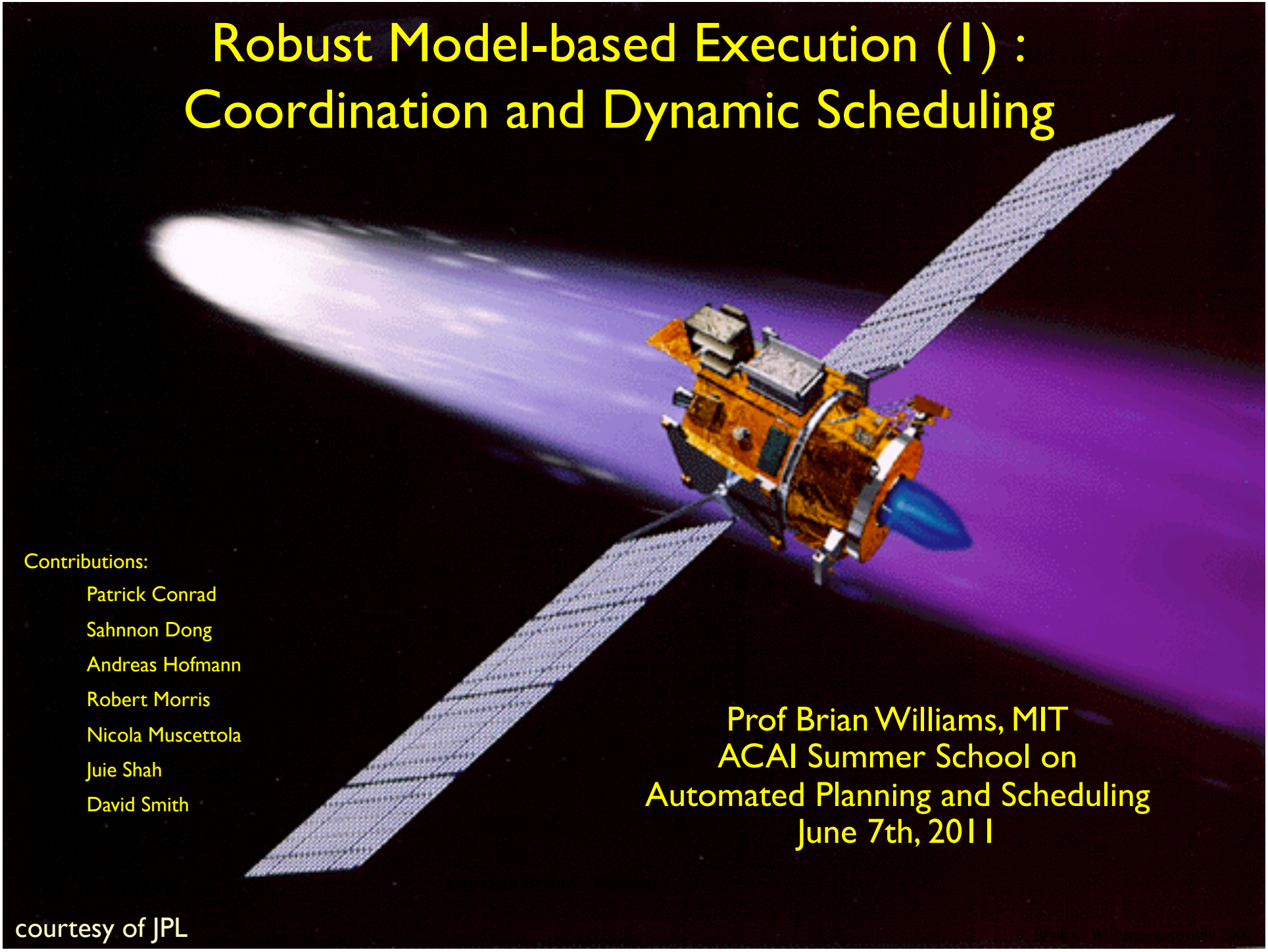
Robust Model-based Execution (1) : Coordination and Dynamic Scheduling



Contributions:

- Patrick Conrad
- Sahnnon Dong
- Andreas Hofmann
- Robert Morris
- Nicola Muscettola
- Juie Shah
- David Smith

Prof Brian Williams, MIT
ACAI Summer School on
Automated Planning and Scheduling
June 7th, 2011

courtesy of JPL

Readings

- Google “MIT OCW 16.412 Cognitive Robotics”
- mers.csail.mit.edu, click “Publications”
- Dechter, R., I. Meiri, J. Pearl, “Temporal Constraint Networks,” Artificial Intelligence, 49, pp. 61-95,1991.
- Muscettola, N., P. Morris and I. Tsamardinos, “Reformulating Temporal Plans for Efficient Execution.” Intl Conf. on Knowledge Representation and Reasoning (KRR), 1998.
- Shah, J.; Stedl, J.;Williams, B.; and Robertson, P. 2007. A Fast Incremental Algorithm for Maintaining Dispatchability of Partially Controllable Plans.
- *P. Morris, N. Muscettola and T Vidal,” Dynamic Control of Plans with Temporal Uncertainty,” in Proc. Int. Joint Conf. on AI, 2001.*
- . Léauté and B. C. Williams, "Coordinating Agile Systems Through the Model-based Execution of Temporal Plans," Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), Pittsburgh, PA, July 2005, pp. 114-120.
- A. G. Hofmann and B. C. Williams, "Exploiting Spatial and Temporal Flexiblity for Plan Execution of Hybrid, Under- Actuated Systems," Proceedings of the 21st National Conference on Artficial Intelligence, Boston, MA, July 2006, pp. 948-955.
- Tsamardinos, I.; Pollack, M.; and Ganchev, P. 2001. Flexible dispatch of disjunctive plans. In 6th European Conference on Planning, 417–422.
- P. Conrad, J. Shah and B. Williams, “Flexible Execution of Plans with Choice,” Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 09), Thessaloniki, Greece, September 2009.
- Julie Shah, Patrick Conrad, and Brian C. Williams, "Fast Distributed Multi-agent Plan Execution with Dynamic Task Assignment and Scheduling," Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 09), Thessaloniki, Greece, September 2009.

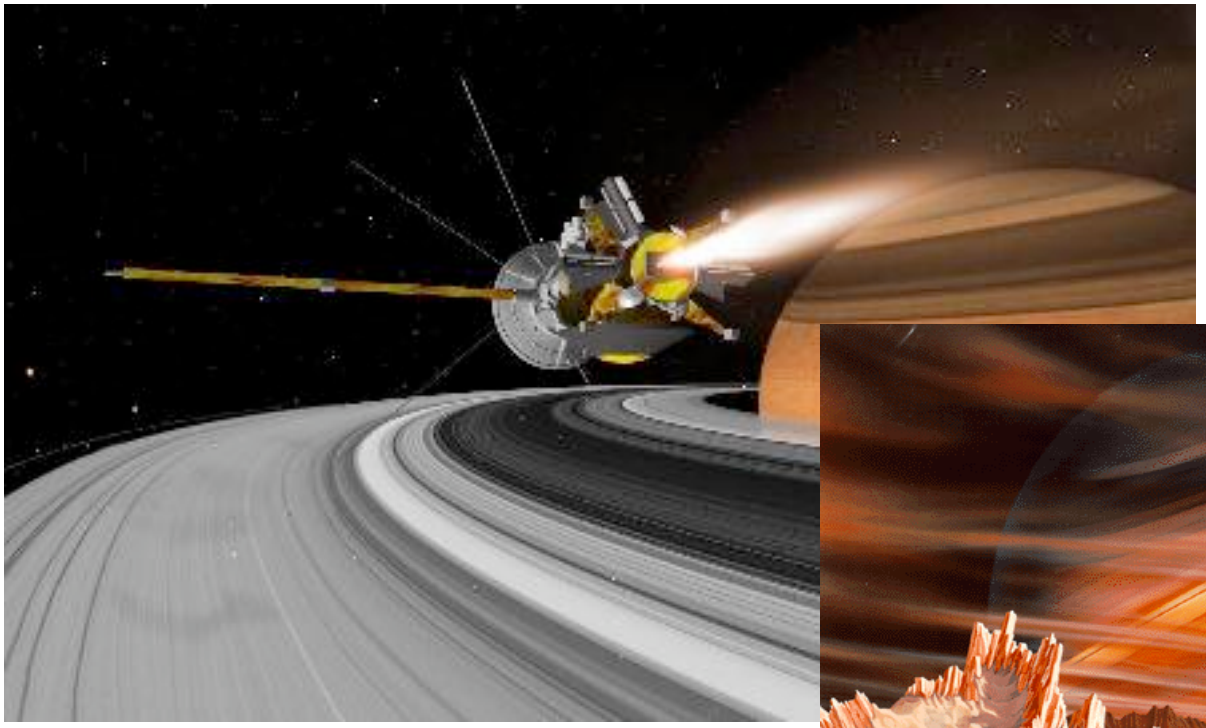
Robust, Model-based Execution (1): Coordination and Dynamic Scheduling

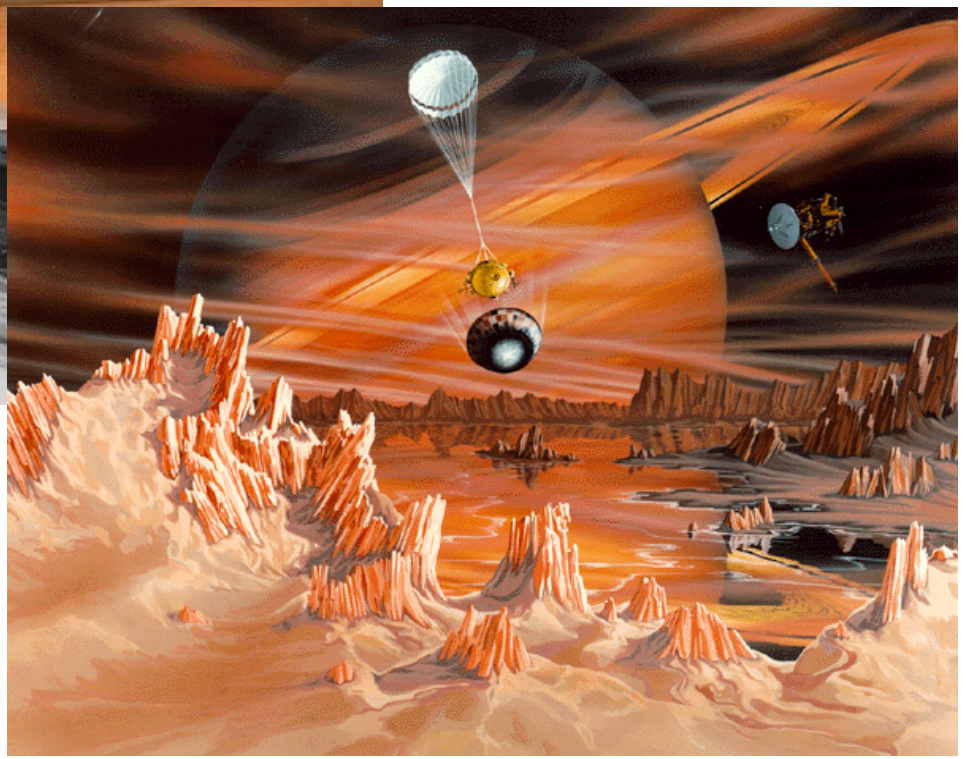
Model-based Embedded & Robotic Systems

- Robust, model-based execution of time critical tasks.
  - Case Study: Remote Agent.
  - Case Study: Personal Transport System.
- Task coordination through dynamic scheduling.
- Task coordination for under-actuated systems.
- Task coordination for multi-robot systems.

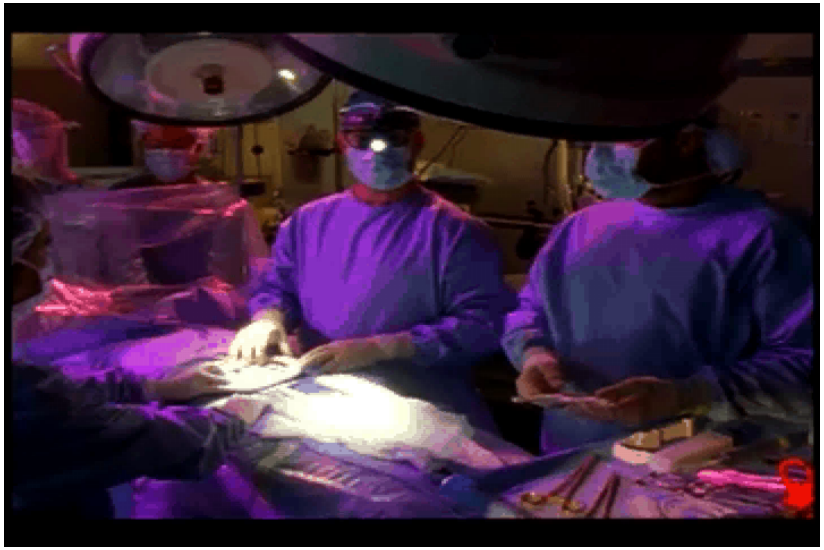
3

Time Critical Tasks





Time Critical Tasks



An effective Scrub Nurse:

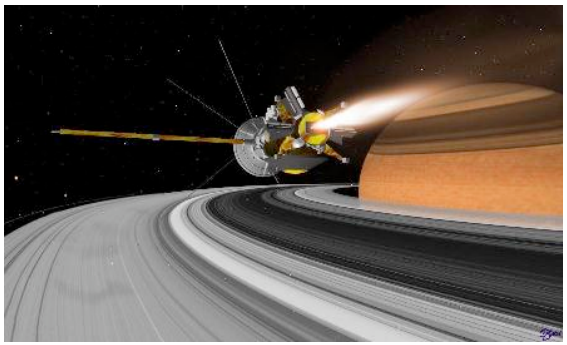
- works hand-to-hand, face-to-face with surgeon,
- assesses and anticipates needs of surgeon,
- provides assistance and tools in order of need,
- responds quickly to changing circumstances,
- responds quickly to surgeon's cues and requests.

[Shah and Williams]

Model-based Execution

The development of autonomous systems that robustly perform complex tasks.

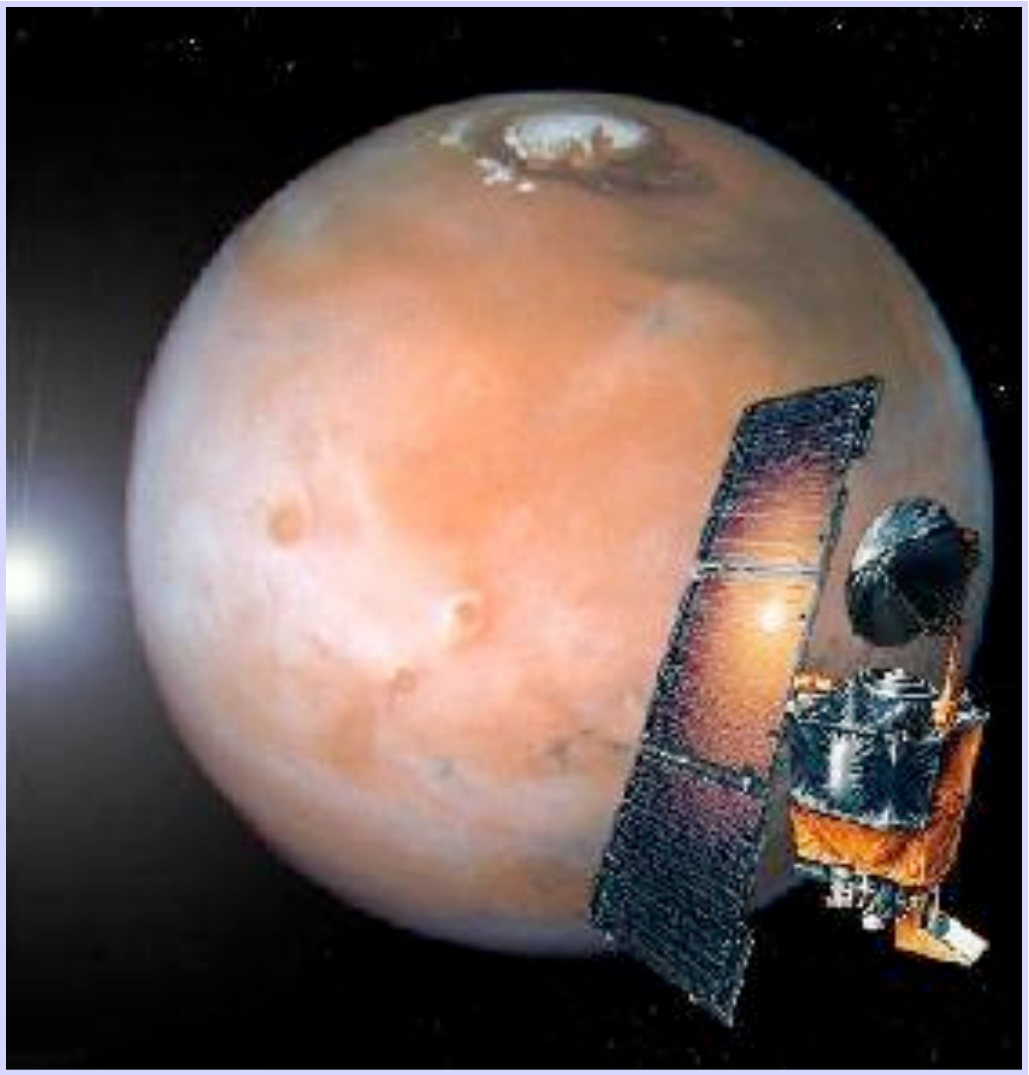
- Goal-directed: Tasks described qualitatively in terms of time-evolved goals.
- Real-time Decisions: Tasks executed using real-time decision making algorithms, based on observations.
- Model-based: Operates on heterogeneous models of the robot, user and environment.

