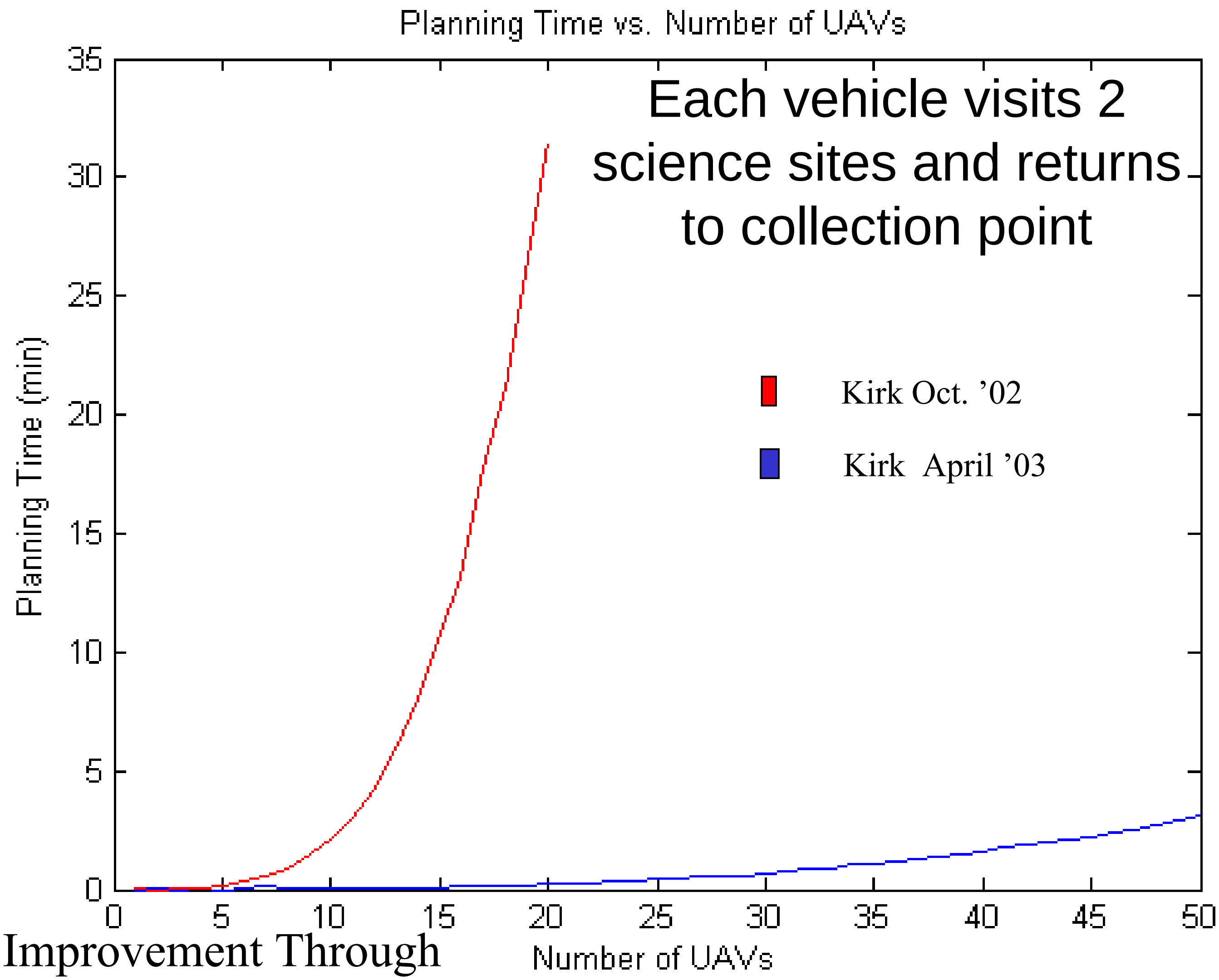
  1. Go to waypoint 1
  2. Observe science
  3. Go to waypoint 2
  4. Observe science
  5. Return to collection point

- **Waypoints generated randomly** from environment with uniform distribution.

Strategy Selection:

- TPN planner chooses **one option per vehicle**.
- **Combined choices** must be **consistent** with **timing constraints** and **vehicle paths**.

Kirk Strategy Selection: Scaling and Robustness



Performance Improvement Through

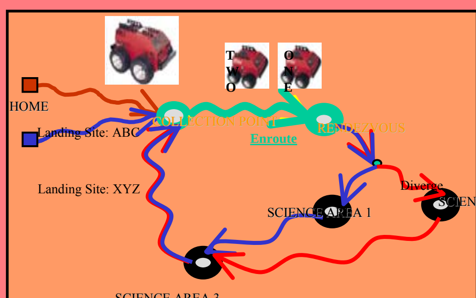
- Incremental temporal consistency
- Conflict-directed Search (in progress)

RMPL Model-based Program   Titan Model-based Executive

Control Program

- Executes concurrently
- Preempts
- non-deterministic choice
- A[l,u]   timing
- A at l   location

Environment Model



Selects consistent threads of activity from redundant methods

location estimates

location goals

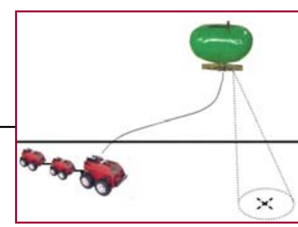
Tracks location

Finds least cost paths

Observations

Commands

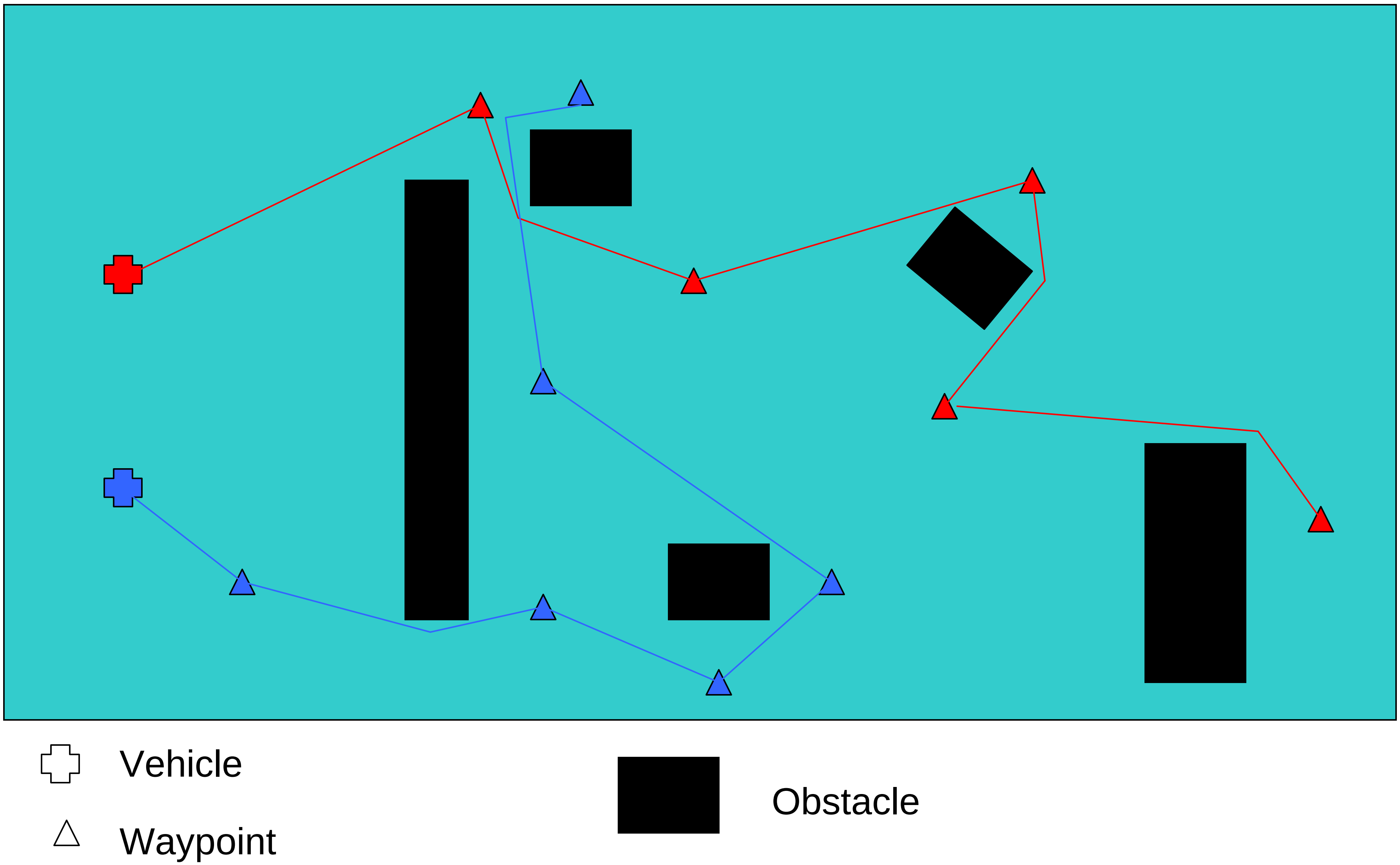
Plant



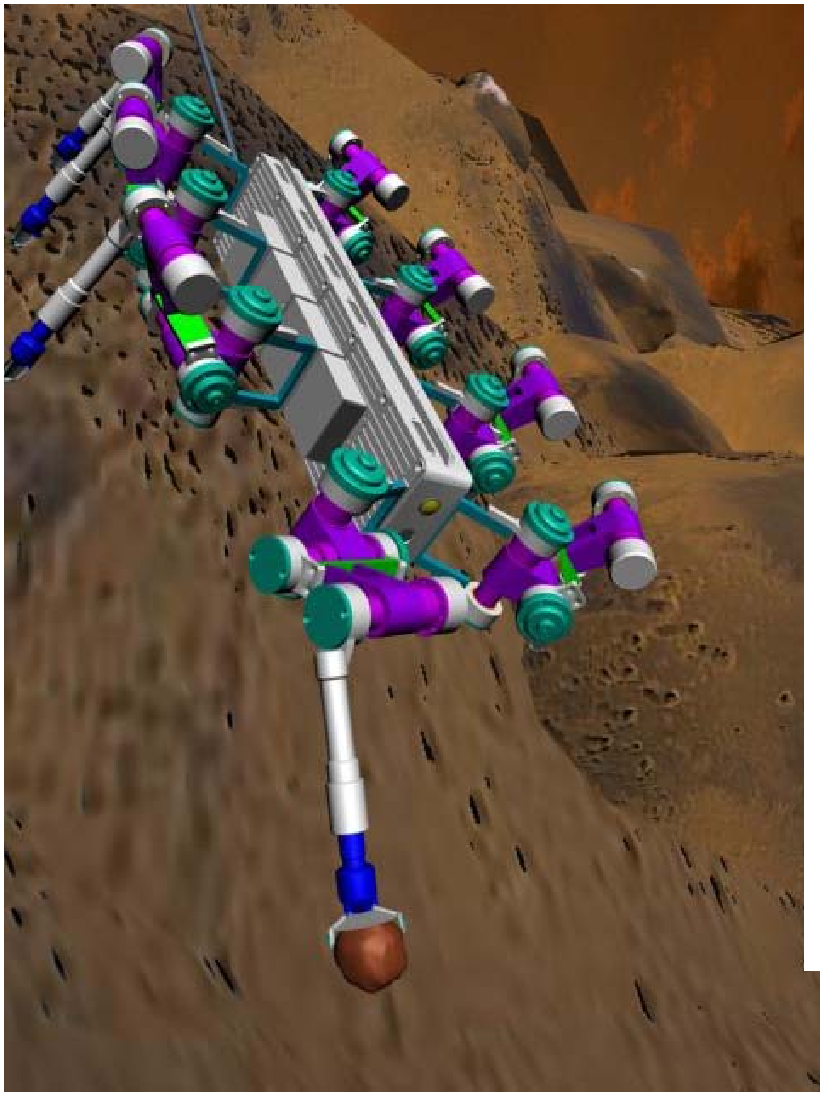
Executive

- **pre-plans activities**
- **pre-plans paths**
- **dynamically schedules [Tsmardinos et al.]**

Achieving Program States Combines Logical Decisions and Trajectory Planning



Explorers Will Need to Be Dexterous



(Courtesy of Frank Kirchner. Used with permission.)

Outline

- Model-based Programming
- Autonomous Engineering Operations
  - An Example
  - Model based Execution
  - Fast Reasoning using Conflicts
- Cooperating Mobile Vehicles
  - Predictive Strategy Selection
  - Planning Out The Strategy

Example: Coaching Heterogeneous Teams

- Search and Rescue
- Ocean Exploration

(Courtesy of Jonathan How. Used with permission.)