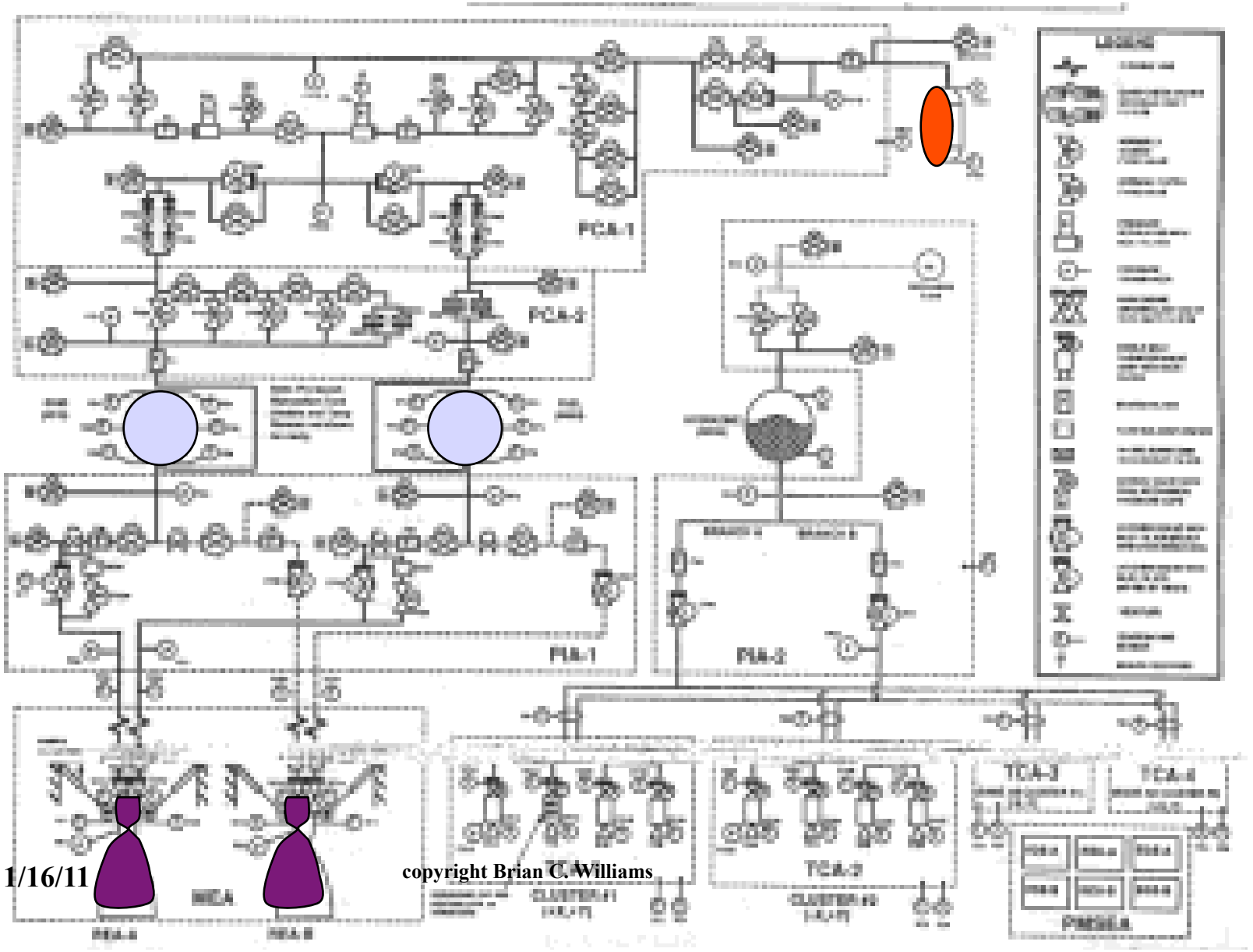


Remote Agent Experiment on Deep Space One

courtesy of JPL

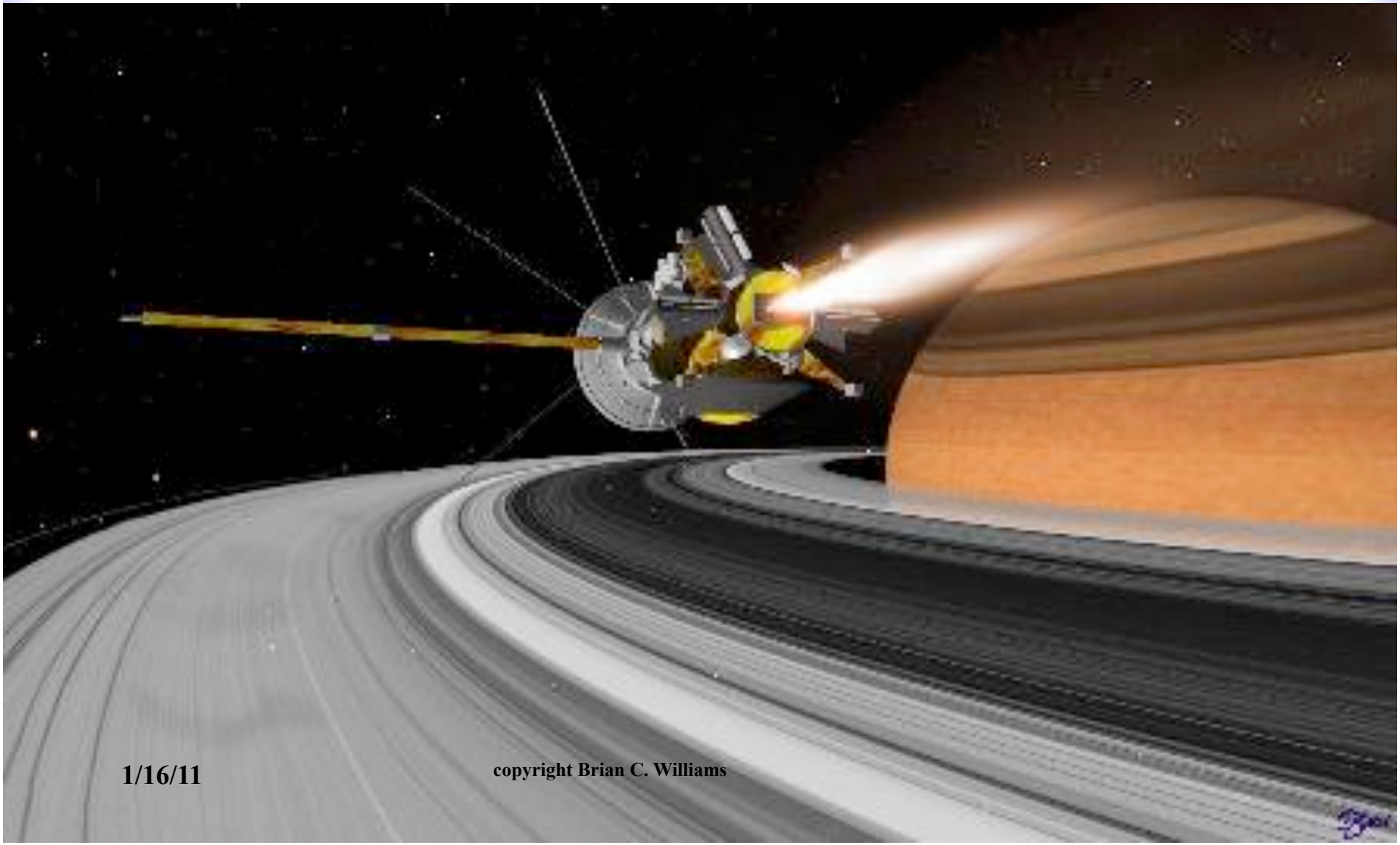
Loss of Mars Observer, early 90's



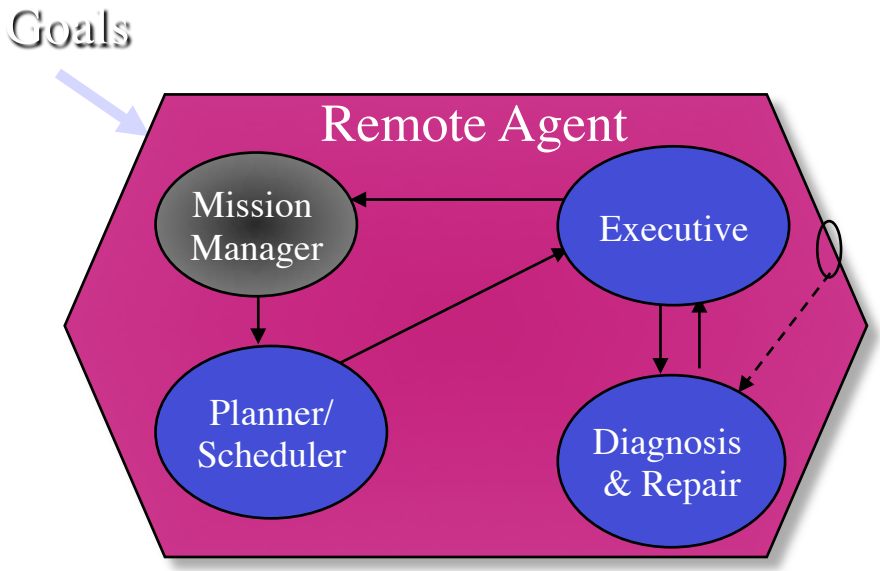


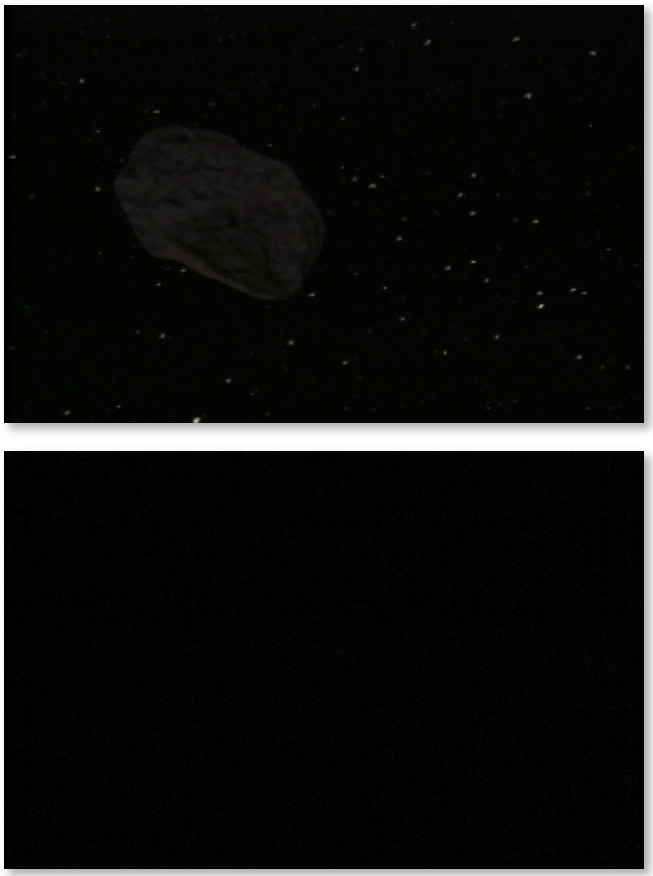
Autonomy Demonstration on Simulated Cassini Probe

Model-based Embedded & Robotic Systems



Remote Agent on Deep Space One

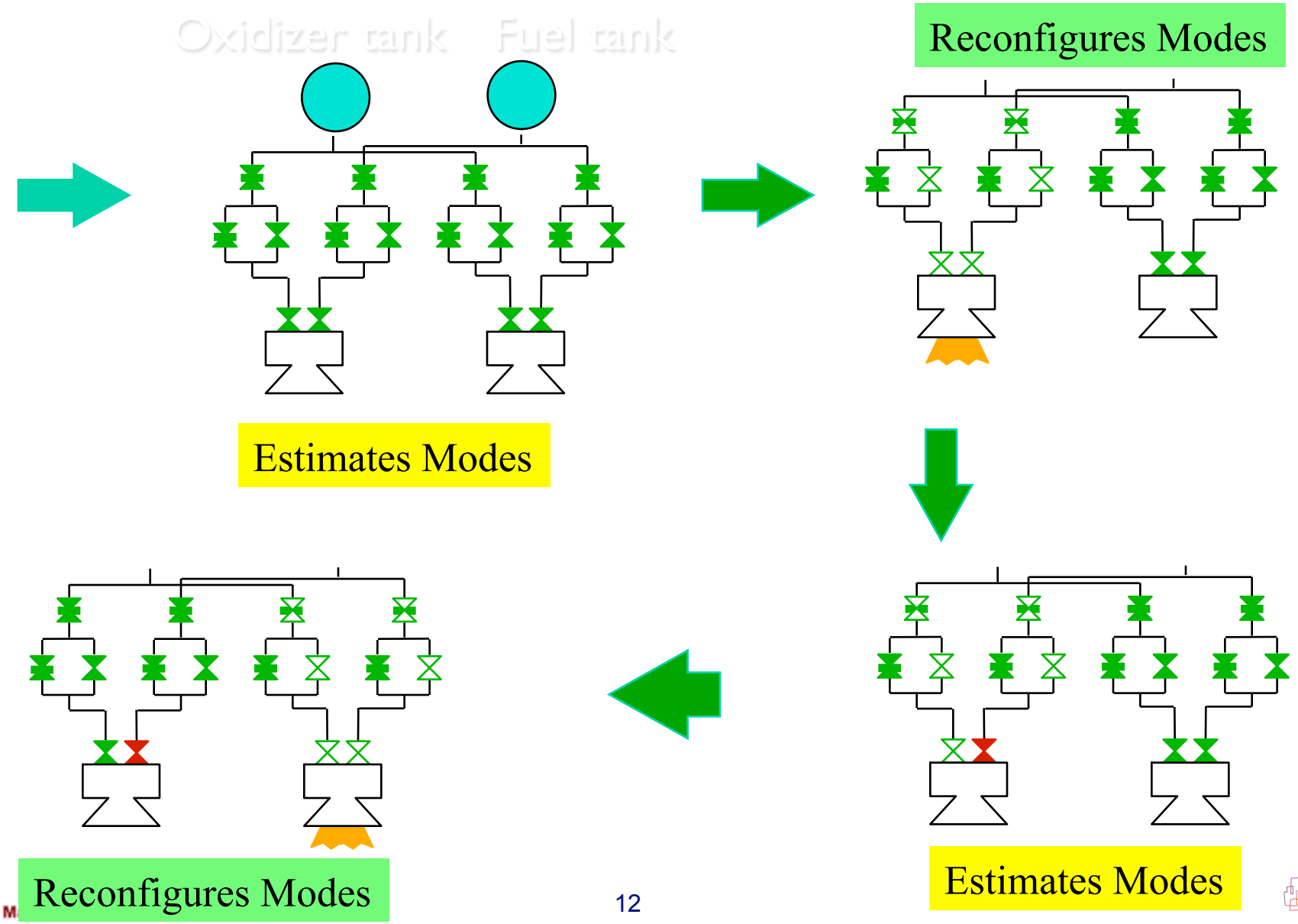




1. Commanded by giving goals
2. Reasoned from commonsense models
3. Closed loop on goals

[Williams & Nayak, AAAI 95; Muscettola et al, AIJ 00]

Goal: Set engine to thrusting for 1hr...

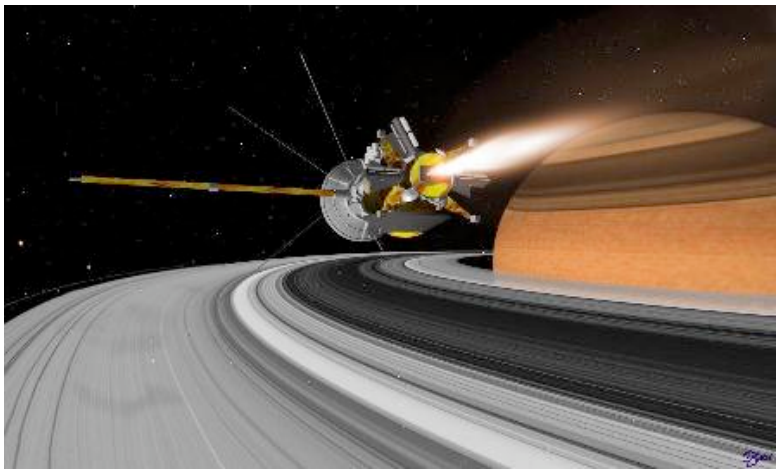


Approach: Model-based Programming and Execution

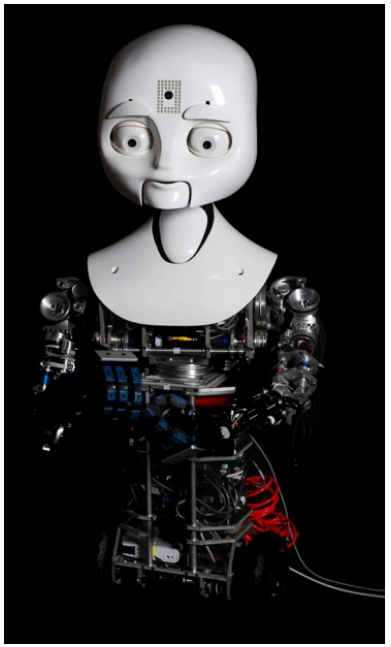
- An embedded programming language elevated to operations on hidden state, and
- A language executive that achieves robustness by reasoning over constraint-based models.

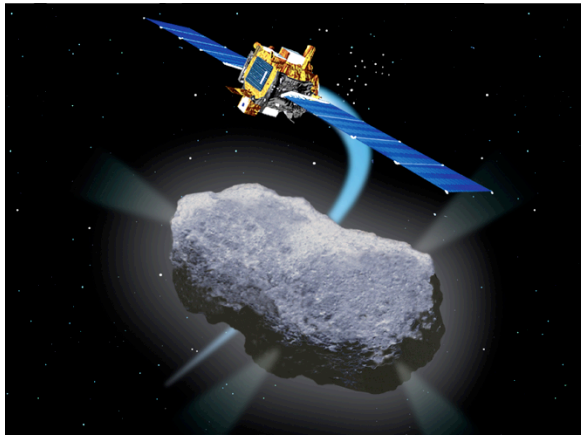
Today: Coordination and dynamic scheduling.

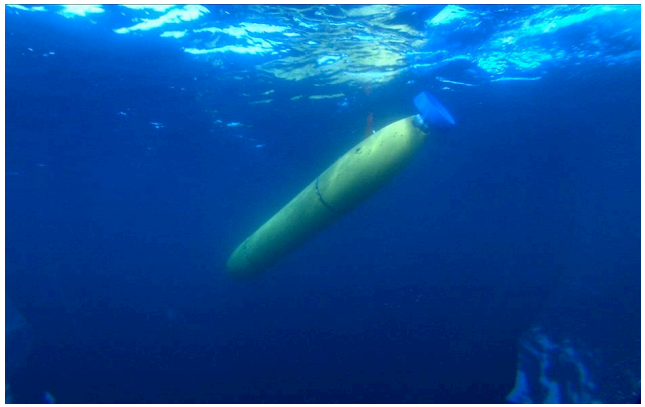
Wednesday: Model-based programming with hidden state.

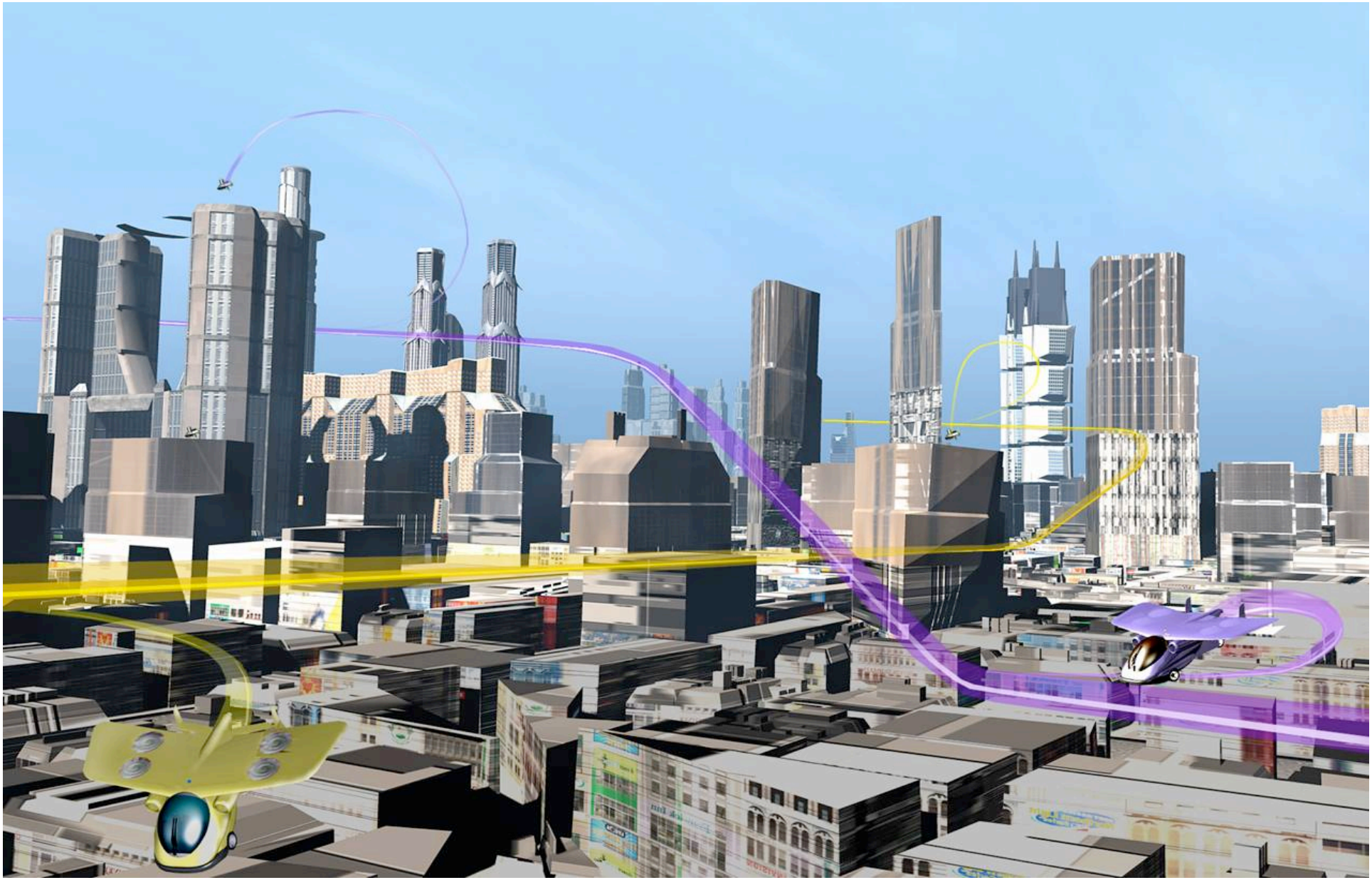


Model-based Execution has been applied to a diverse set of robotic systems.