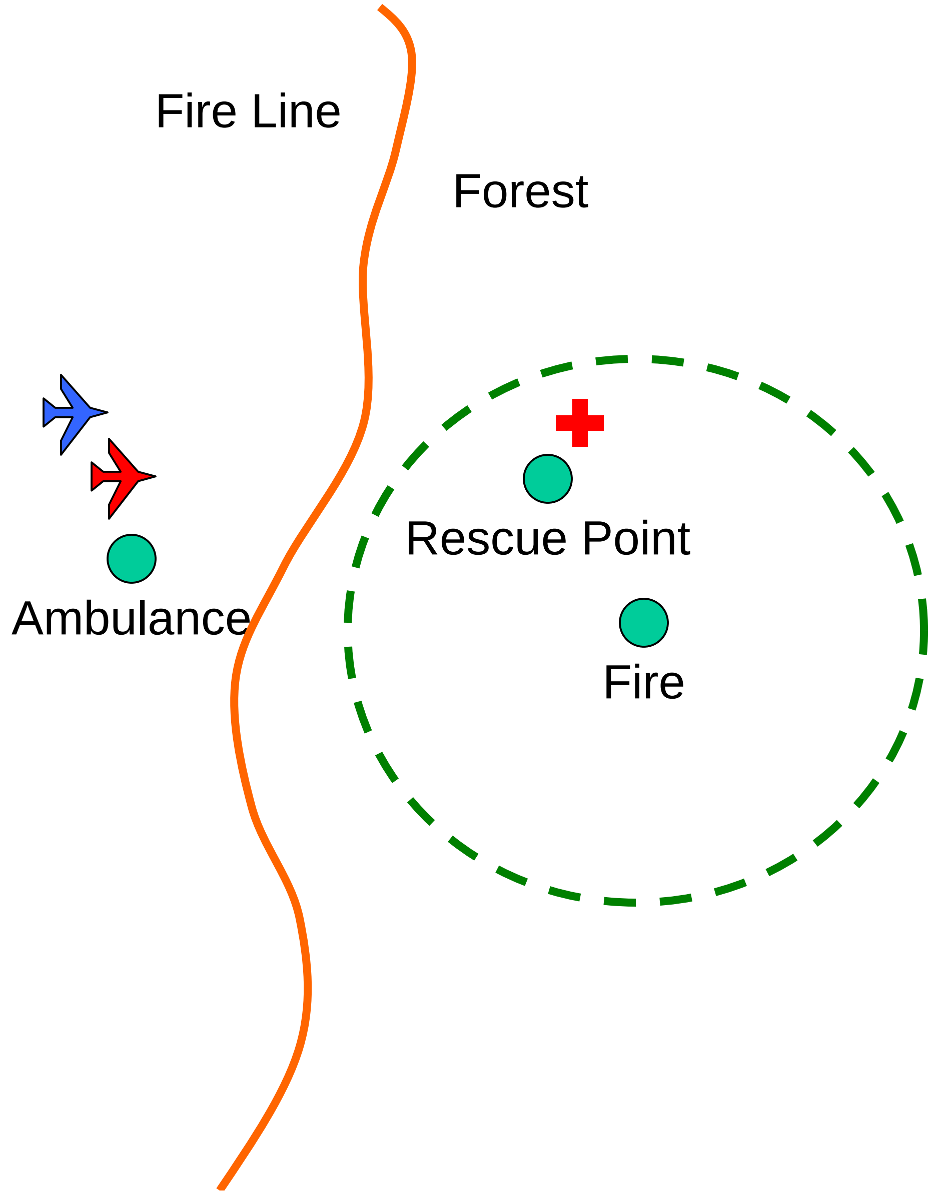
A dozen vehicles is too many to micro manage

→ **Act as a coach:**

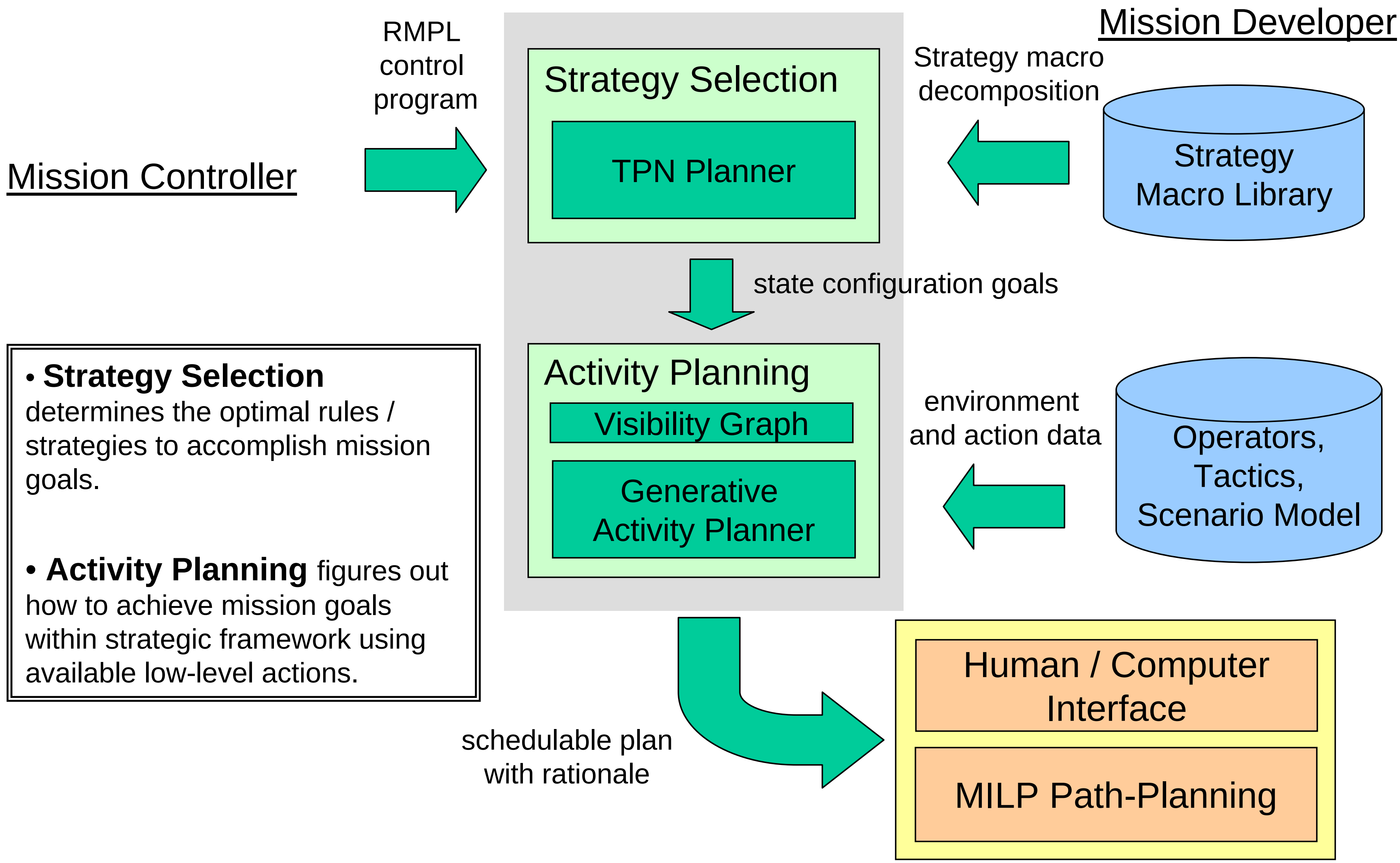
- **Specify evolution of state and location.**

Forest Fire Rescue

- Goal: retrieve family from fire.
- Rescue cannot take place until the local fire is suppressed.
- Retrofit one rescue vehicle for fire suppression



Kirk Model-based Execution System Overview



- **Strategy Selection** determines the optimal rules / strategies to accomplish mission goals.

- **Activity Planning** figures out how to achieve mission goals within strategic framework using available low-level actions.

RMPL Control Program

- (defclass rescue-team

```
(execute ()
  (sequence
    (parallel [l1,u1]
      (tell-start(at uav1 Ambulance))
      (tell-start(at uav2 Ambulance))
      (ask-end(suppressed Fire))
    )
    (parallel [l2,u2]
      (tell-start(at family RescuePoint))
      (ask-end(rescued family))
      (ask-end(at uav1 Ambulance))
      (ask-end(at uav2 Ambulance))
    )
  )
)
)
```

Initial State

Intermediate State

Goal State

Phase 1

Phase 2

Environment Model

- Terrain Map

- Object instantiations:
  – UAV uav1
  – UAV uav2
  – RESCU-READY uav1
  – RESCUE-READY uav2
  – IN-DISTRESS family
  – LOCATION Ambulance
  – LOCATION Fire
  – LOCATION RescuePoint

Vehicle Specifications

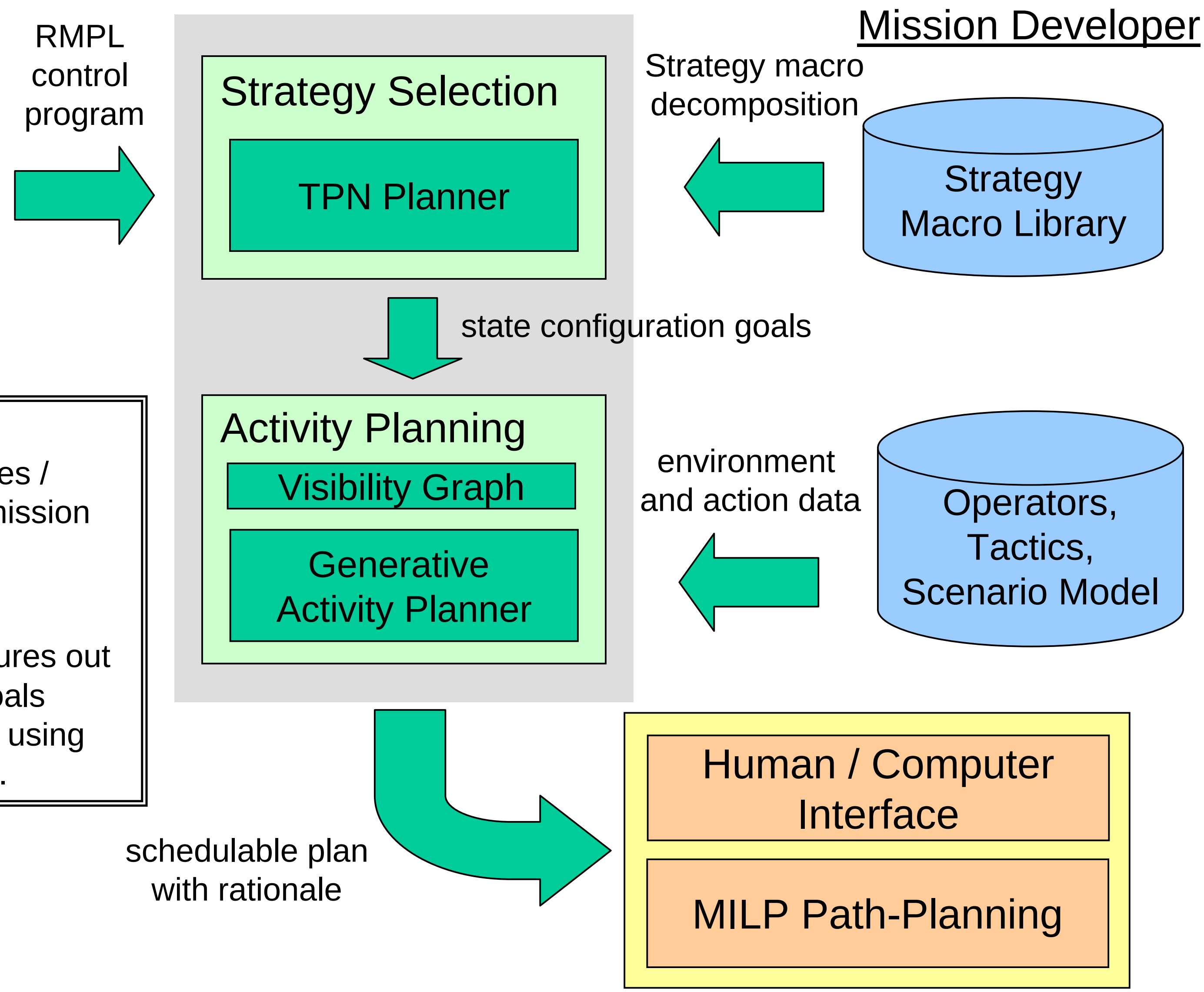
- Vehicle linearized dynamics

- Vehicle primitive operators:

  - Fly(V,A,B)
    - move UAV “V” from location “A” to location “B”
  - Refit(V)
    - Prepare UAV “V” to drop fire retardant
  - Drop(V,A)
    - Drop fire retardant at location “A” with UAV “V”
  - Rescue(V,P,A)
    - Rescue people “P” in distress with UAV “V” at location “A”