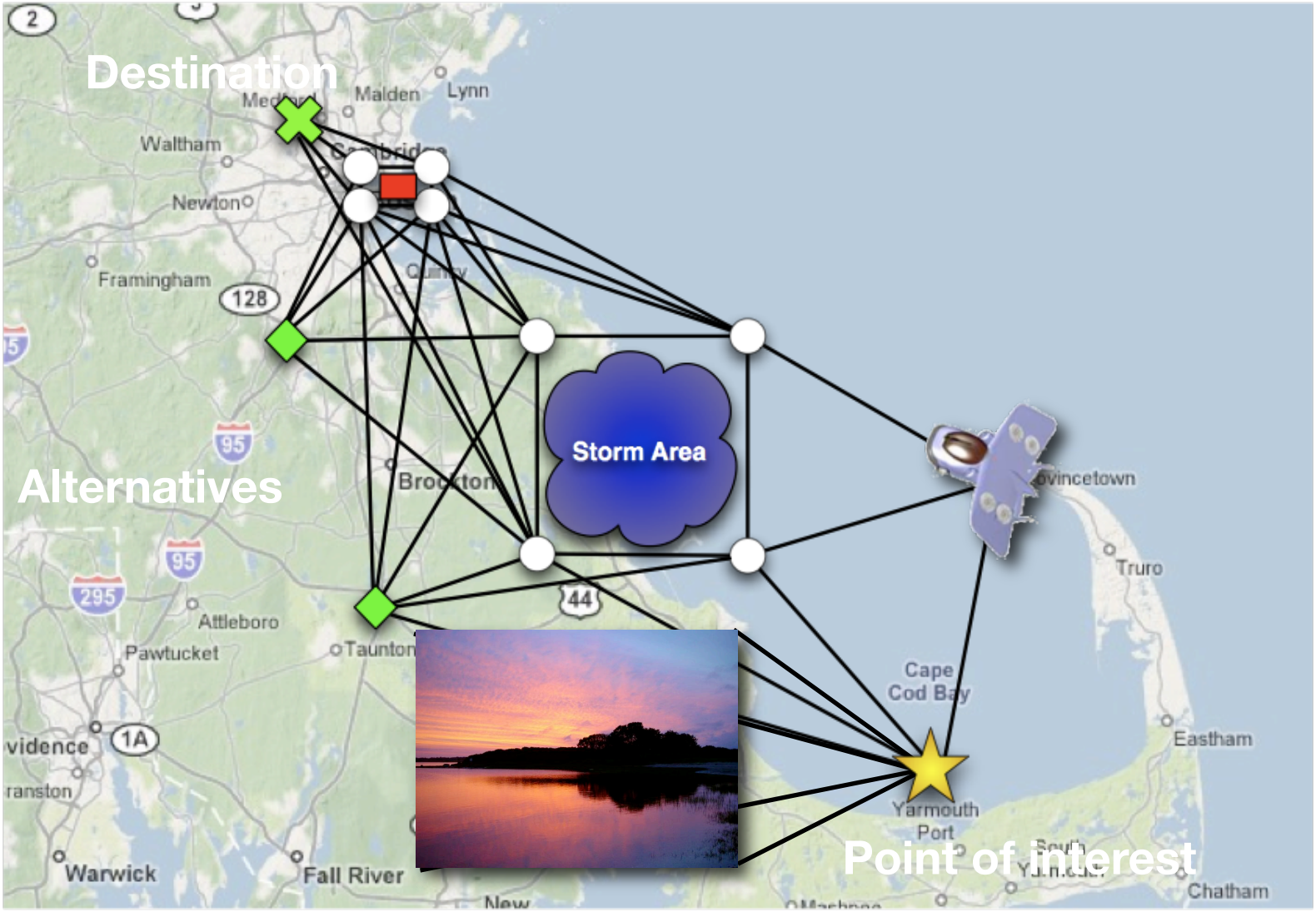




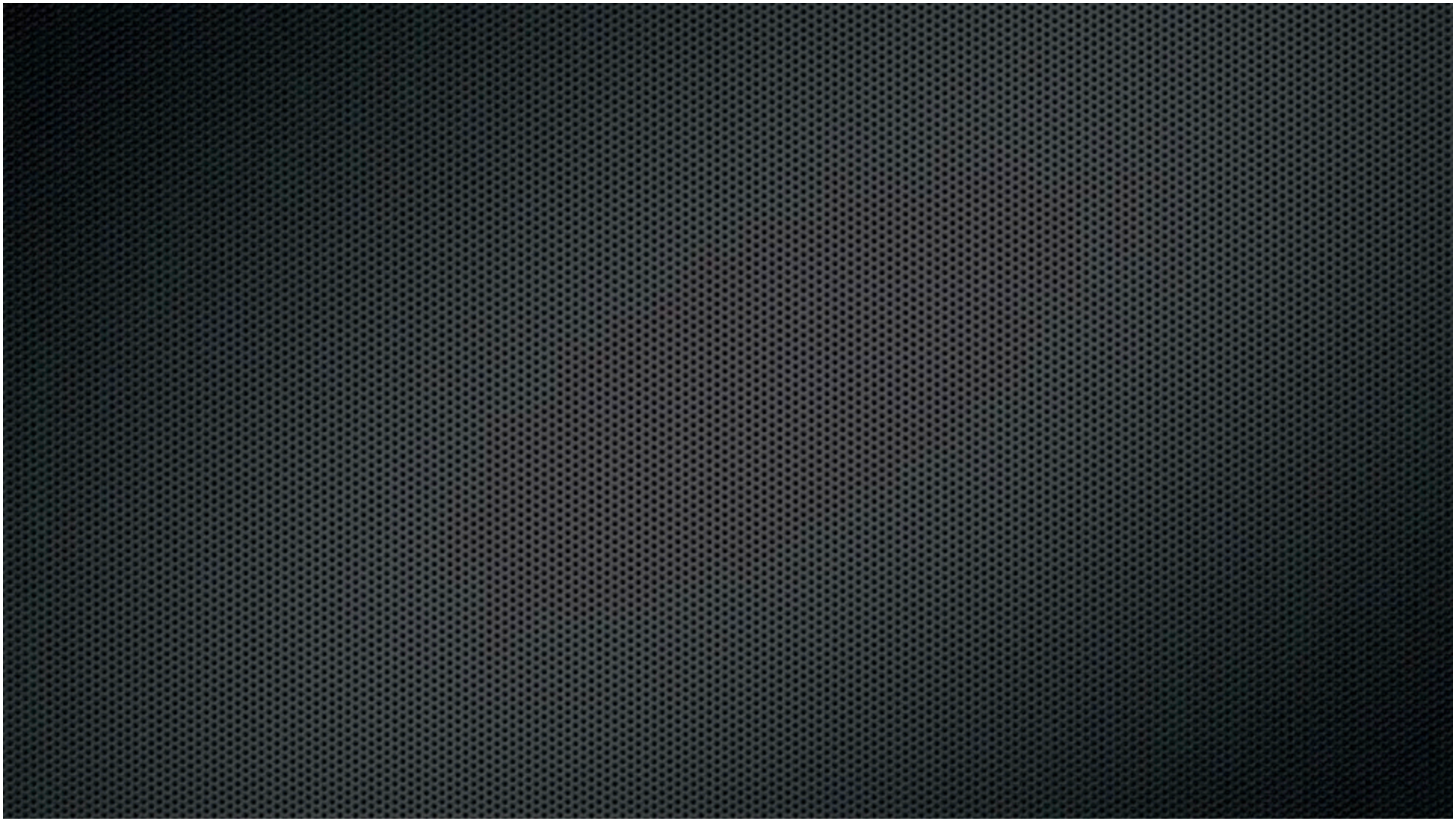




A Robot Air Taxi Driver



X Plane Simulation of Personal Transportation System (PTS)



Movie M?

Robust, Model-based Execution (1): Coordination and Dynamic Scheduling

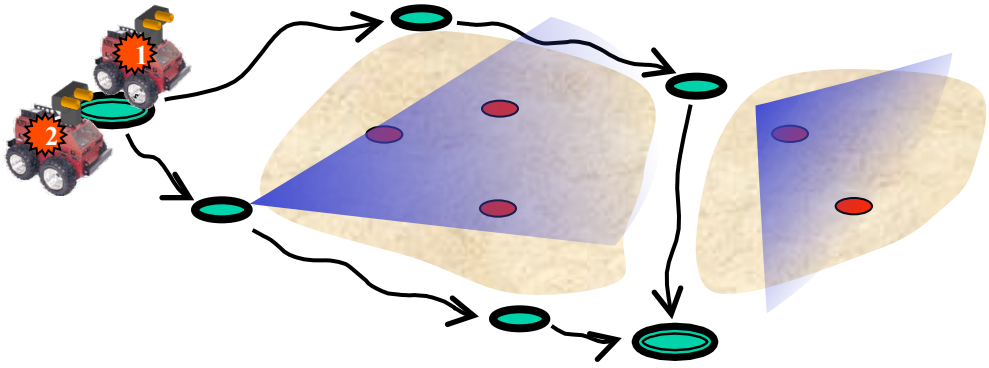
Model-based Embedded & Robotic Systems

- Robust, model-based execution of time critical tasks.
- Task coordination through dynamic scheduling.
  - Representing plans and temporal relationships.
  - Scheduling based on decomposability.
  - Dynamic scheduling.
  - Dynamic scheduling with models of uncertainty.
- Task coordination for under-actuated systems.
- Task coordination for multi-robot systems.

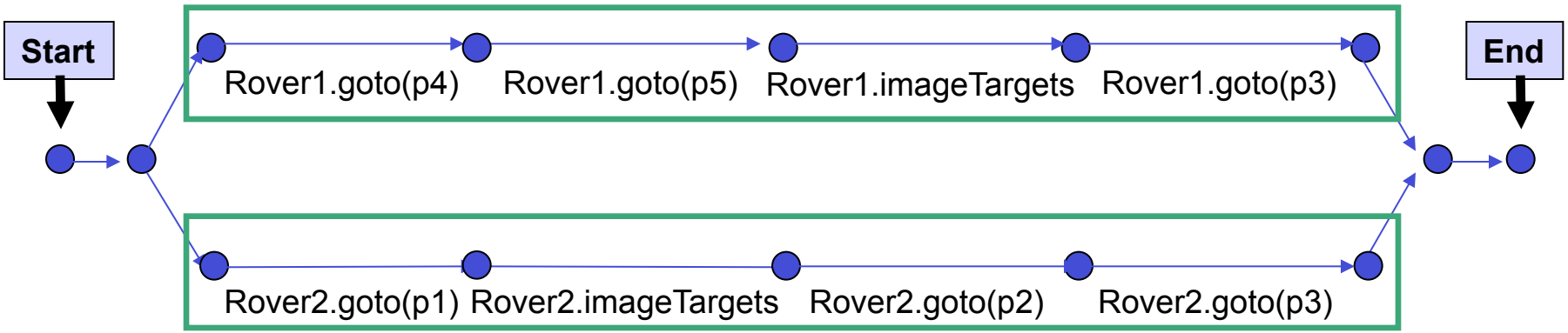
Robust Program and Plan Execution

Model-based Embedded & Robotic Systems

```
imageScienceTargets(Rover1, Rover2)
{Parallel
  {Sequence
    [5,10] Rover1.goto(p4);
    [5,10] Rover1.goto(p5);
    [2,5] Rover1.imageTargets();
    [5,10] Rover1.goto(p3);
  },
  {Sequence
    [5,10] Rover2.goto(p1);
    [5,10]Rover2.imageTargets();
    [2,5] Rover2.goto(p2);
    [5,10] Rover2.goto(p3);
  }
}
```

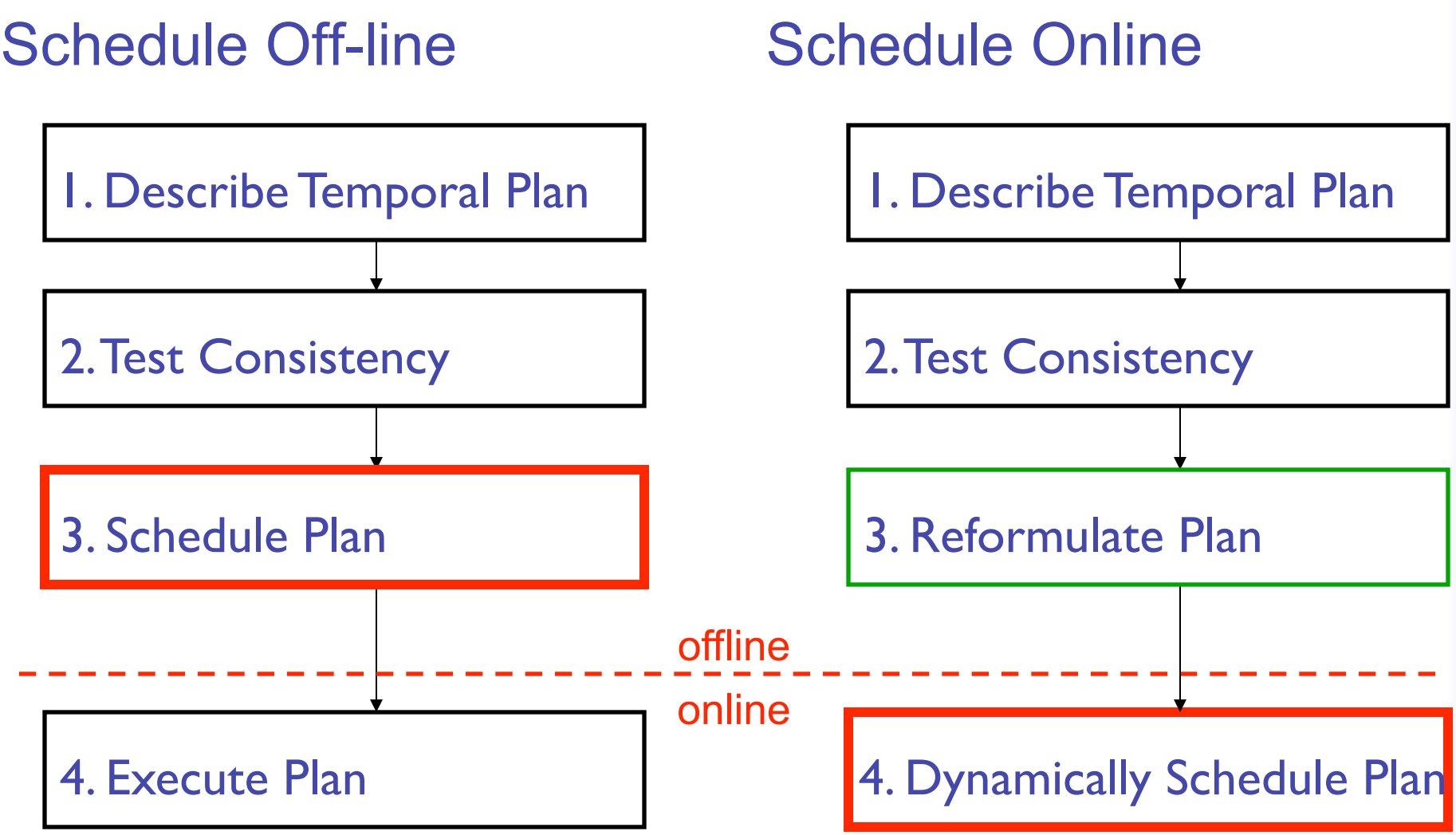


in RMPL [williams et al]



Agents adapt to temporal disturbances in a coordinated manner by scheduling the start of activities on the fly.

To Execute a Temporal Plan



Outline: To Execute a Temporal Plan

Part I: Schedule Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan

offline

online

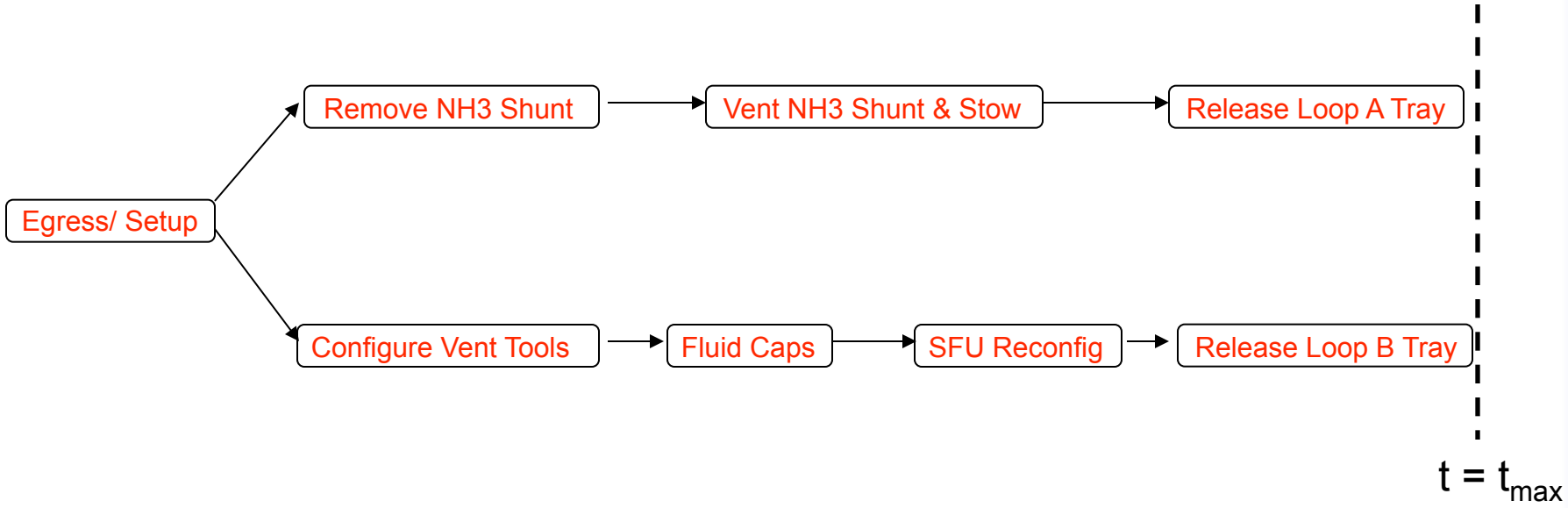
4. Execute Plan

Part II: Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan
4. Dynamically Execute Plan

Describe Temporal Plan

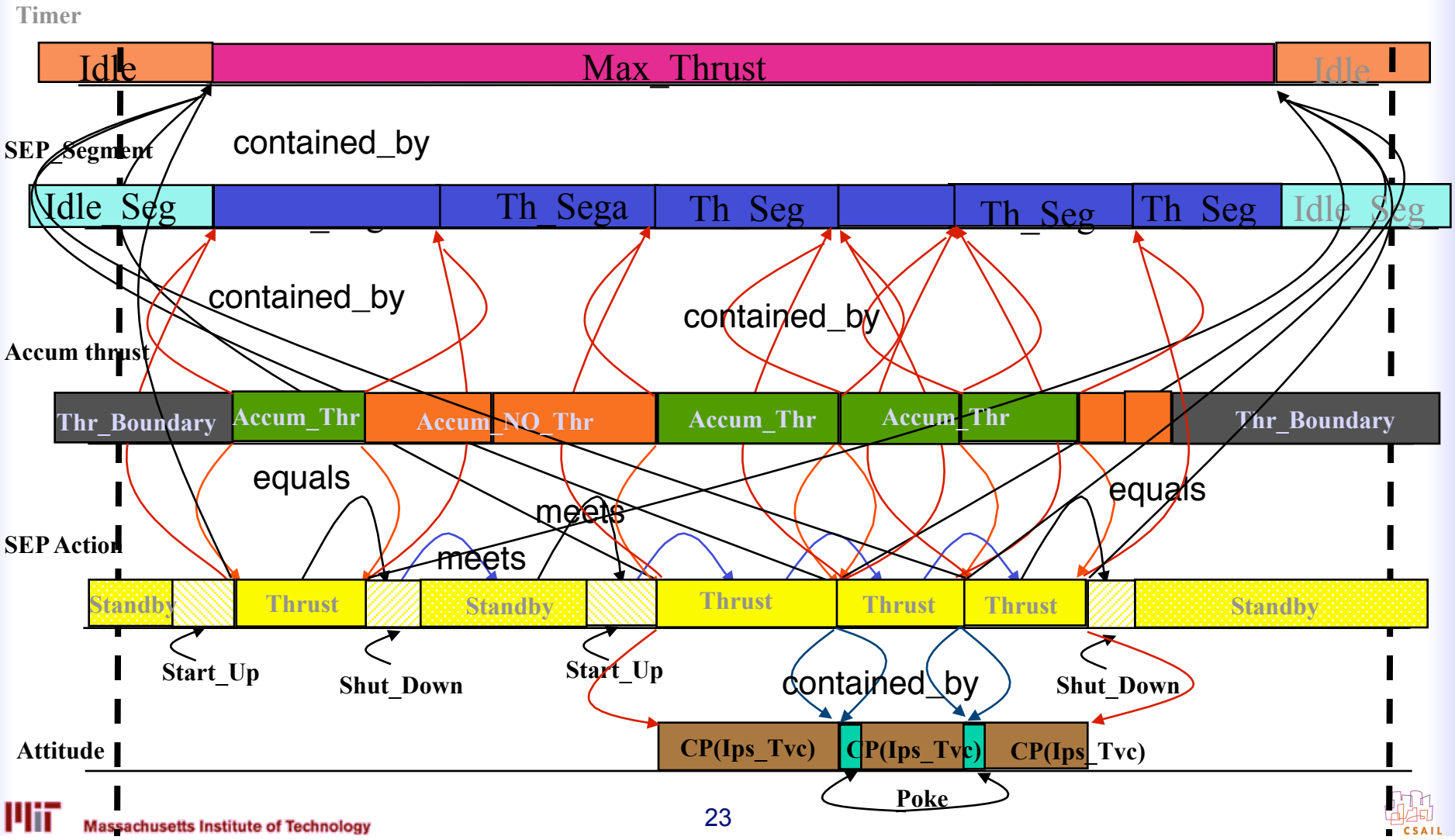
- Activities to perform
- Relationships among activities



Time Lines

Model-based Embedded & Robotic Systems

Example: Deep Space One Remote Agent Experiment

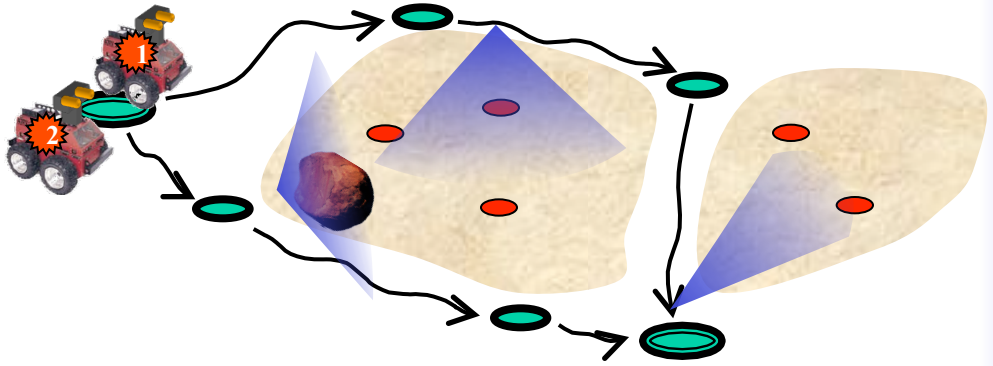


Temporal Plan Networks

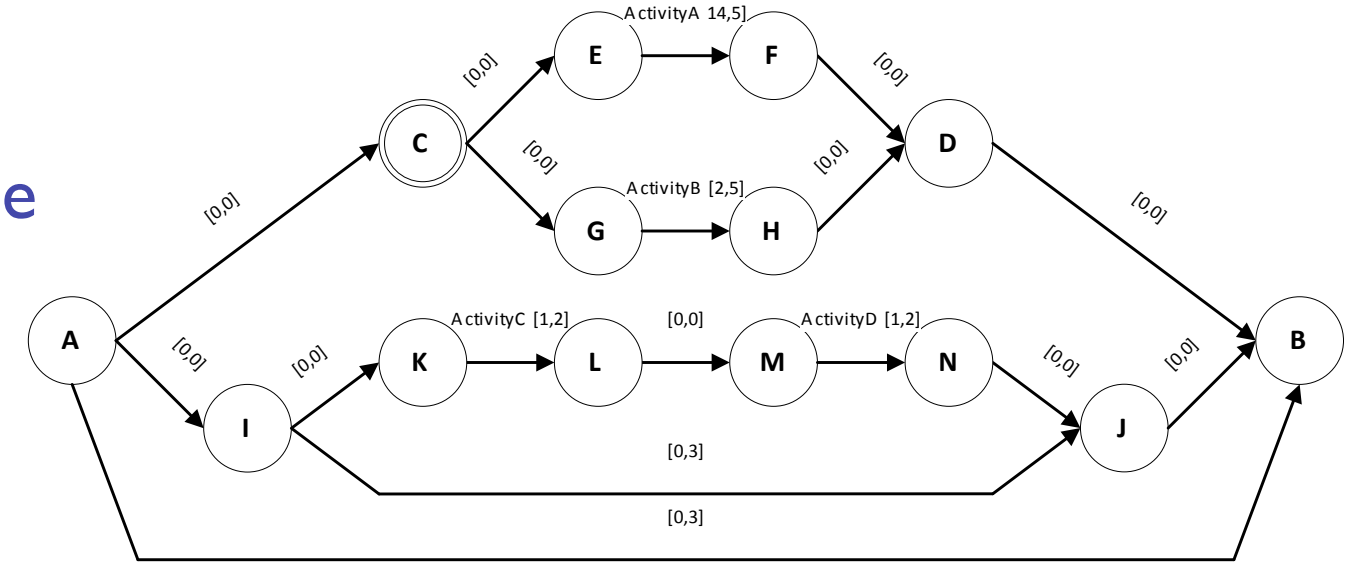
Nested Compositions:

(non-deterministic programs)

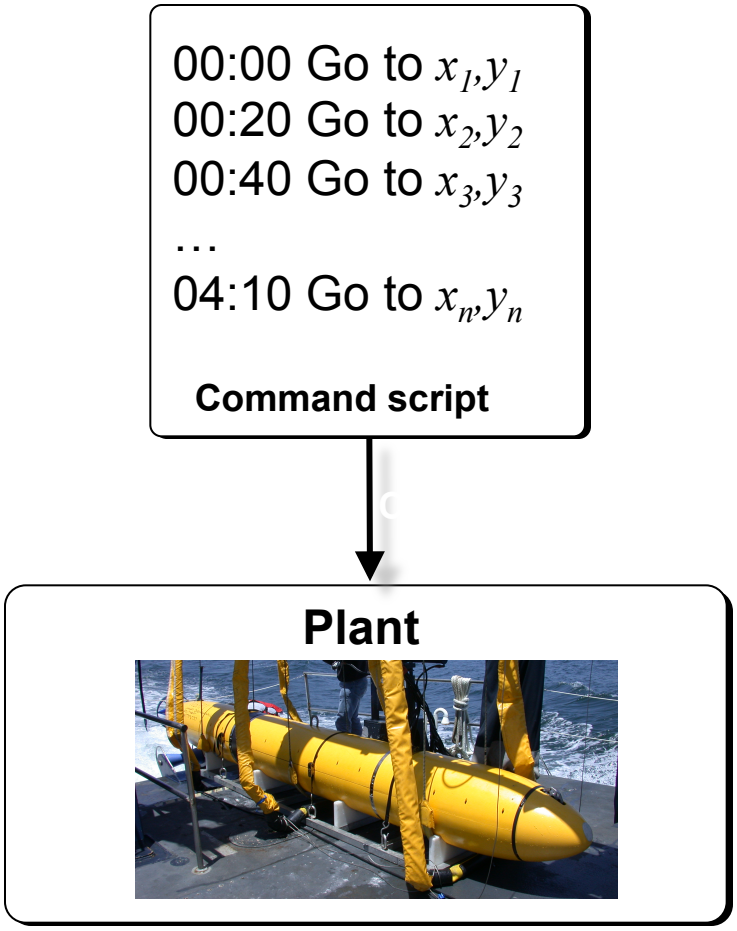
- Activity
- Sequence
- Parallel
- Choice
- With Time



[Kim, Williams, Abramson, IJCAI01]

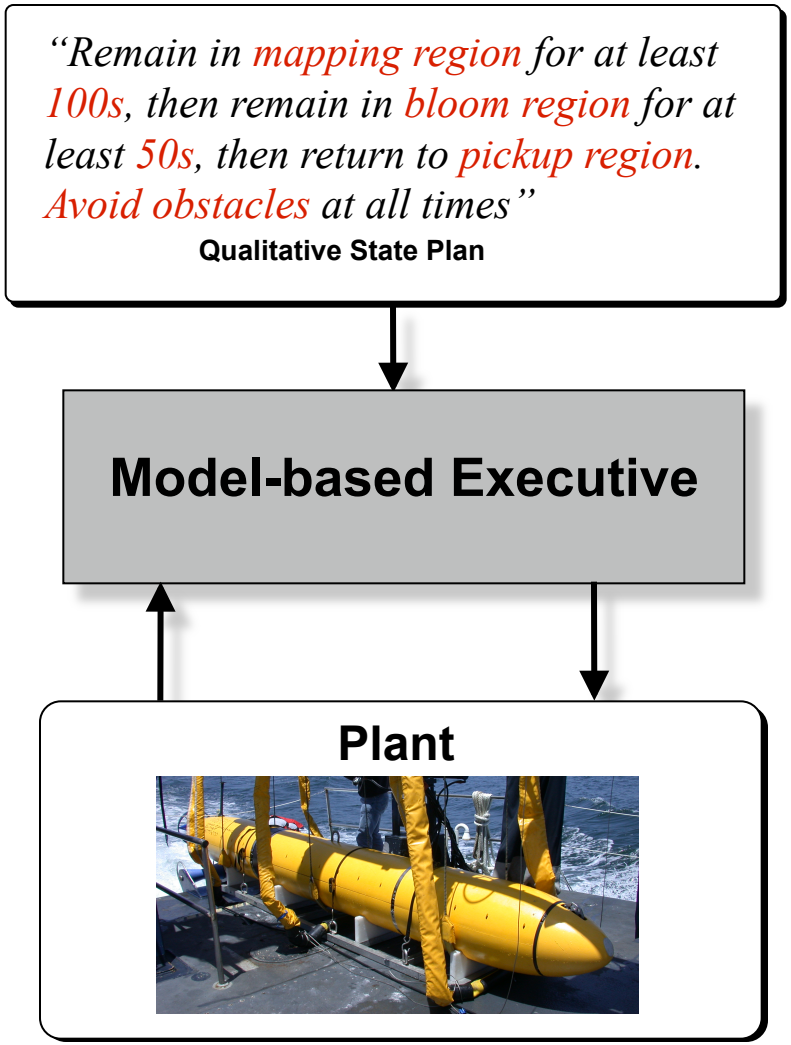


Qualitative State Plans



Leaute & Williams, AAAI 05

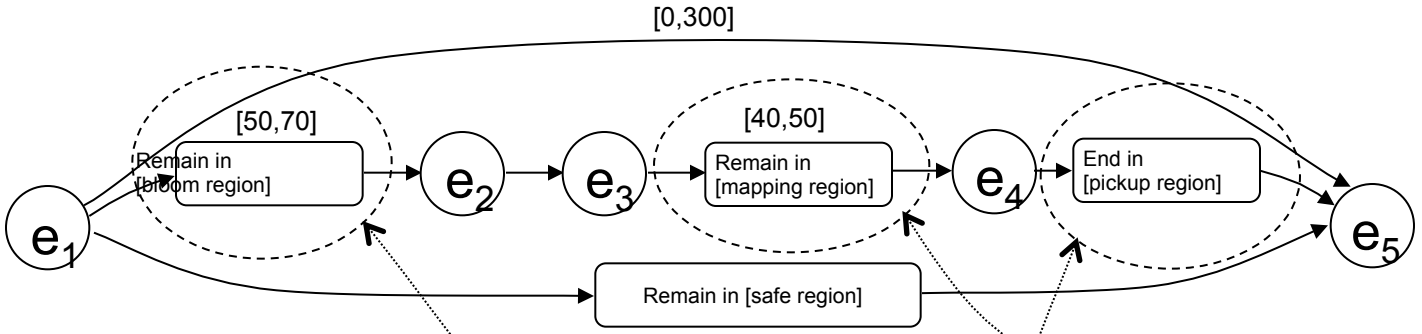
Qualitative State Plans



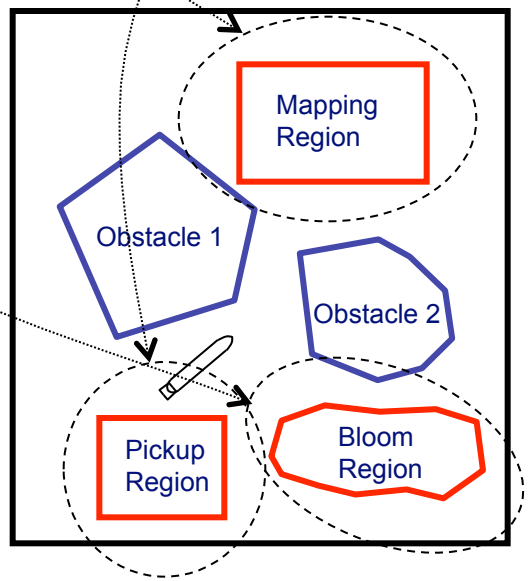
Leaute & Williams, AAAI 05

Qualitative State Plans

A temporal plan whose activities impose constraints on system state.



"Remain in bloom region for between 50 and 70 seconds. Afterwards, remain in mapping region for between 40s and 50s. End in the pickup region. Avoid obstacles at all times. Complete the mission within 300s"

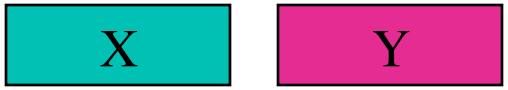
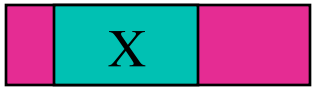
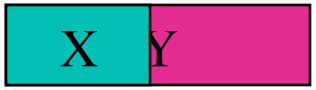
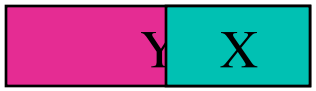
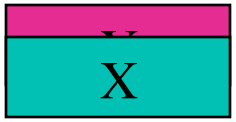


Approach Frame as Model-Predictive Control using Mixed Logic or Integer / Linear Programming

Leaute & Williams, AAAI 05

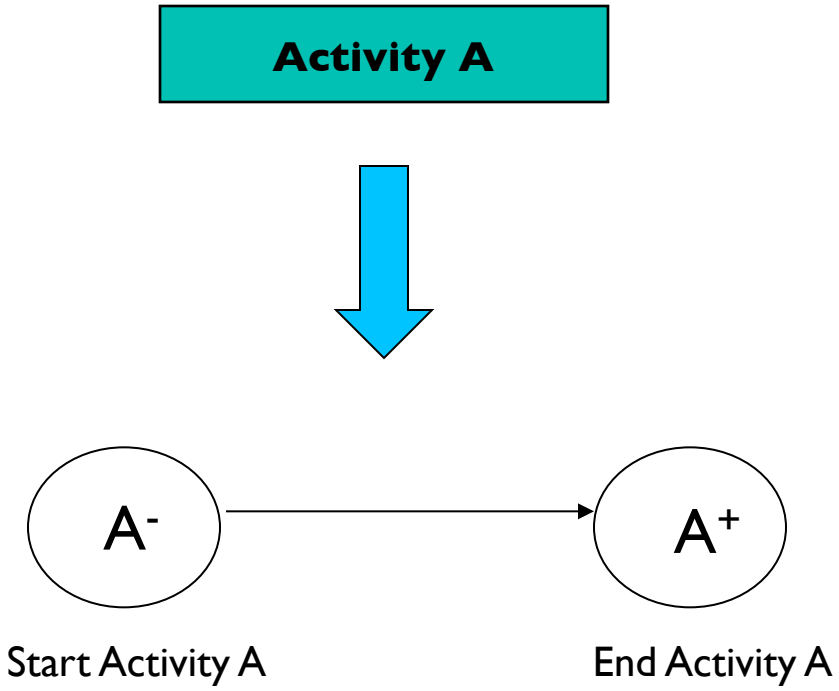
Temporal Relationships

Qualitative Temporal Relations [Allen 83]

X before Y		Y after X
X meets Y		Y met-by X
X overlaps Y		Y overlapped-by X
X during Y		Y contains X
X starts Y		Y started-by X
X finishes Y		Y finished-by X
X equals Y		Y equals X
X disjoint Y		

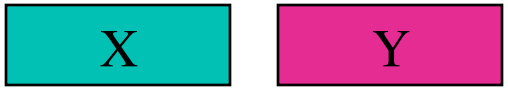
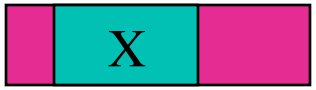
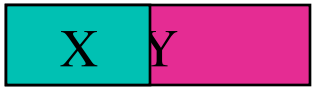
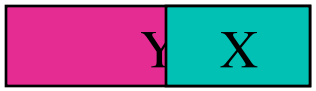
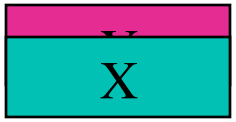
Temporal Relationships

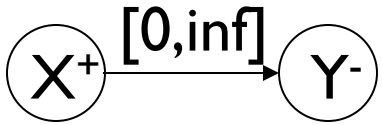
Simplify by reducing interval relations to relations on timepoints.

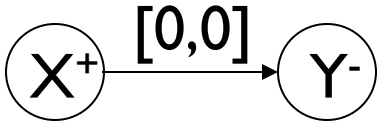


Temporal Relationships

Qualitative Temporal Relationships as timepoint inequalities

X before Y		$X^+ < Y^-$
X meets Y		$X^+ = Y^-$
X overlaps Y		$Y^- < X^+$ and $X^- < Y^+$
X during Y		$Y^- < X^-$ and $X^+ < Y^+$
X starts Y		$X^- = Y^-$ and $X^+ < Y^+$
X finishes Y		$X^- < Y^-$ and $X^+ = Y^+$
X equals Y		$X^- = Y^-$ and $X^+ = Y^+$
X disjoint Y		$X^+ < Y^-$ or $Y^+ < X^-$

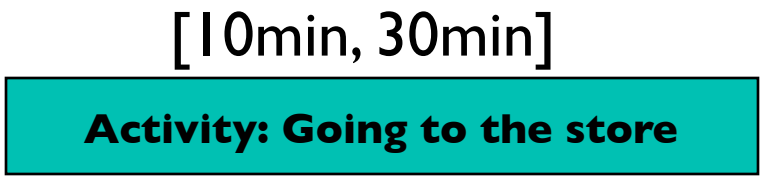




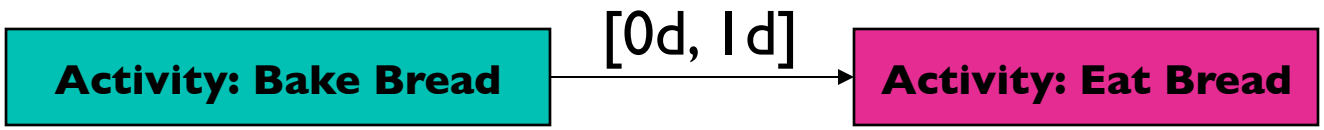
Metric Temporal Relations

Add Metric Information:

- Going to the store takes at least 10 min and at most 30 min.



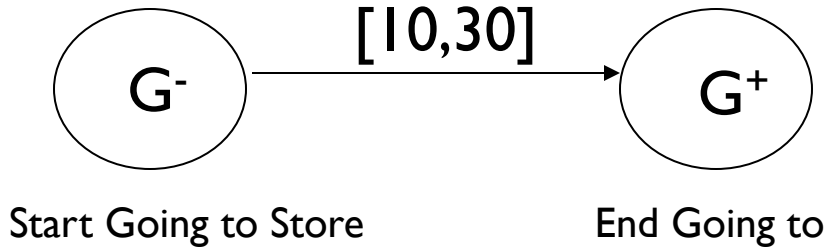
- Bread should be eaten within one day of baking.



Metric Temporal Relations

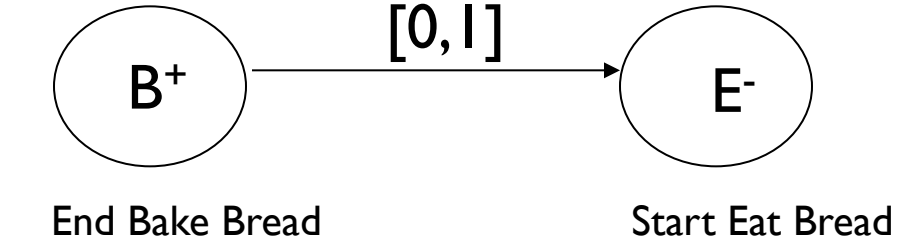
Add Metric Information: inequalities → interval constraints

- Going to the store takes at least 10 min and at most 30 min.



$10 \leq [G^{+} - G^{-}] \leq 30$

- Bread should be eaten within one day of baking.



$0 \leq [E^{-} - B^{+}] \leq 1$

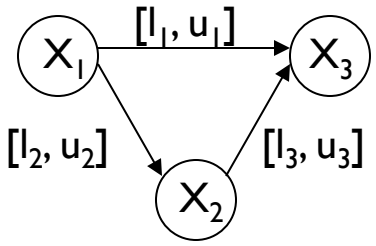
Temporal Relations Described by a STN

- Simple Temporal Network
  - variables $X_1,\ldots X_n$, representing timepoints with real-valued domains,
  - binary constraints of the form:

$$(X_k - X_i) \in [a_{ik}, b_{ik}]$$

    - called *links*.

[Dechter, Meiri, Pearl 91]



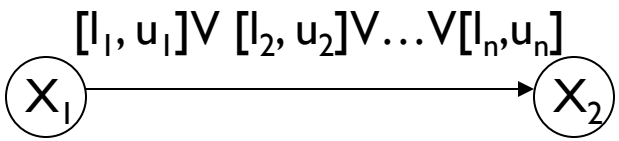
Sufficient to represent:
- all Allen relations but 1…
- simple metric constraints

Can't represent:
- Disjoint activities

Temporal Relations Described by a TCN

- Temporal Constraint Network (TCN)
  - Extends STN by allowing multiple intervals for each binary constraint (link):

[Dechter, Meiri, Pearl 91]



$$(X_k - X_i) \in P(\{[a_{ik}, b_{ik}] \mid a_{ik} \leq b_{ik}\}).$$

Supports:

•Multiple time windows for accomplishing an activity.

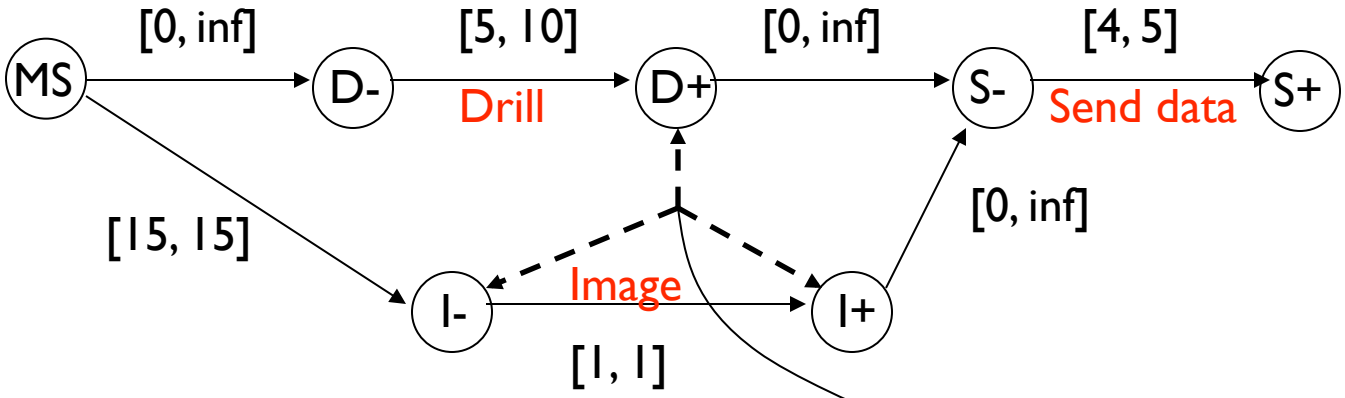
•Different methods of accomplishing an activity.