Kirk Model-based Execution System Overview

Mission Controller



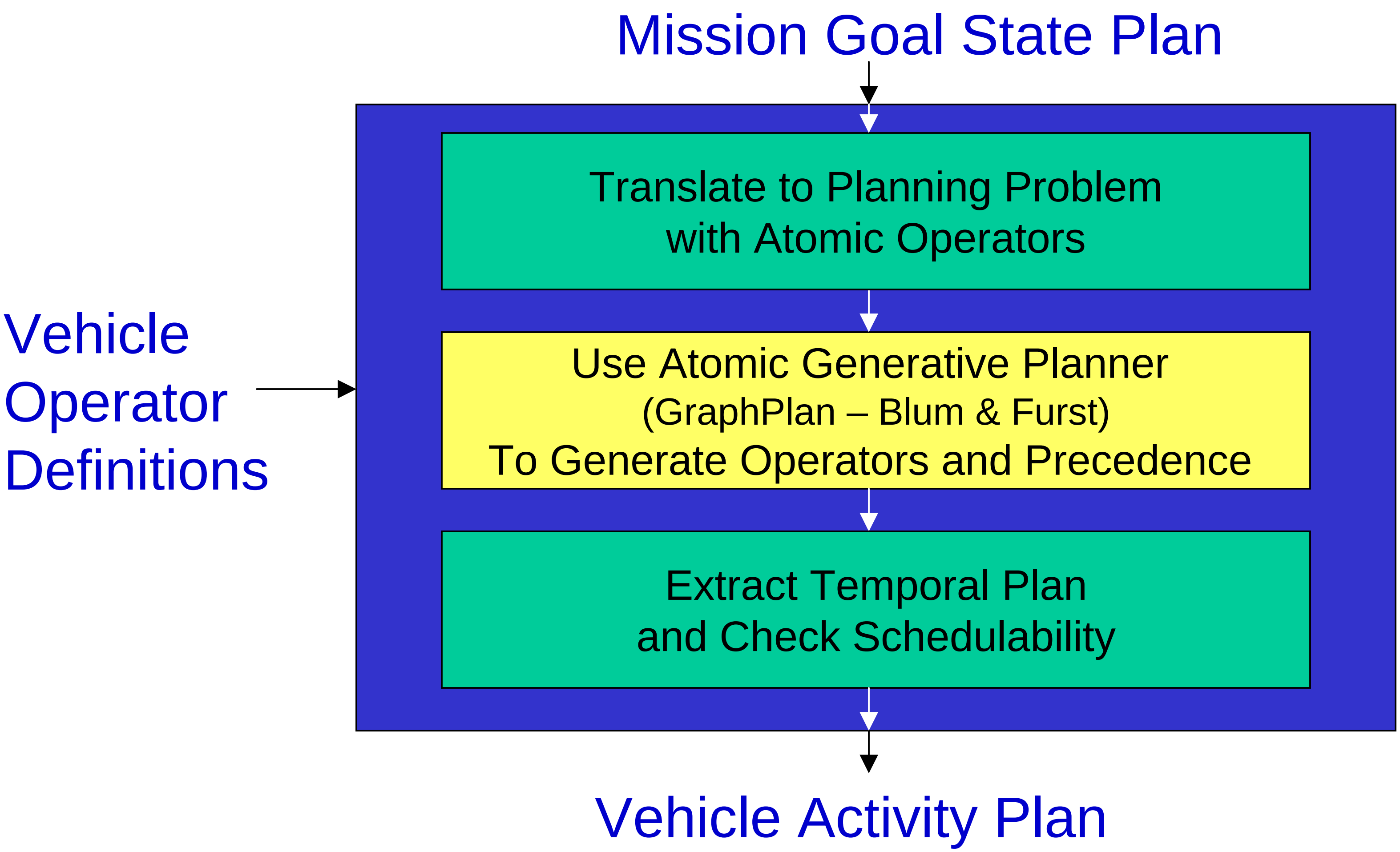
- **Strategy Selection** determines the optimal rules / strategies to accomplish mission goals.

- **Activity Planning** figures out how to achieve mission goals within strategic framework using available low-level actions.

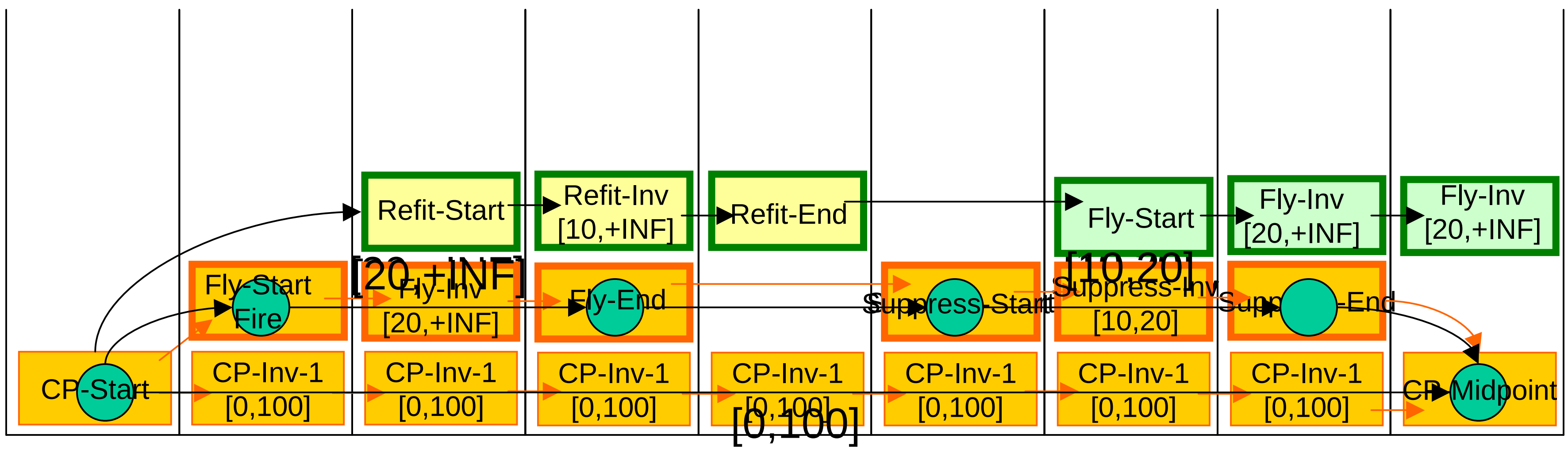
Kirk Constructs Vehicle Activity Plan Using a Generative Temporal Planner

Approach:

- Encode Goal Plan using an LPGP-style encoding
- Prototype using LPGP [Fox/Long, CP03]