Temporal Relations Described by a DTN

Model-based Embedded & Robotic Systems

- Disjunctive Temporal Network (DTN)
  - Extends TCN by allowing non-binary constraints.

Activities of Mars Rover: Drill (D) , Image (I), Send Data (S)



Drilling causes vibration.

Image cannot occur

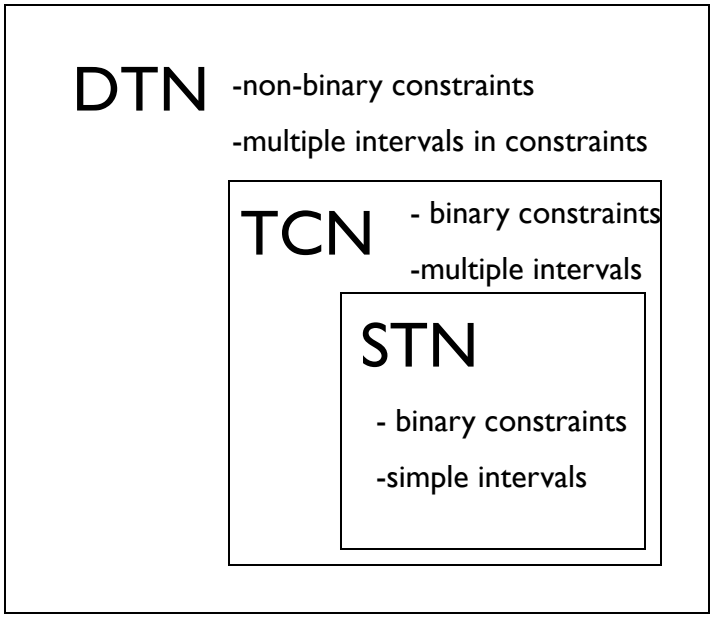
- during the last two minutes before drilling, or
- during the first minute after drilling ends.

$2 \leq D^+ - I^+ \leq inf$

OR

$I \leq I^- - D^+ \leq inf$

A Hierarchy of Temporal Relations



Tsamardinos, Pollack, M. Ganchev, ECP 01]
[Shah, Conrad, Williams ICAPS 09]
[Conrad, Shah, Williams ICAPS 09]

Outline: To Execute a Temporal Plan

Part I: Schedule Off-line

1. Describe Temporal Plan

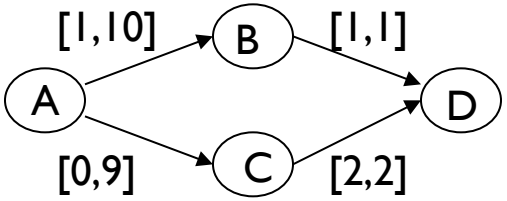
2. Test Consistency

3. Schedule Plan

offline

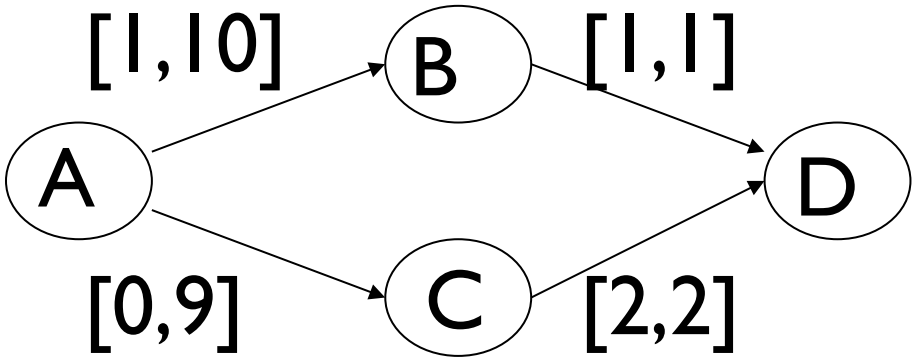
online

4. Execute Plan



Consistency of an STN

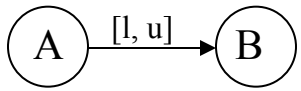
Input: An STN <X, C> where $C_j = \langle\langle X_k, X_i\rangle\langle a_j, b_j\rangle\rangle$



Output: True iff there exists an assignment to X satisfying C.

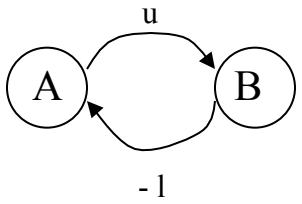
Map STN to Distance (D-)Graph

Constraint Graph



$l \leq B - A \leq u$

Distance Graph



$B - A \leq u$

$l \leq B - A$ $\quad A - B \leq -l$

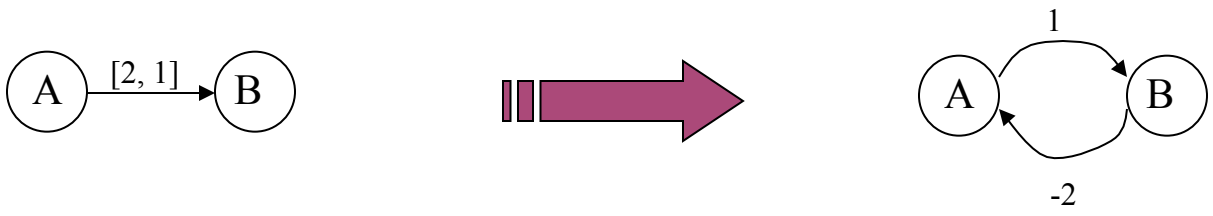
- Upperbound mapped to outgoing, non-negative arc.
- Lowerbound mapped to incoming, negative arc.

[Dechter, Meiri, Pearl 91]

Check D-Graph Consistency

- consistent iff d-graph has no negative cycles.
- Detect by computing shortest path from one node to all other nodes.
  - Single Source Shortest Path (SSSP).

Example of inconsistent constraint:



Constraint Graph **Distance Graph**

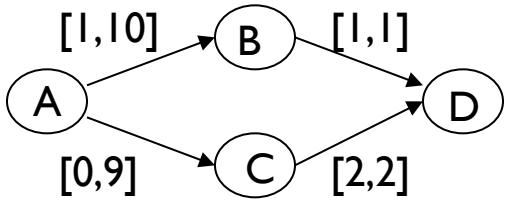
Outline: To Execute a Temporal Plan

Part I: Schedule Off-line

1. Describe Temporal Plan

2. Test Consistency

3. Schedule Plan



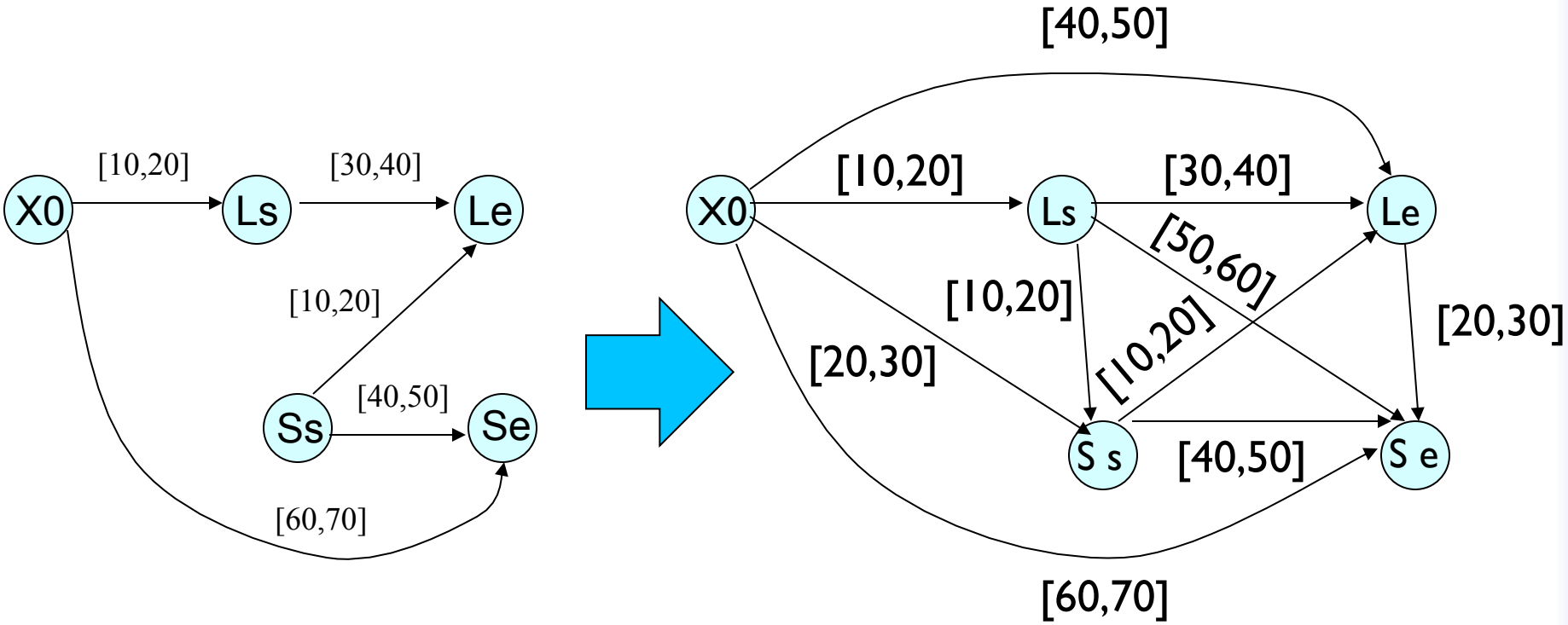
offline

online

4. Execute Plan

[Dechter, Meiri, Pearl 91]

Scheduling



Idea: Expose Implicit Constraints of STN ➔ Schedule

- Input: STN
- Output: “Decomposable” (Implied) STN
- Algorithm: All-Pairs-Shortest-Path (APSP) of D-graph (Floyd-Warshall).

Scheduling without Search: Solution by Decomposition

Model-based Embedded & Robotic Systems

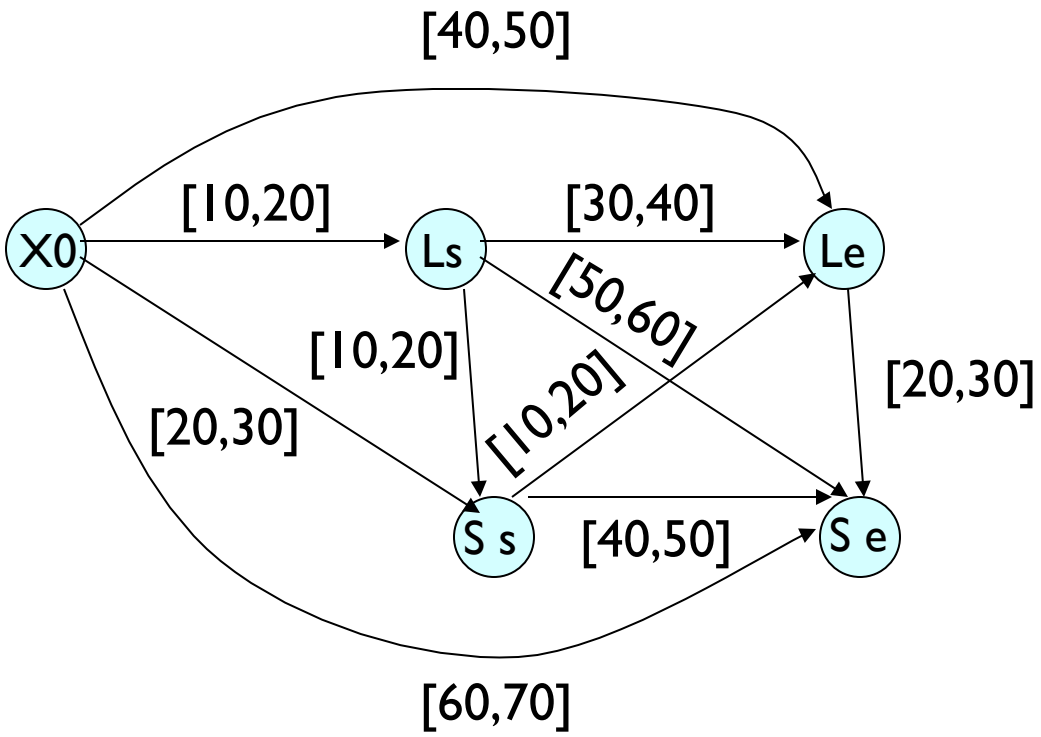
Input: Decomposable STN (APSP D-Graph)

Output: Schedule (Assignment to X, consistent with STN)

Property: Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**



Scheduling without Search: Solution by Decomposition

Model-based Embedded & Robotic Systems

Input: Decomposable STN (APSP D-Graph)

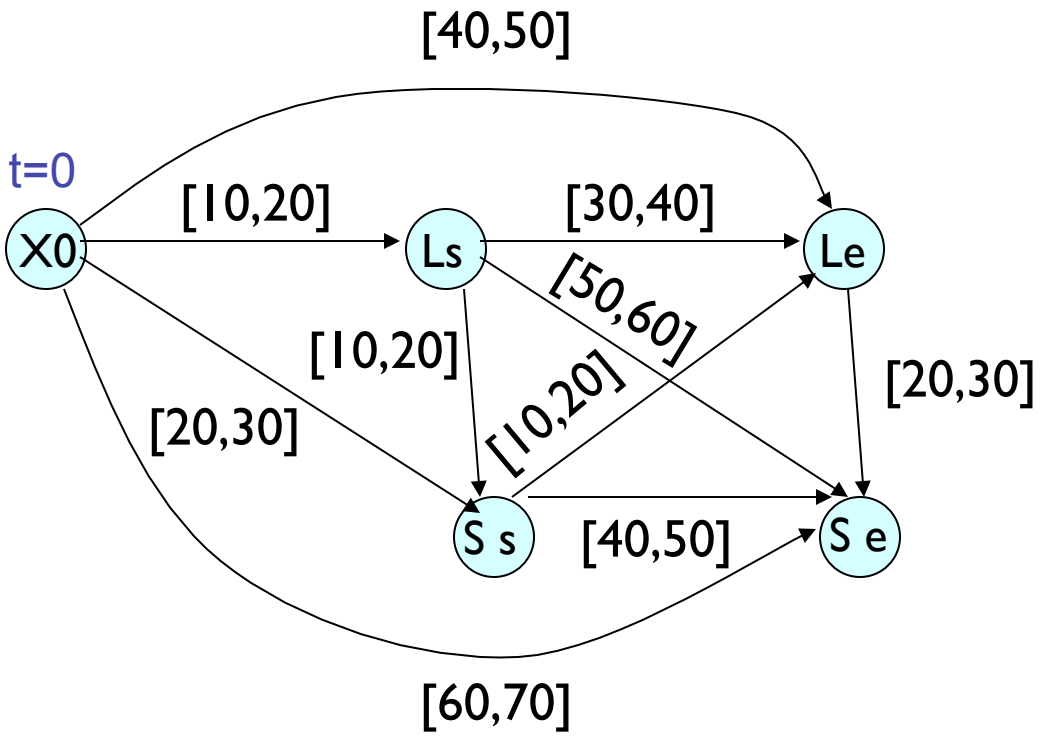
Output: Schedule (Assignment to X, consistent with STN)

Property: Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0



Scheduling without Search: Solution by Decomposition

Model-based Embedded & Robotic Systems

Input: Decomposable STN (APSP D-Graph)

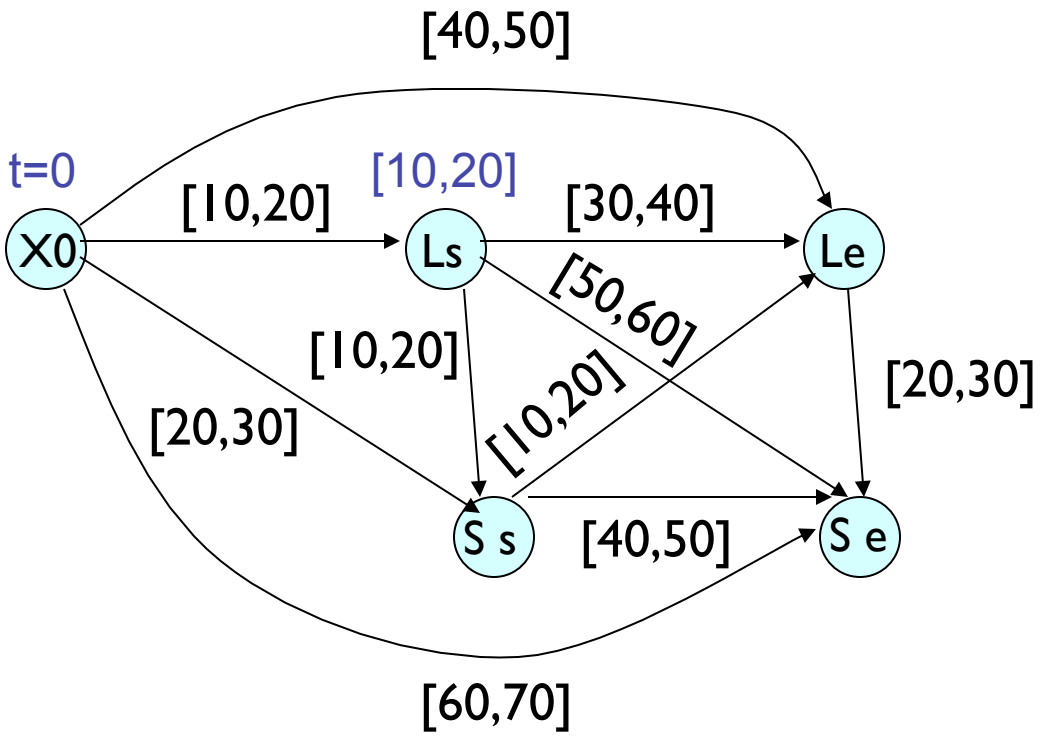
Output: Schedule (Assignment to X, consistent with STN)

Property: Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0
- Select value for Ls, consistent with X0



Scheduling without Search: Solution by Decomposition

Model-based Embedded & Robotic Systems

Input: Decomposable STN (APSP D-Graph)

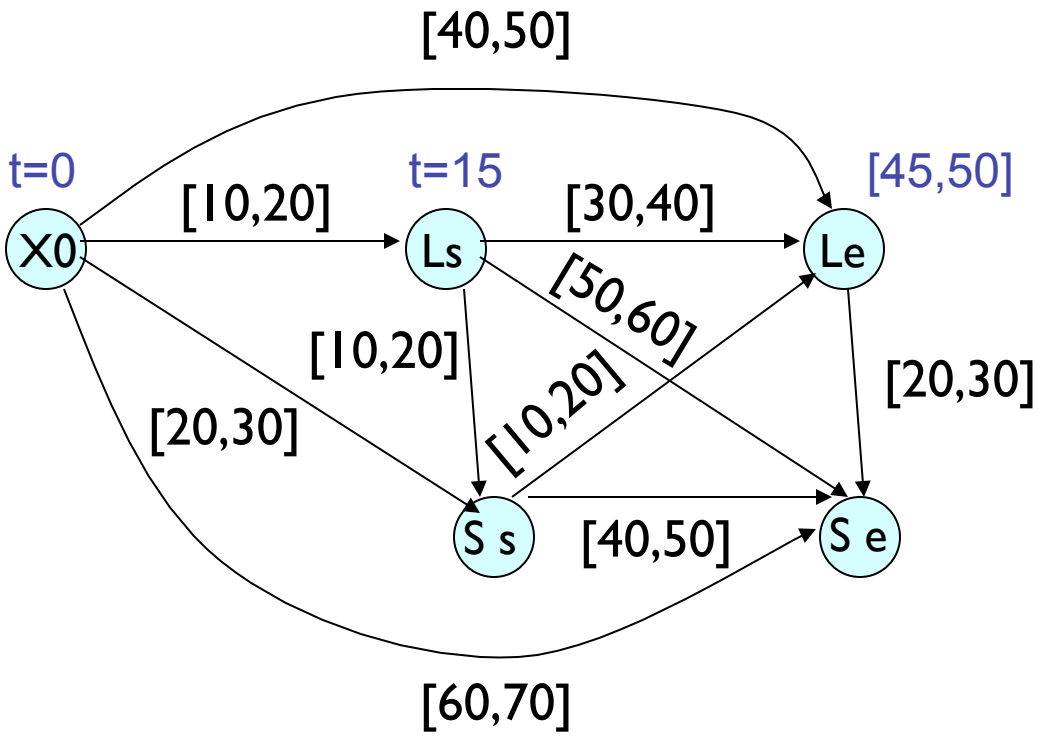
Output: Schedule (Assignment to X, consistent with STN)

Property: Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0
- Select value for Ls, consistent with X0
- Select value for Le, consistent with X0, Ls



Scheduling without Search: Solution by Decomposition

Model-based Embedded & Robotic Systems

Input: Decomposable STN (APSP D-Graph)

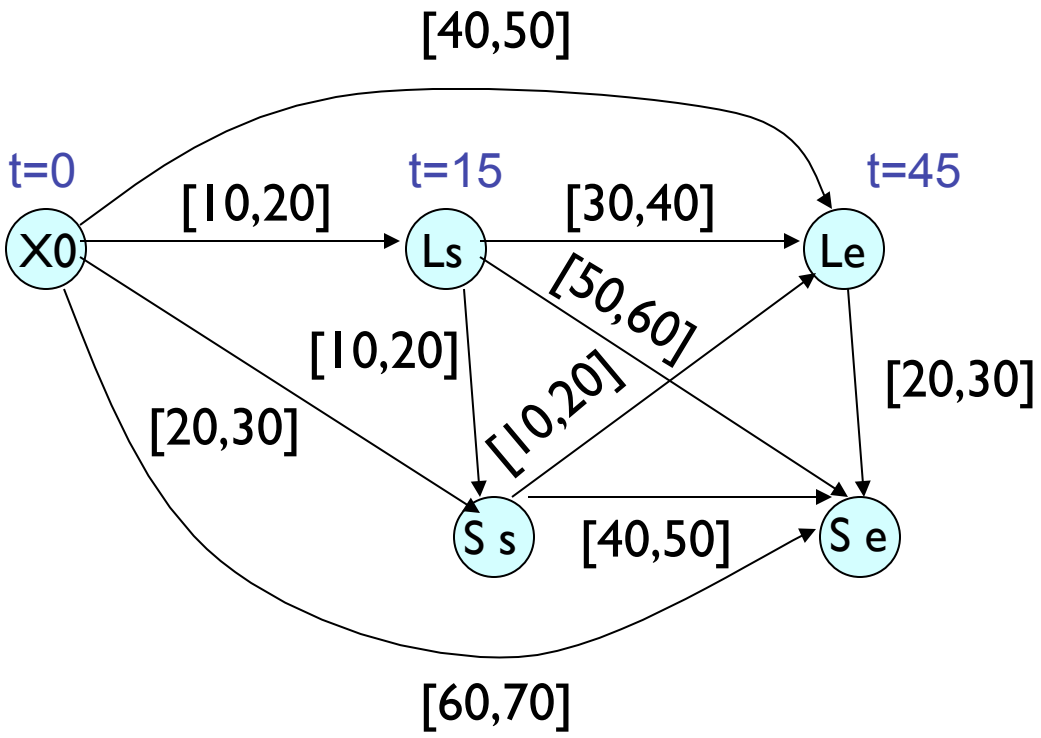
Output: Schedule (Assignment to X, consistent with STN)

Property: Can assign variables in any order, without backtracking.