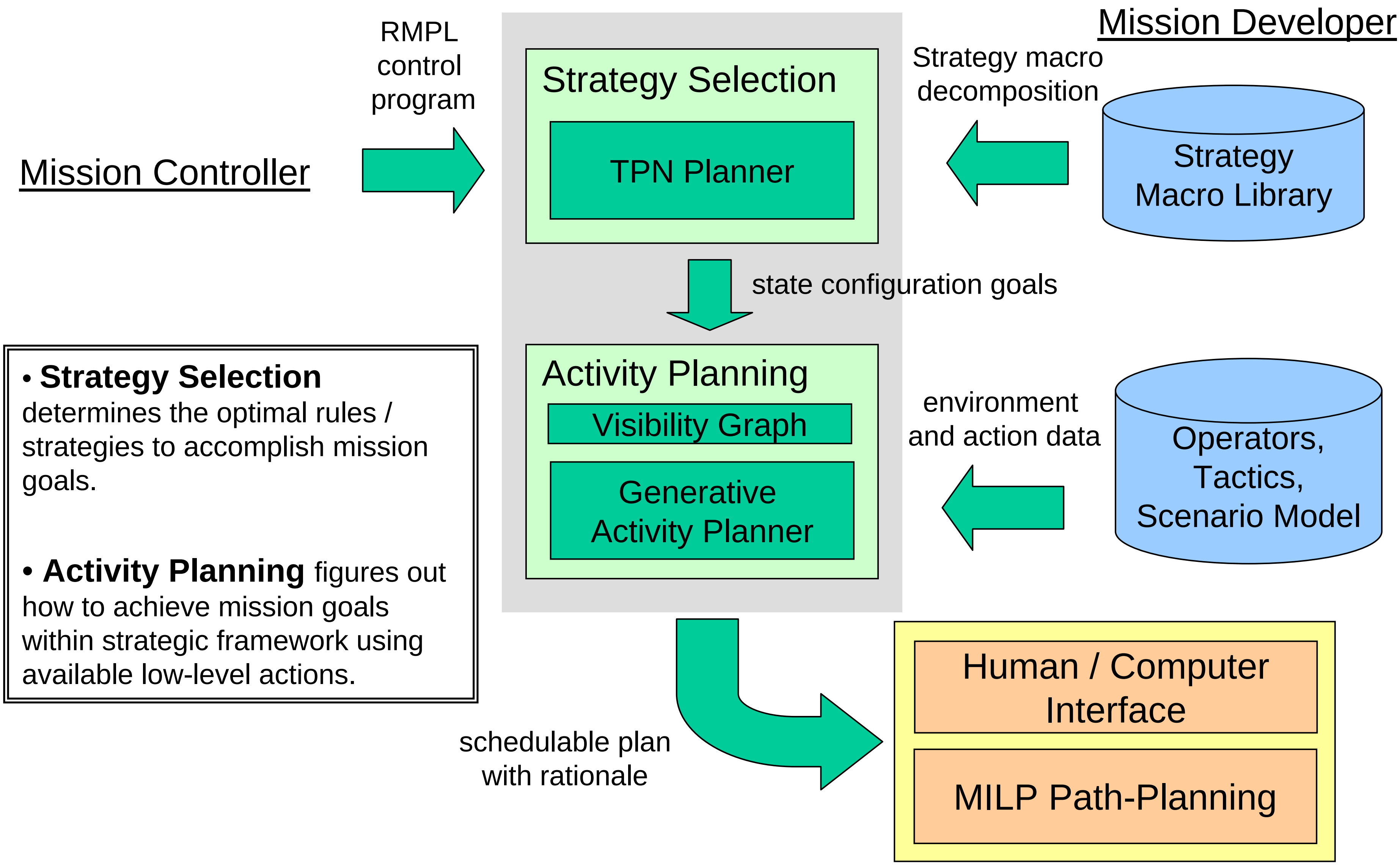


Generated Activity Plan



Kirk extracts a least commitment plan and generates a rationale

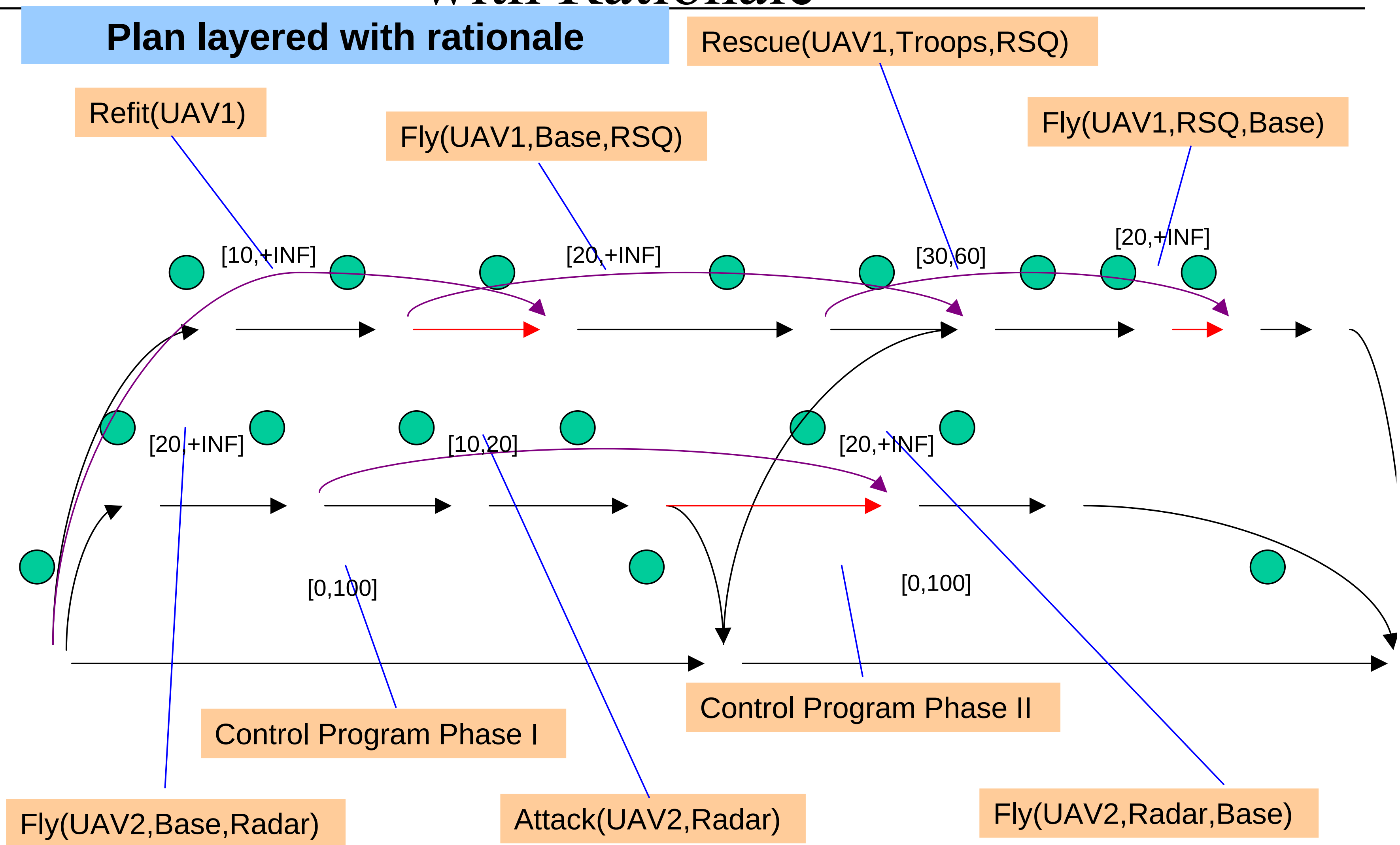
Kirk Model-based Execution System Overview



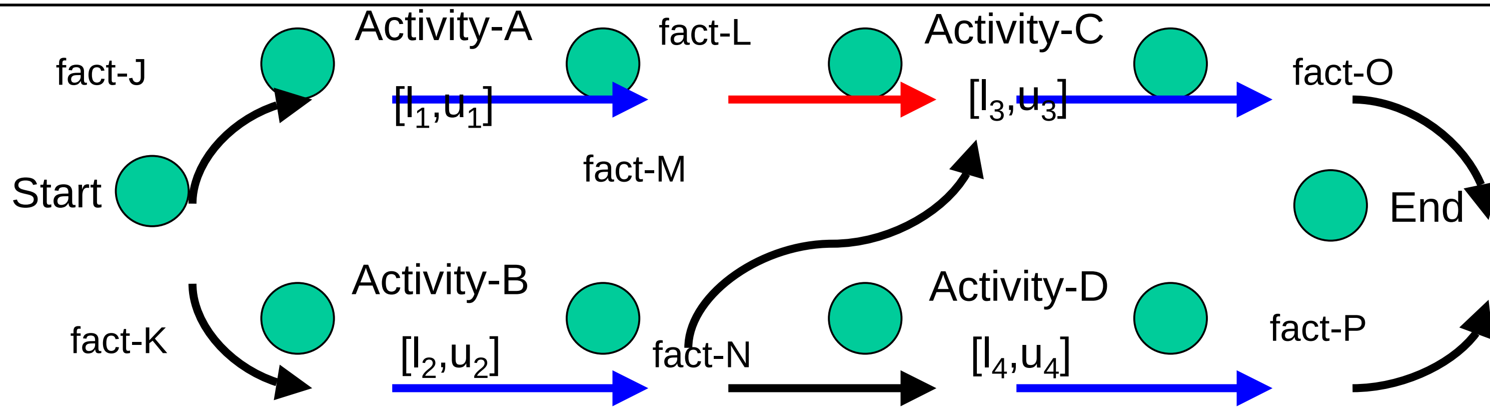
- **Strategy Selection** determines the optimal rules / strategies to accomplish mission goals.