Key ideas

- **Incrementally tighten feasible intervals, as commitments are made.**
- **Perform on demand.**

- Select value for X0
- Select value for Ls, consistent with X0
- Select value for Le, consistent with X0, Ls



To Execute a Temporal Plan

Part I : Schedule Off-line

1. Describe Temporal Plan

2. Test Consistency

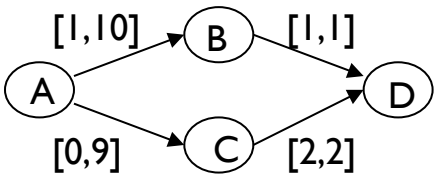
3. Schedule Plan

offline

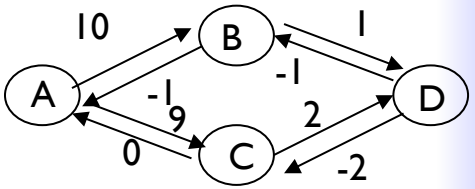
online

4. Execute Plan

STN



D Graph

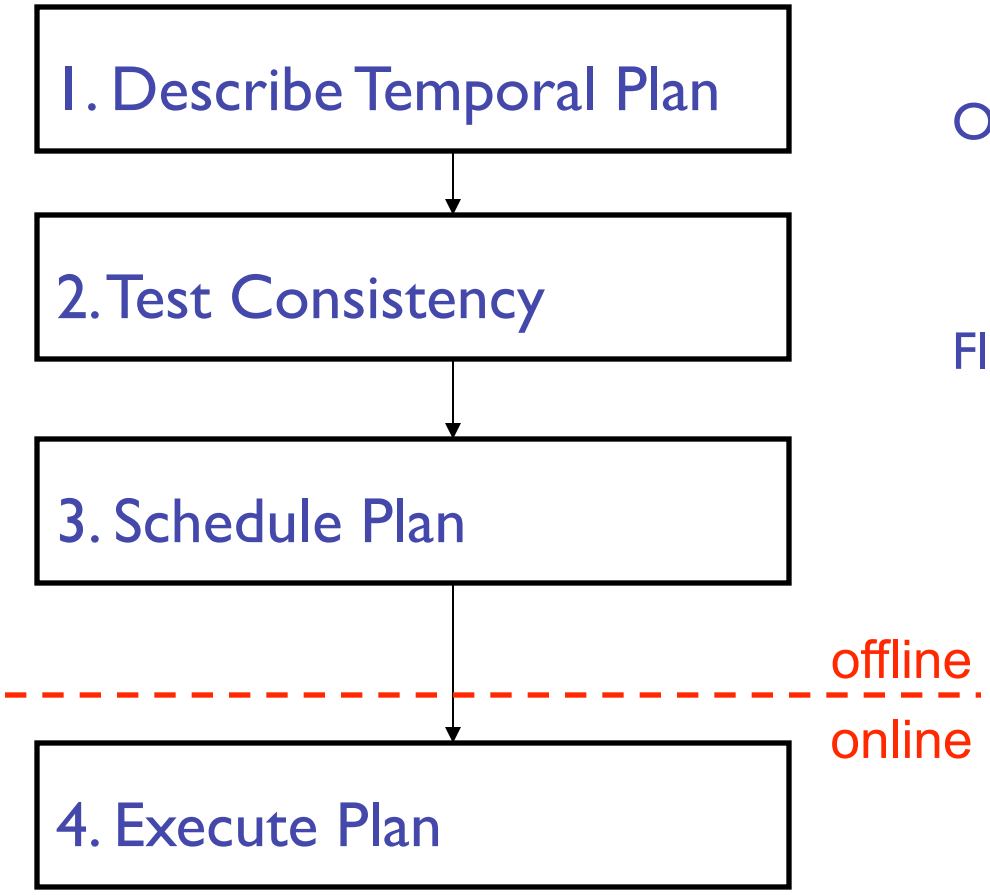


Detect negative loops (SSSP).

APSP + Decomposition.

To Execute a Temporal Plan

Part I: Schedule Off-line



Problem: delays and fluctuations in task duration can cause plan failure.

Observation: temporal plans leave room to adapt.

Flexible Execution adapts through dynamic scheduling [Muscettola et al]

- Assign time to event when executed.
- Guarantee that all constraints will be satisfied.
- Schedule with low latency through pre-compilation.

To Execute a Temporal Plan

Part I : Schedule Off-line

1. Describe Temporal Plan
2. Test Consistency
3. Schedule Plan
4. Execute Plan

offline

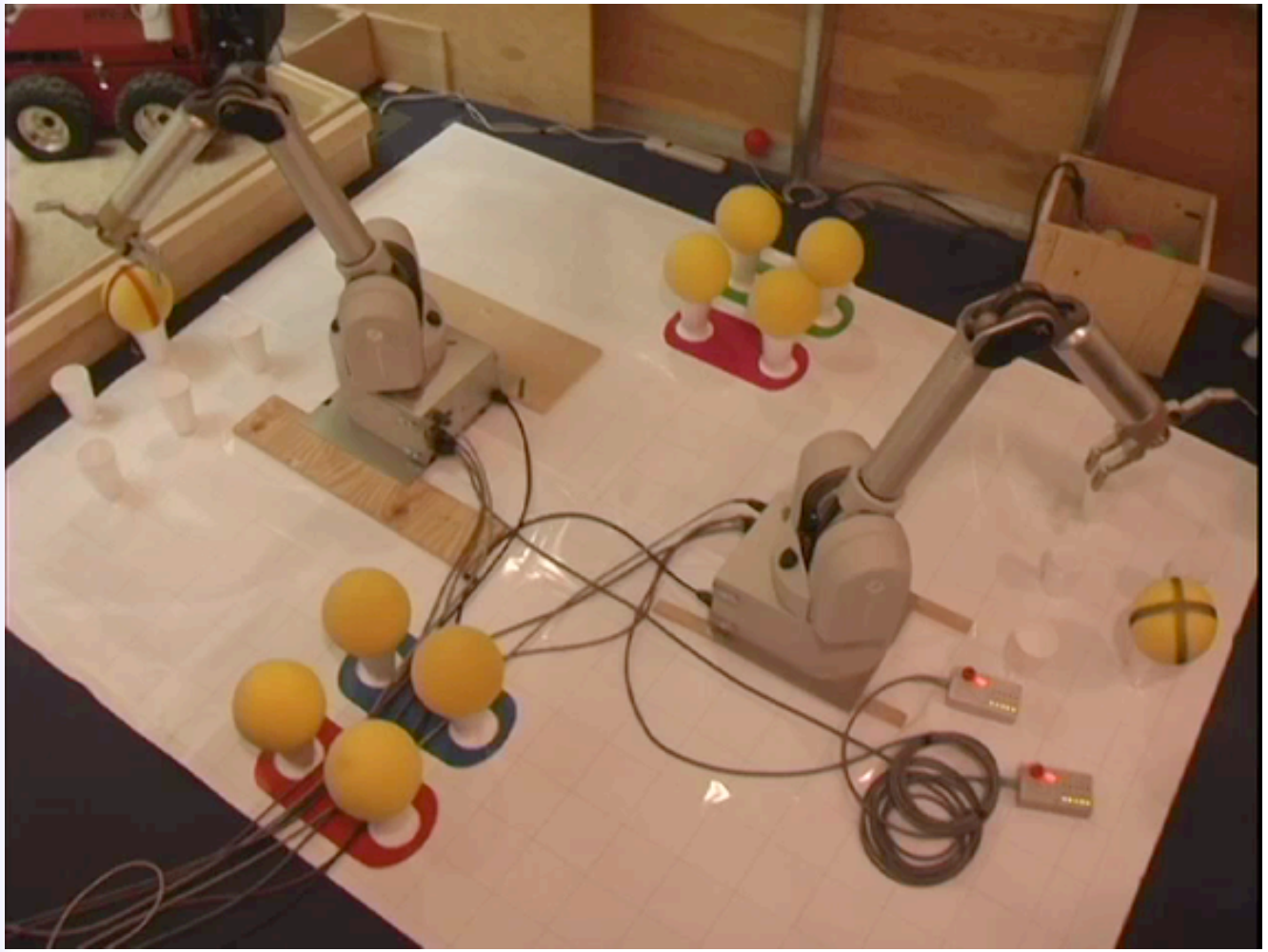
online

Part II: Schedule Online

1. Describe Temporal Plan
2. Test Consistency
3. Reformulate Plan
4. Dynamically Execute Plan

How do we schedule on line?

Multi-Robot Teamwork



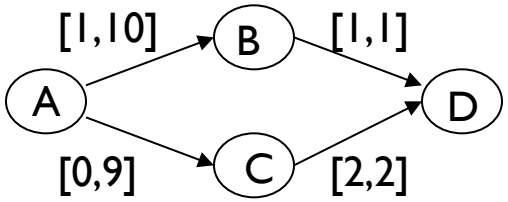
- Off-nominal
- Partner adapts in response to teammate's failure.

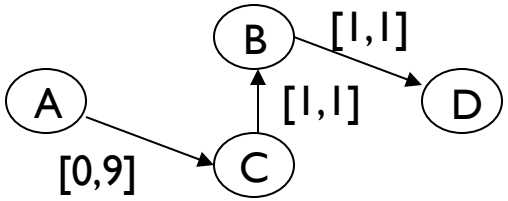
To Execute a Temporal Plan

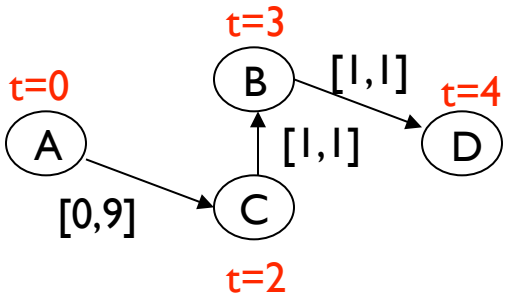
Model-based Embedded & Robotic Systems

[Muscettola, Morris, Tsamardinos KR98]

Part II: Schedule Online







1. Describe Temporal Plan

2. Test Consistency

3. Reformulate Plan

offline

online

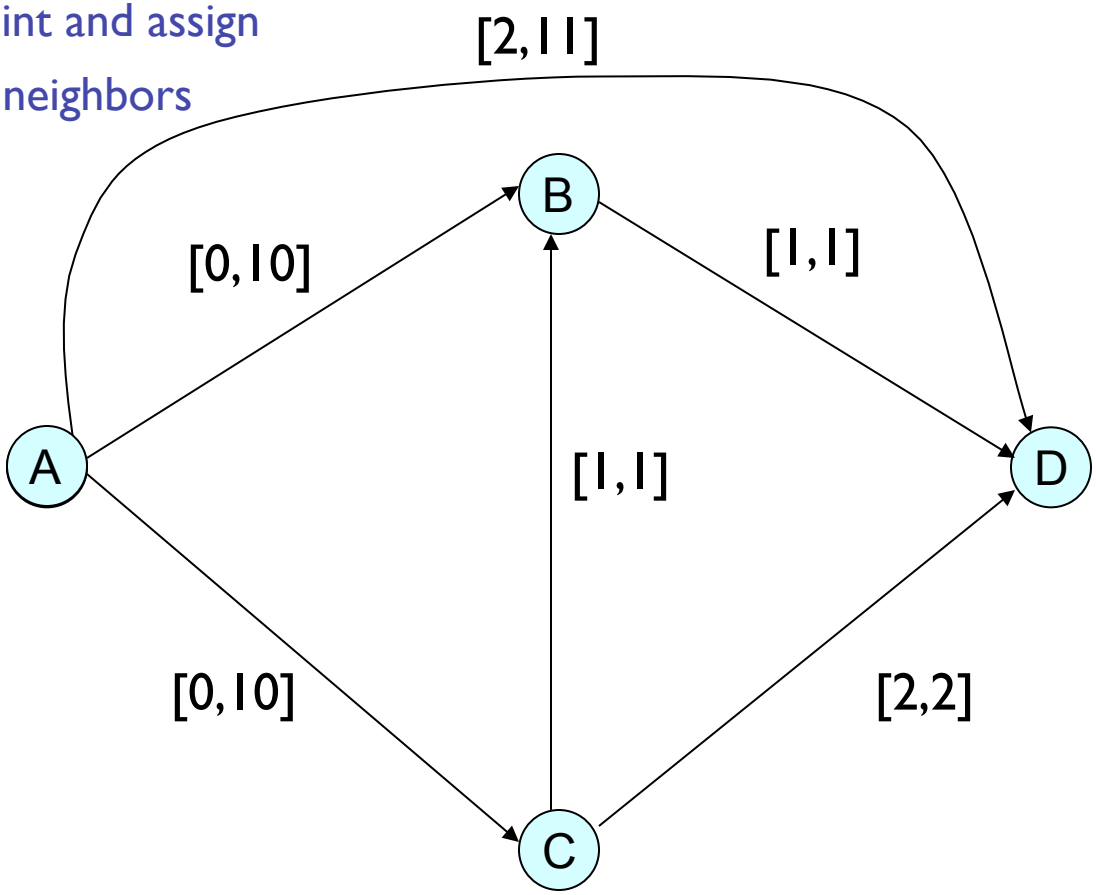
4. Dynamically Execute Plan

How do we schedule on line?

Dynamic Scheduling by Decomposition?

Consider a Simple Example

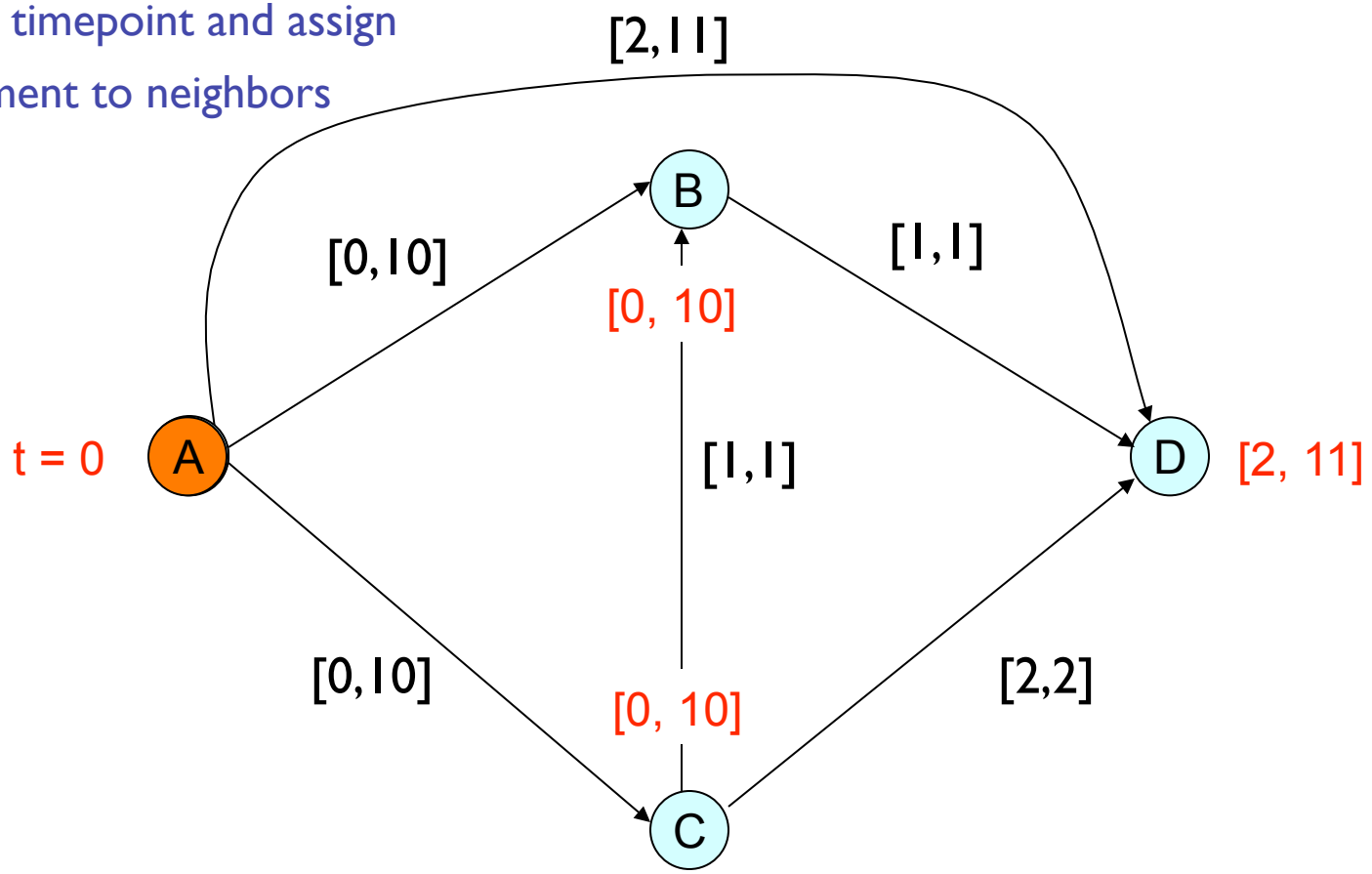
- Select executable timepoint and assign
- Propagate assignment to neighbors



Dynamic Scheduling by Decomposition?