- **Activity Planning** figures out how to achieve mission goals within strategic framework using available low-level actions.

Output: Least Commitment Plan with Rationale



Kirk Ensures Plan Completeness, Consistency and Minimality

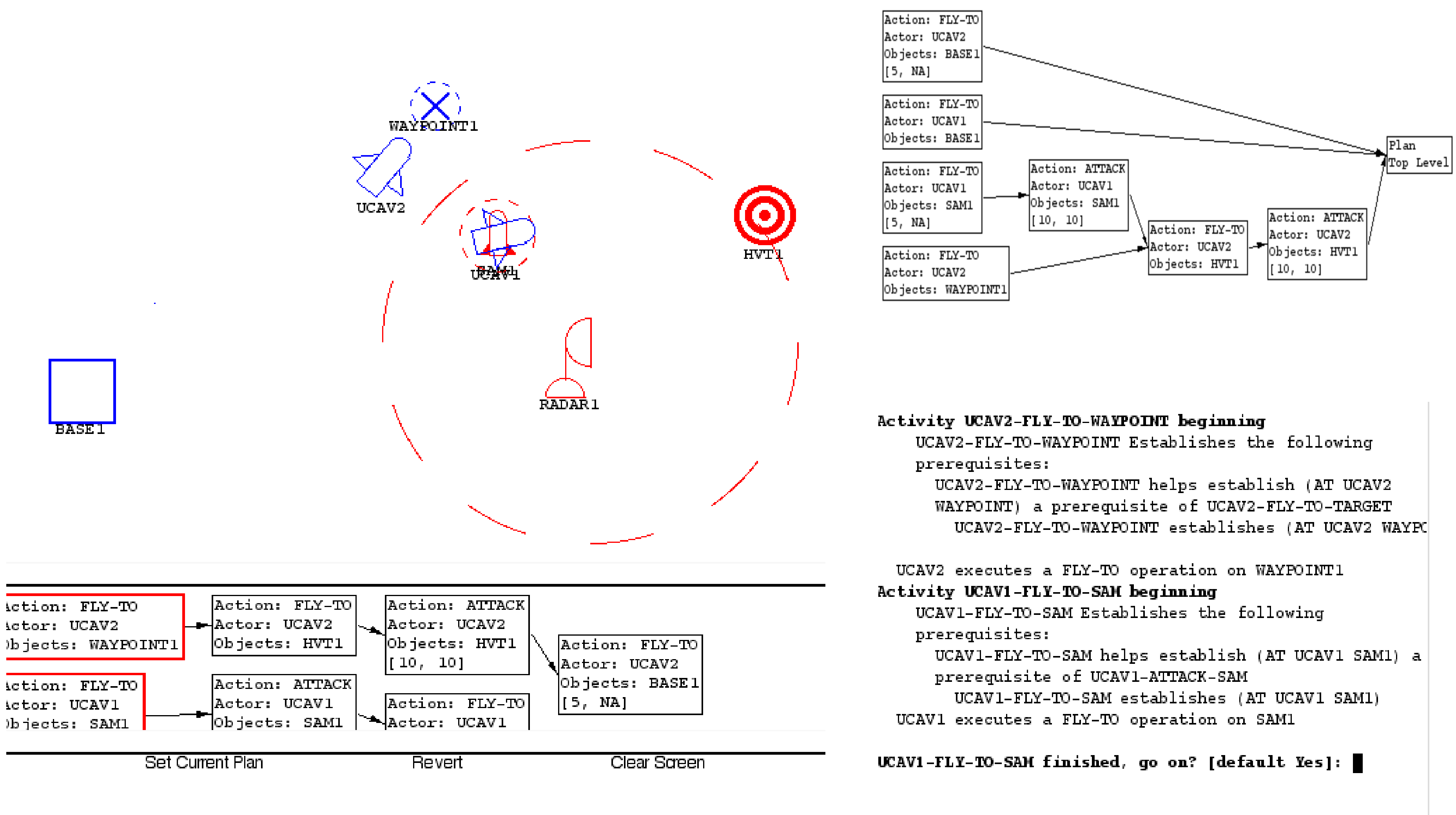


- **Complete Plan**
  - A plan is ***complete*** IFF every precondition of every activity is achieved.
  - An activity's precondition is achieved IFF:
    - The precondition is the effect of a preceding activity (support), and
    - No intervening step conflicts with the precondition (mutex).

- **Consistent Plan**
  - The plan is ***consistent*** IFF the temporal constraints of its activities are consistent (the associated distance graph has no negative cycles), and
  - no conflicting (mutex) activities can co-occur.

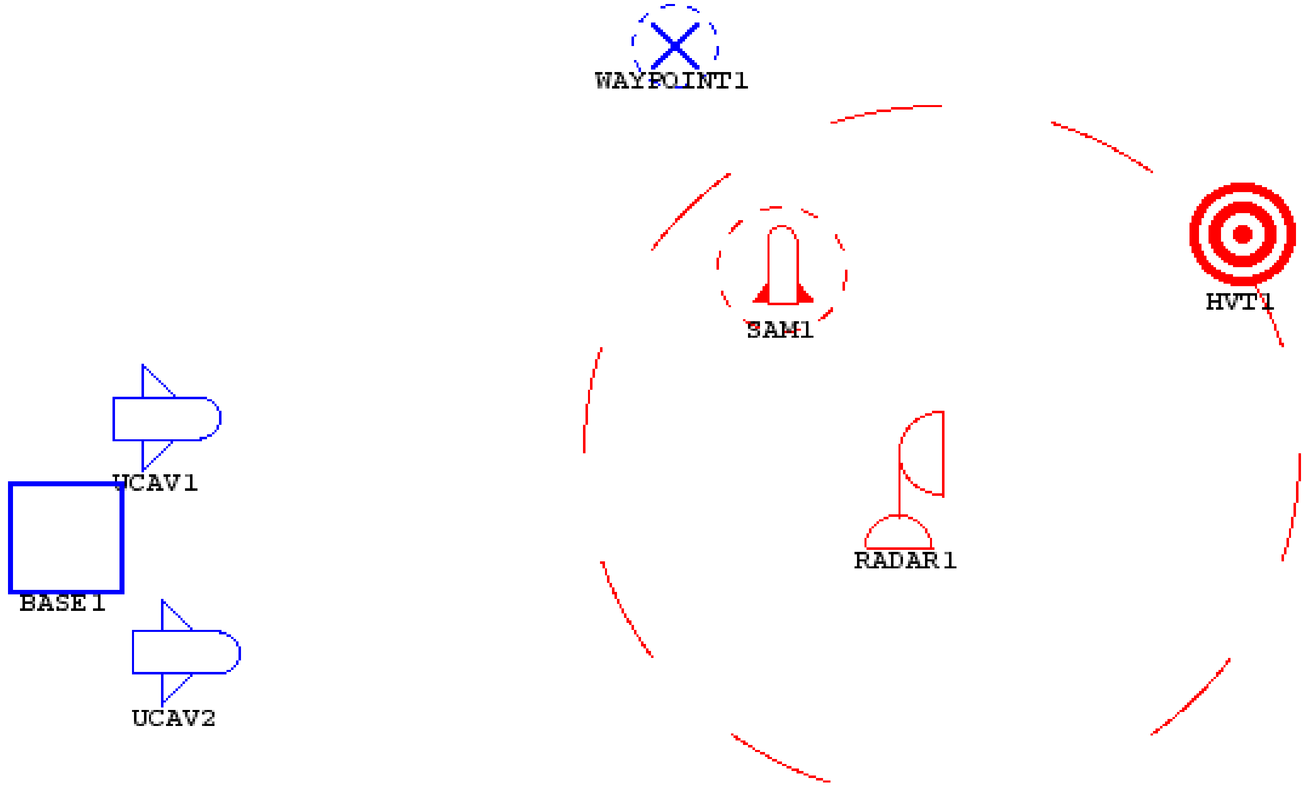
- **Minimal Plan**
  - The plan is ***minimal*** IFF every constraint serves a purpose, *i.e.,*
    - If we remove any temporal or symbolic constraint from a minimal plan, the new plan is not equivalent to the original plan

Plan-based HCI Proof of Concept: Coaching through Coordinated Views

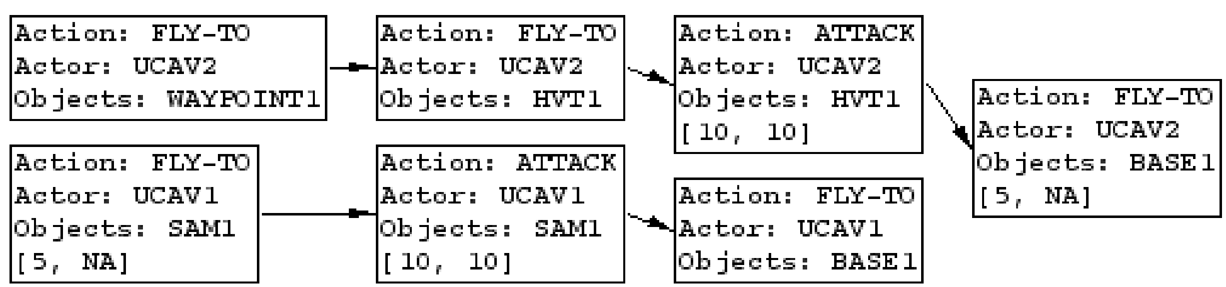


(Courtesy of Howard Shrobe, Principal Research Scientist, MIT CSAIL. Used with permission.)

Plan & Geography View

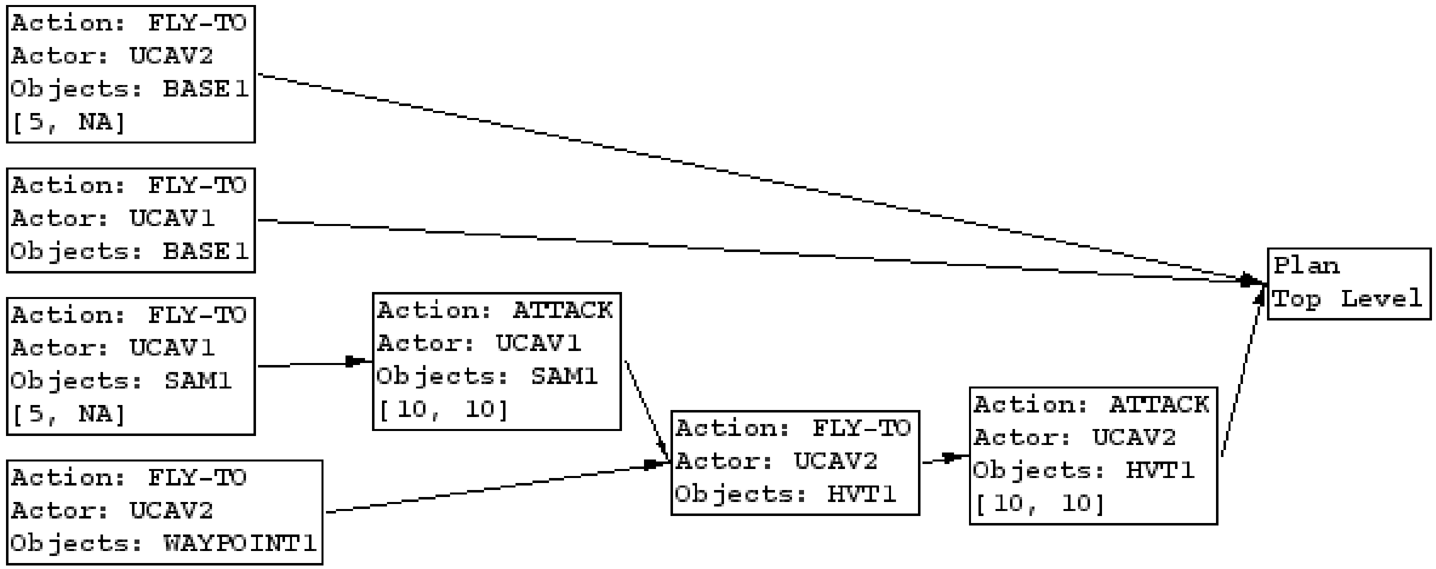


Sequencing:



Causal View

Causality



Explanation

```
UCAV2-ATTACK-TARGET Has the following prerequisites:
  UCAV2-ATTACK-TARGET requires (AT UCAV2 HVT1)
  This was established by
    Activity UCAV2-FLY-TO-TARGET achieving (AT UCAV2 HV

UCAV2-ATTACK-TARGET Establishes the following
prerequisites:
  UCAV2-ATTACK-TARGET helps establish (DESTROYED HVT1)
  prerequisite of NEW
    UCAV2-ATTACK-TARGET establishes (DESTROYED HVT1)
```

Model-based Programming of Robust Robotic Networks

- Long-lived systems achieve robustness by coordinating a complex network of internal devices.
- Programmers make a myriad of mistakes when programming these autonomic processes.
- Model-based programming simplifies this task by elevating the programmer to the level of a coach:
  - Makes hidden states directly accessible to the programmer.
  - Automatically mapping between states, observables and control variables.
- Model-based executives reasoning quickly and extensively by exploiting conflicts.
- Mission-level executives combine activity planning, logical decision making and control into a single hybrid decision problem.