Consider a Simple Example

- Select executable timepoint and assign
- Propagate assignment to neighbors



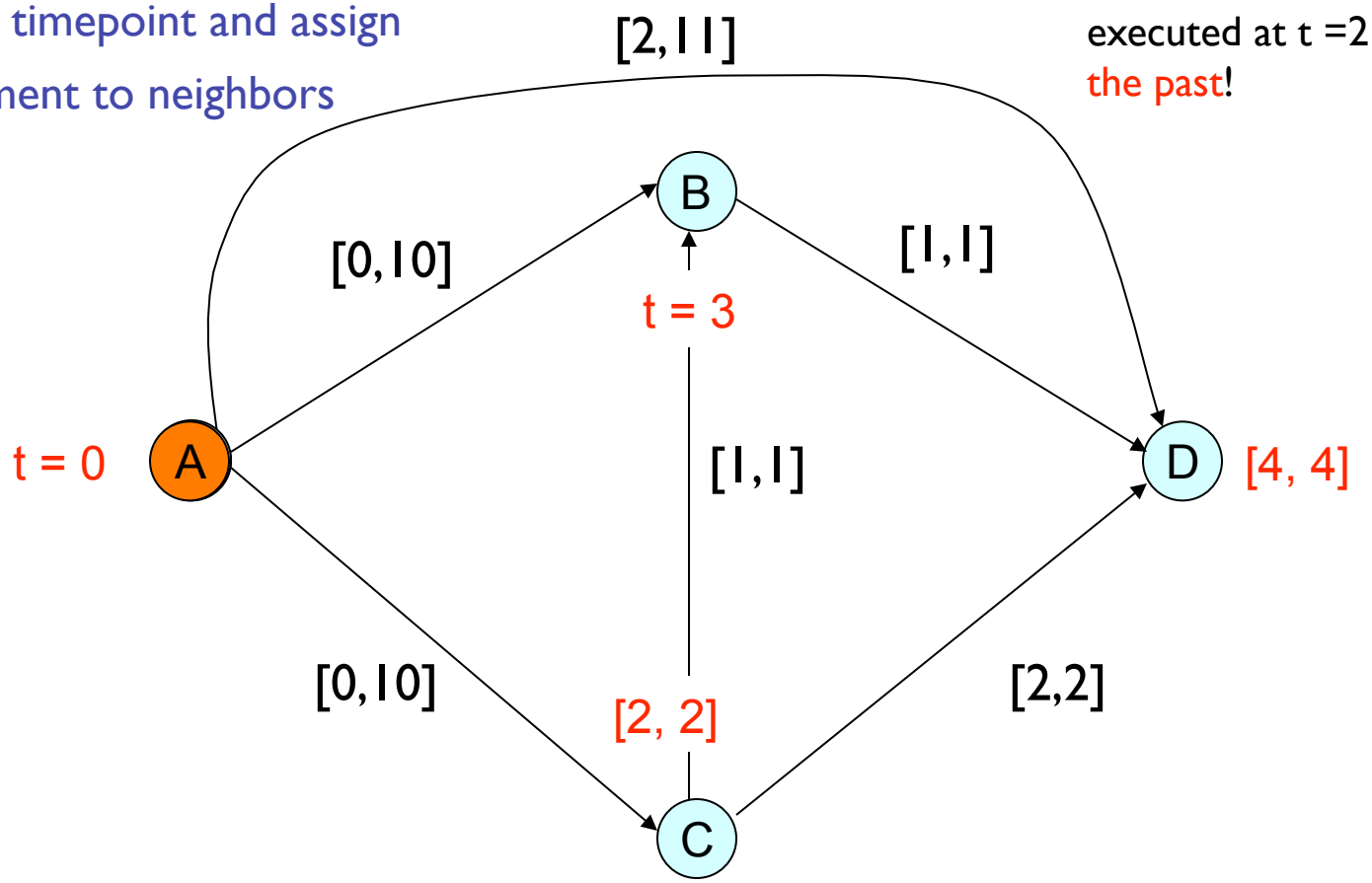
~~Dynamic Scheduling by Decomposition?~~

Consider a Simple Example

- Select executable timepoint and assign
- Propagate assignment to neighbors

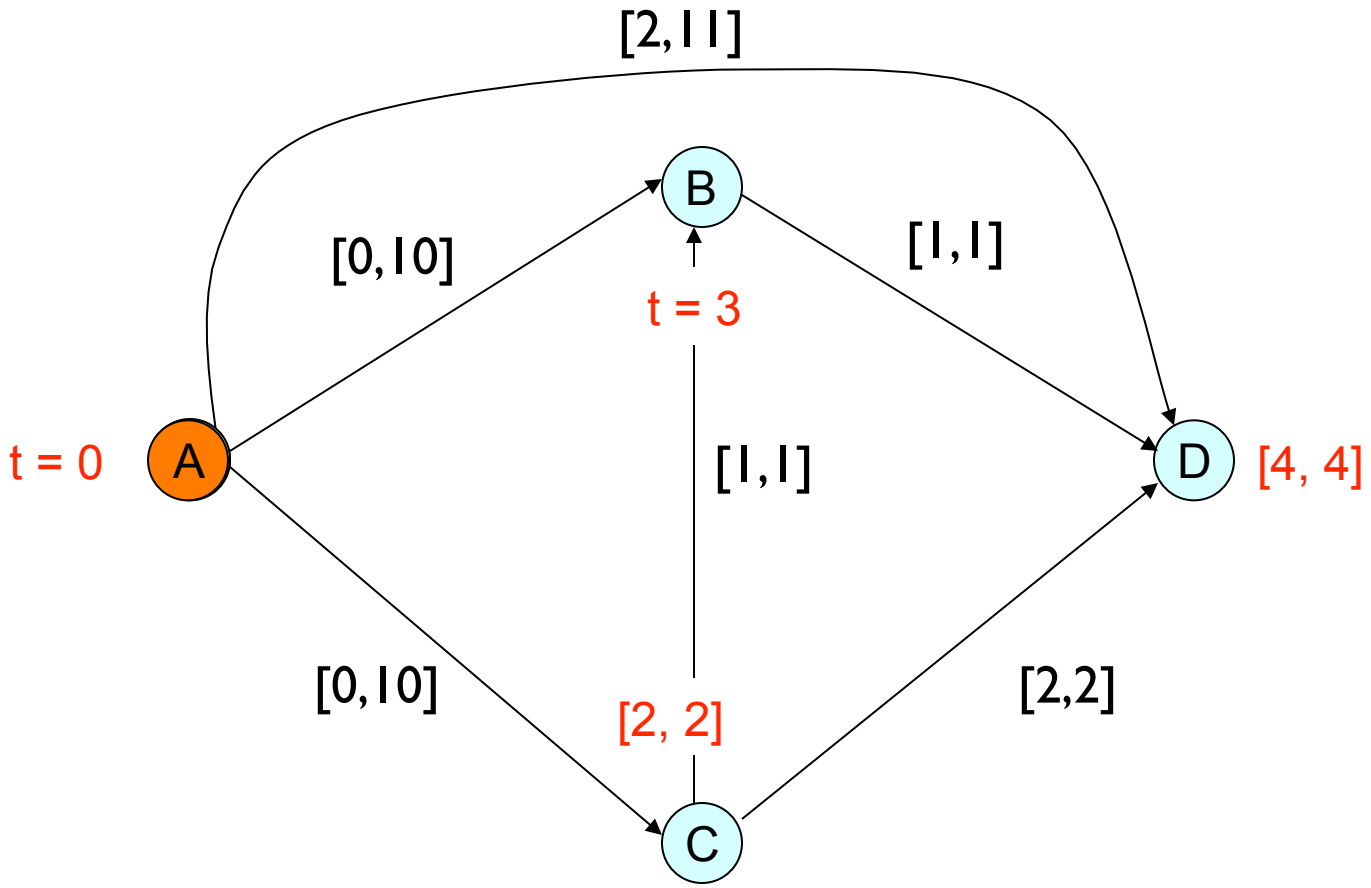
Uh oh!

C must be executed at t =2 in the past!



~~Dynamic Scheduling by Decomposition?~~

- How can we fix it?
  - Assignments must monotonically increase in value
  - Must respect induced orderings



Dispatching Execution Controller

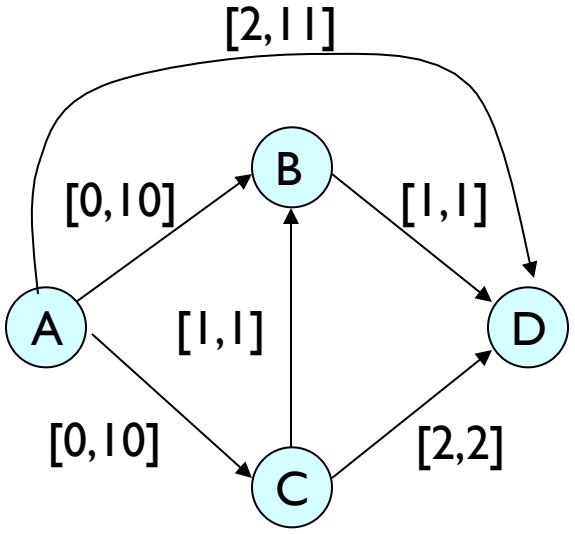
- How can we fix it?
  - Assignments must monotonically increase in value
  - Must respect induced orderings

- Execute an event when enabled and active

  - A is enabled – Predecessors of A are scheduled.

  - A is active – Current time is within bound of A

  - A is a *predecessor* of B if BA has negative weight, (A - B < [-]) hence A + [+] < B.



STN Dispatching

Initially:

- E(nabled) =Time points w/o predecessors
- S(cheduled) = { }

Repeat:

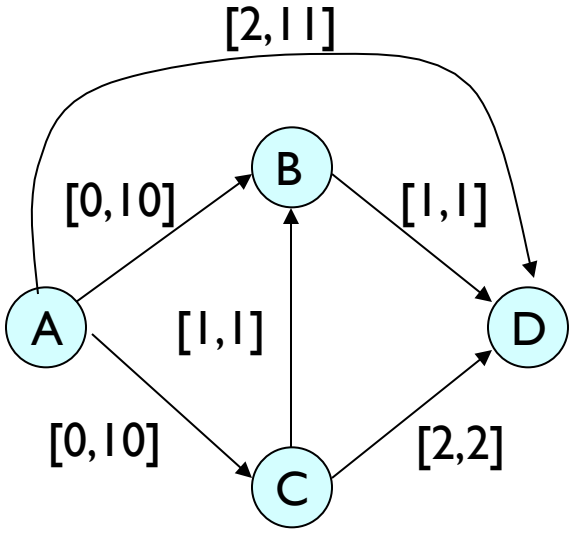
1. Wait until current time has advanced such that some TP in E is active.
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors
5. Add to E, all immediate neighbors that become enabled
    - TPx enabled if all +lb edges starting at TPv have their destination in S.

E = {A}

S = { }

Predecessors:

A	none
B	A, C
C	A
D	A, B, C



STN Dispatching

Initially:

- E = Time points w/o predecessors
- S = { }

Repeat:

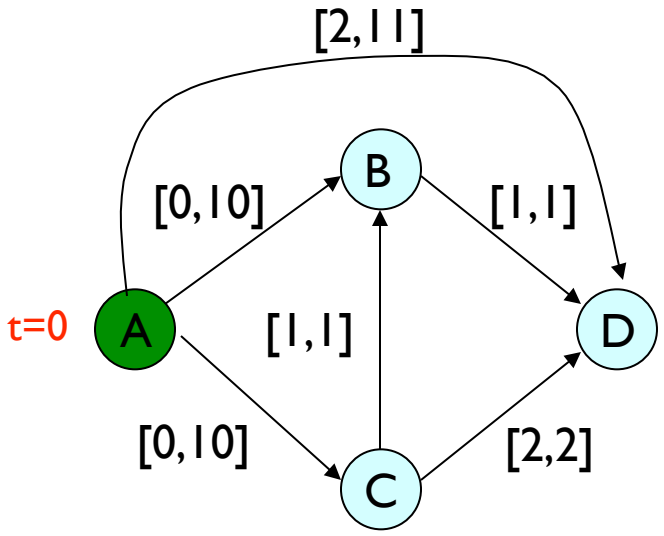
1. Wait until current time has advanced such that some TP in E is active.
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors.
5. Add to E, all immediate neighbors that become enabled.
    - TPx enabled if all +lb edges starting at TPx have their destination in S.

E = { }

S = {A @ t = 0}

Predecessors:

A	none
B	A, C
C	A
D	A, B, C



STN Dispatching

Initially:

- E = Time points w/o predecessors
- S = { }

Repeat:

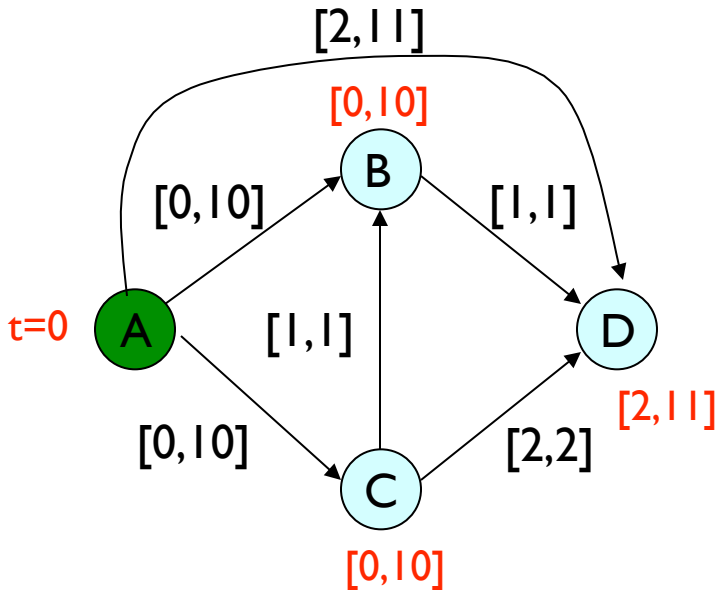
1. Wait until current time has advanced such that some TP in E is active.
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors.
5. Add to E, all immediate neighbors that become enabled.
    - TPx enabled if all +lb edges starting at TPx have their destination in S.

E = { }

S = {A @ t = 0}

Predecessors:

A	none
B	A, C
C	A
D	A, B, C



STN Dispatching

Initially:

- E = Time points w/o predecessors
- S = { }

Repeat:

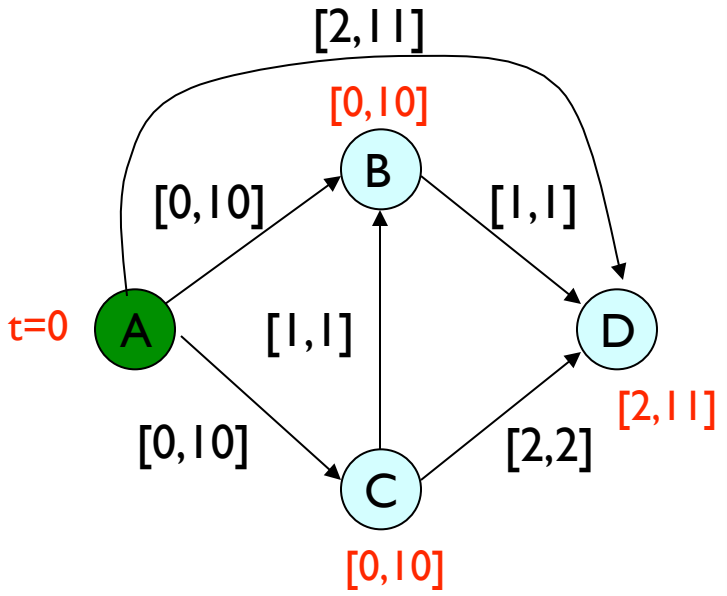
1. Wait until current time has advanced such that some TP in E is active.
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors.
5. Add to E, all immediate neighbors that become enabled.
    - TPx enabled if all +lb edges starting at TPx have their destination in S.

E = {C} (not B,D)

S = {A @ t = 0}

Predecessors:

A	none
B	A, C
C	A
D	A, B, C



STN Dispatching

Initially:

- E = Time points w/o predecessors
- S = { }

Repeat:

1. Wait until current time has advanced such that some TP in E is active.
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors.
5. Add to E, all immediate neighbors that become enabled.
    - TPx enabled if all +lb edges starting at TPx have their destination in S.

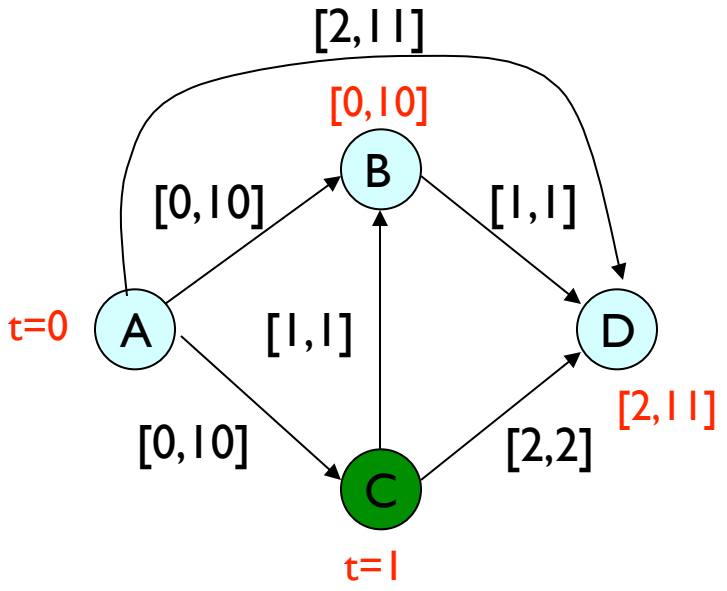
E = { }

S = {A @ t = 0

C @ t = 1}

Predecessors:

A	none
B	A, C
C	A
D	A, B, C



STN Dispatching

Initially:

- E = Time points w/o predecessors
- S = { }

Repeat:

1. Wait until current time has advanced such that some TP in E is active.
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors.
5. Add to E, all immediate neighbors that become enabled.
    - TPx enabled if all +lb edges starting at TPx have their destination in S.

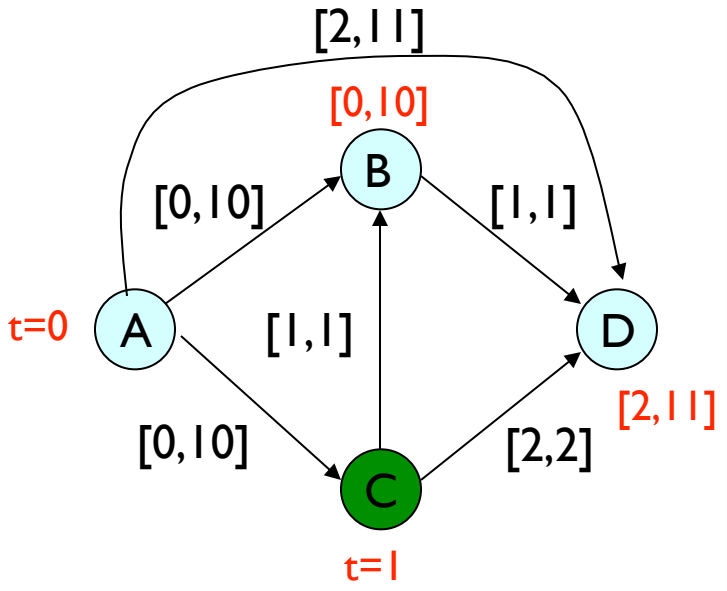
E = { }

S = {A @ t = 0

C @ t = 1}

Predecessors:

A	none
B	A, C
C	A
D	A, B, C



STN Dispatching

Initially:

- E = Time points w/o predecessors
- S = { }

Repeat:

1. Wait until current time has advanced such that some TP in E is active.
2. Set TP's execution time to current time.
3. Add TP to S.
4. Propagate time of execution to TP's immediate neighbors.
5. Add to E, all immediate neighbors that become enabled.
    - TPx enabled if all +lb edges starting at TPx have their destination in S.

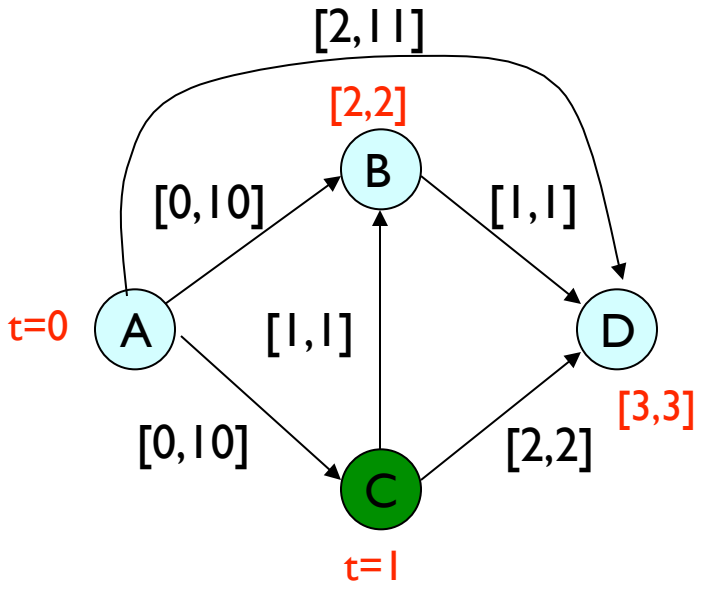
E = { }

S = {A @ t = 0

C @ t = 1}

Predecessors:

A	none
B	A, C
C	A
D	A, B, C



Outline: To Execute a Temporal Plan

Part II: Schedule Online

1. Describe Temporal Plan

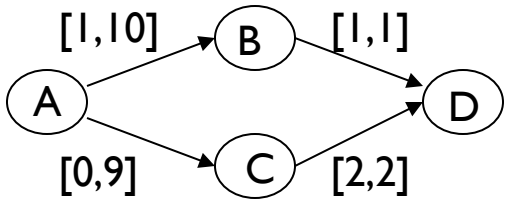
2. Test Consistency

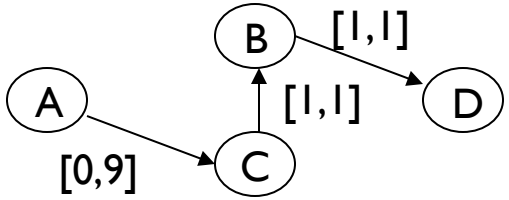
3. Reformulate Plan

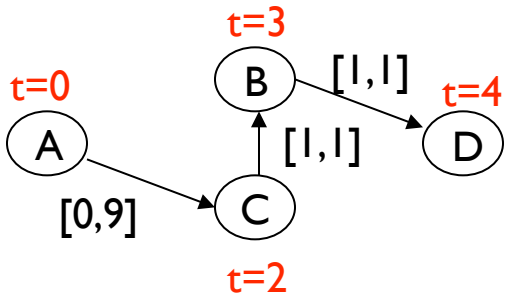
offline

online

4. Dynamically Execute Plan

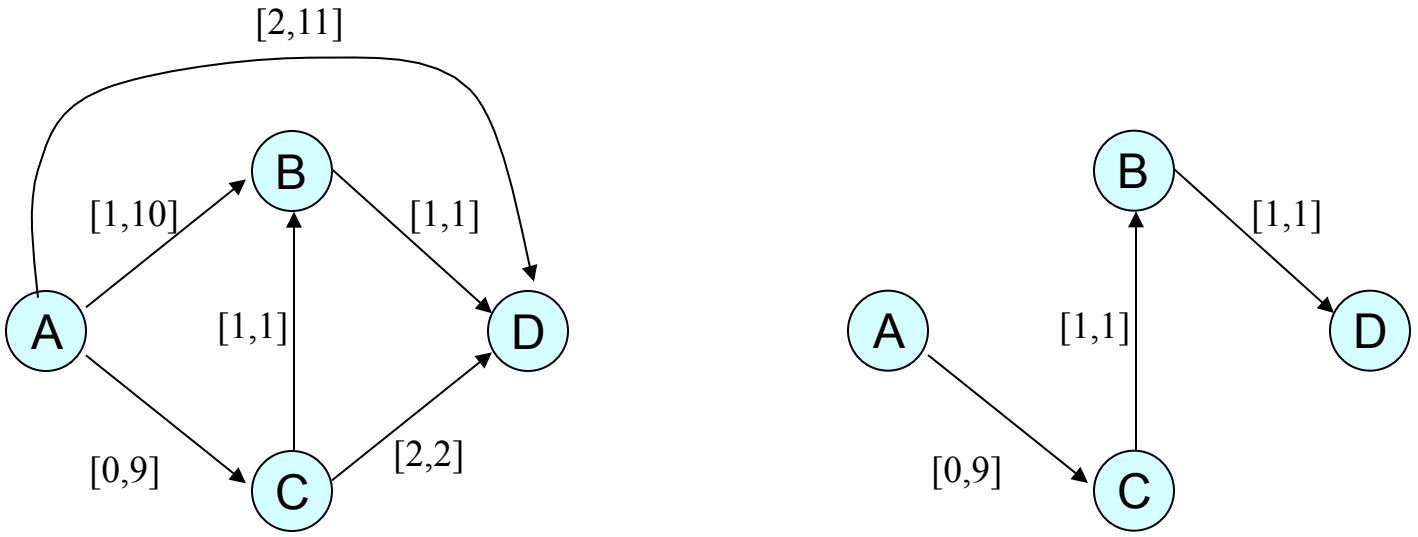






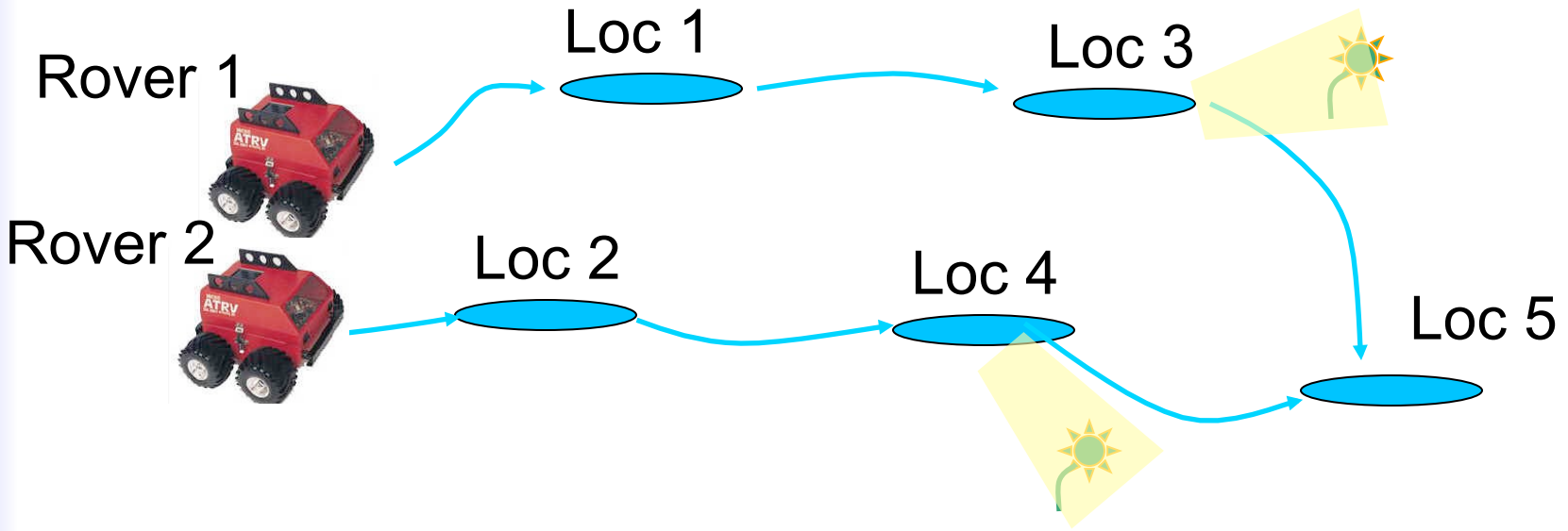
Issues in Flexible Execution

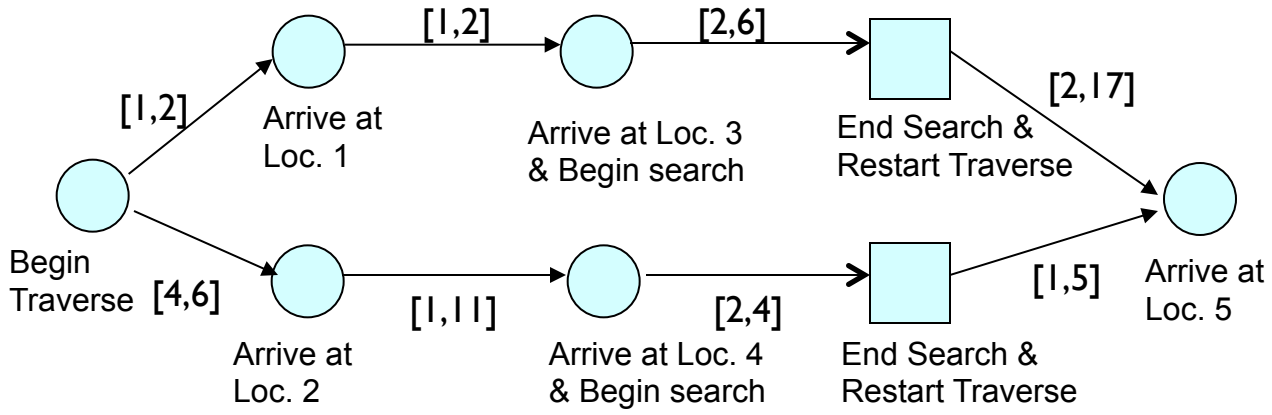
- How do we minimize execution latency?
  - Identify and remove redundant edges.



- Two dispatchable forms with equivalent results.

Temporal Reasoning with Uncertainty

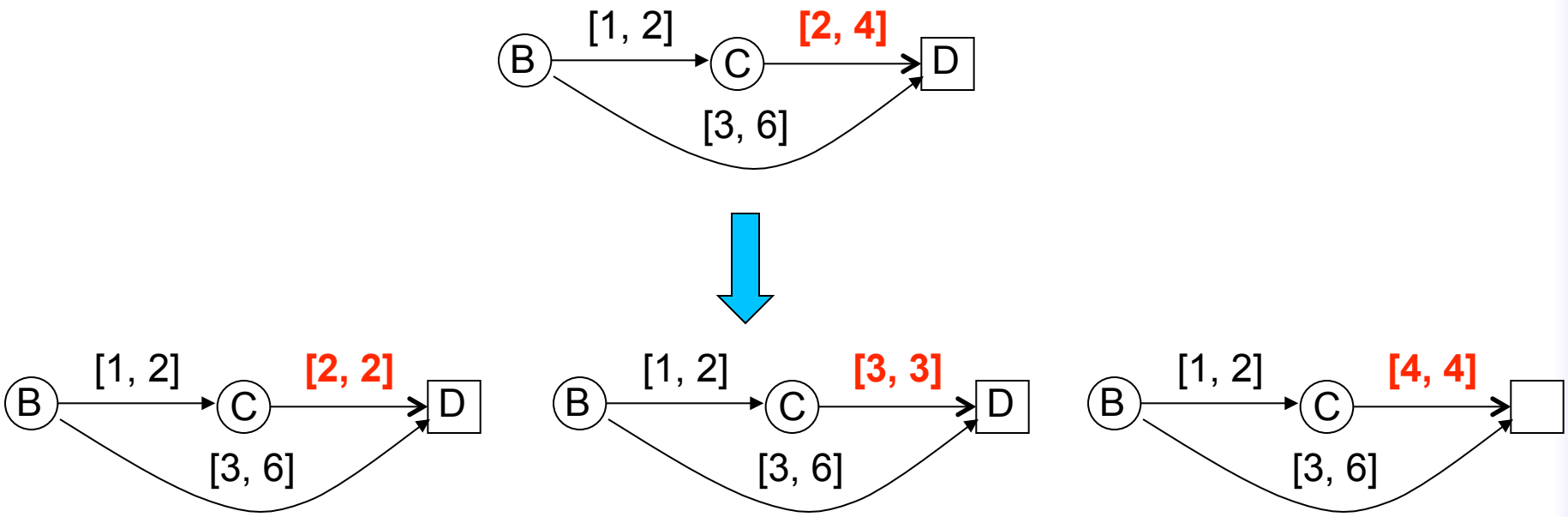




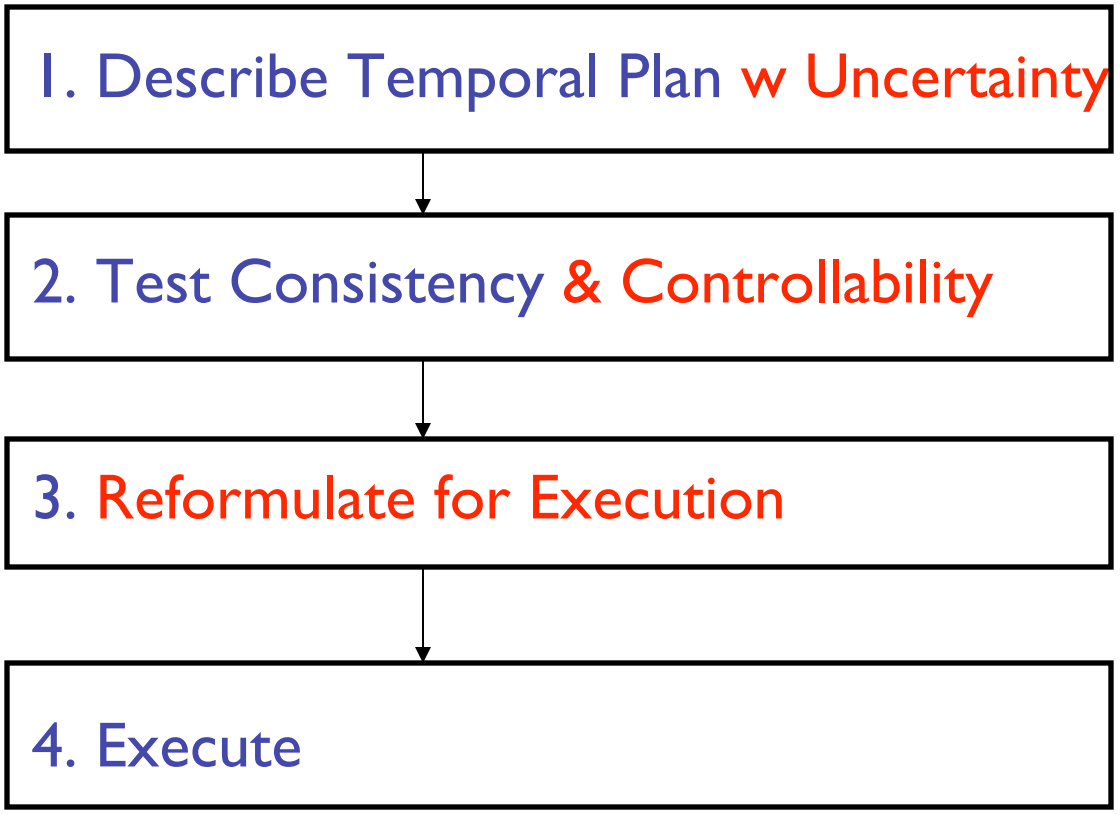
Simple Temporal Network with Uncertainty (STNU).

Definition of STNU

STNU is equivalent to a family of STNs *(projections)*, one for each allowed assignment to uncontrolled durations.



To Execute a Temporal Plan with Uncertainty

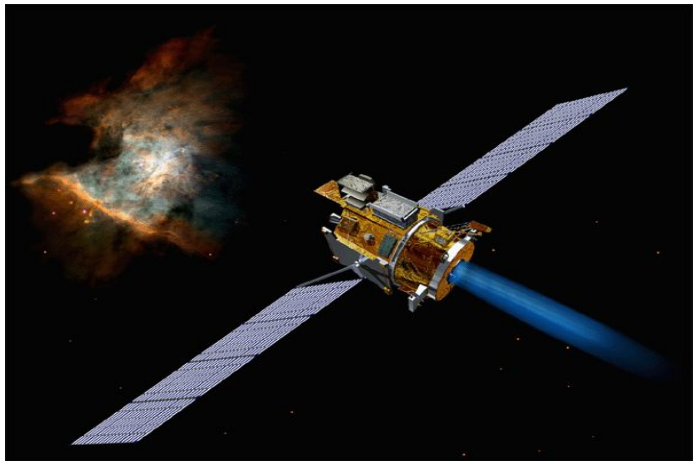


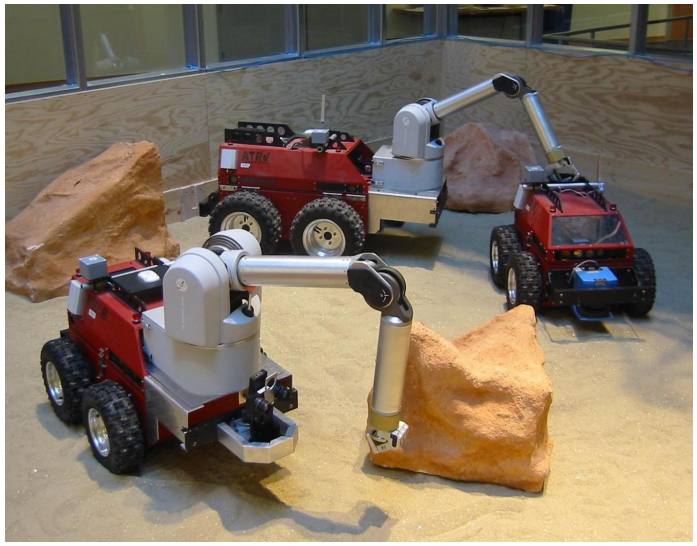
Queries about STNUs

- Is the STNU *consistent*?
  - Exists an assignment to executable time points consistent with some outcomes for uncontrollable durations.
- Is the STNU *controllable*?
  - Exists assignments to executable time points consistent with all outcomes for uncontrollable durations.
  - Strong Controllability
    - Assignment can be generated *a priori*.
  - Dynamic Controllability
    - Assignment can be generated online, given observations of past uncontrollable durations.

[Morris, Muscettola, Vidal IJCAI 01]

Dynamic Scheduling through Dispatchable Execution

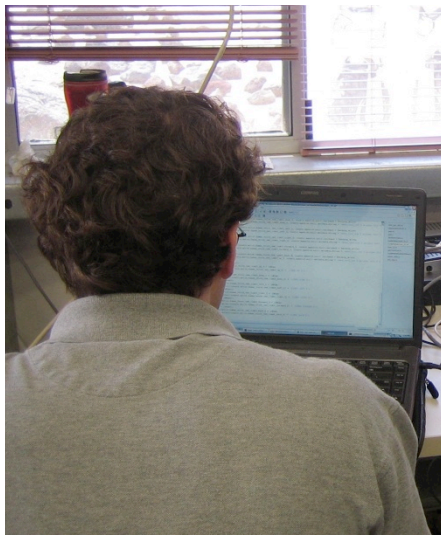




Operating JPL's Athlete Lunar Rover

Commanded through time-stamped sequences, similar to Spirit and Opportunity.

Reactive Model-based Programming Language (RMPL)



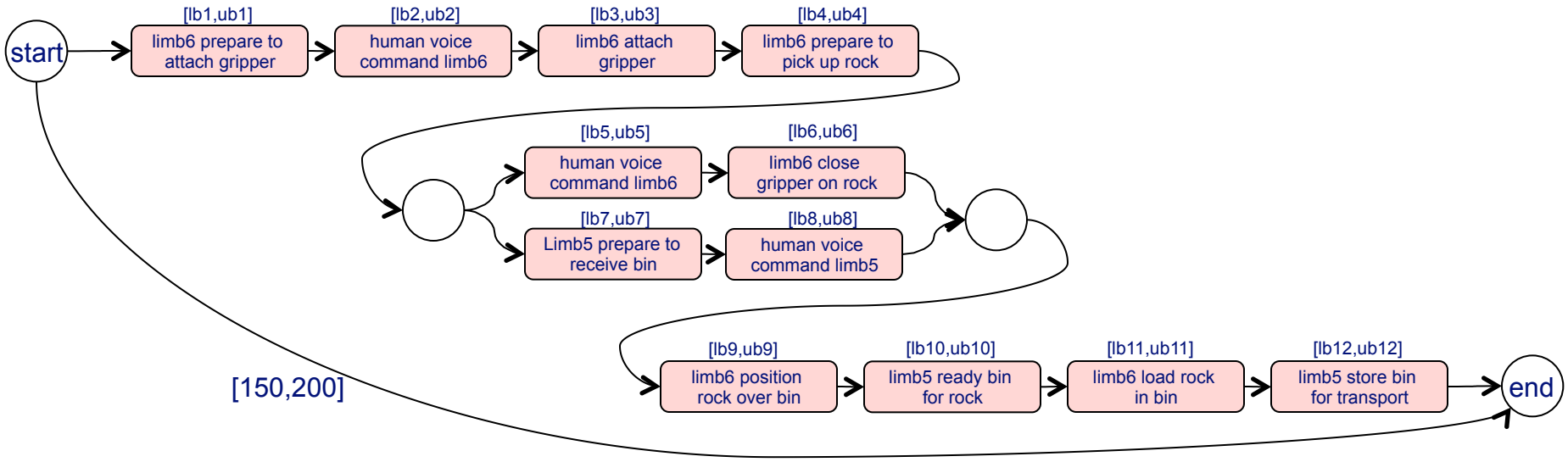
Write Common sense instructions

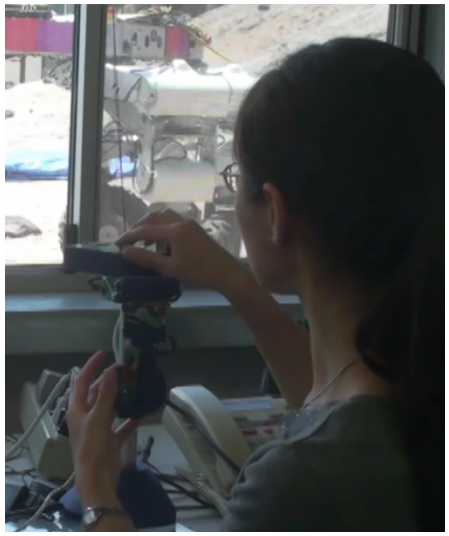
```
method run ()
{
    [1s,200s] sequence {
        prepare  limb(6) to attach gripper
        human voice commands the limb
        attach gripper to limb
        prepare limp to pick up rock with gripper

        parallel {
            sequence {
                human voice command the limb
                close gripper on rock
            };
            sequence {
                limb5 prepare limb (5) to receive bin
                human voice commands the limb
            };
        };

        position rock over bin with gripper
        ready bin for rock
        load rock in bin
        store bin for transport
    }
}
```

Reactive Model-based Programming Language (RMPL)

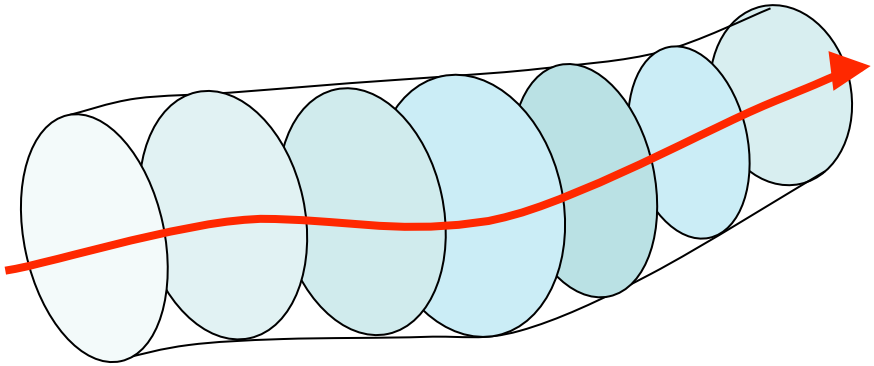
```
method run ()
{
    [1s,200s] sequence {
        prepare  limb(6) to attach gripper
        human voice commands the limb
        attach gripper to limb
        prepare limp to pick up rock with gripper

        parallel {
            sequence {
                human voice command the limb
                close gripper on rock
            };
            sequence {
                limb5 prepare limb (5) to receive bin
                human voice commands the limb
            };
        };

        position rock over bin with gripper
        ready bin for rock
        load rock in bin
        store bin for transport
    }
}
```

Compile to Temporal Plan Network





Demonstrate actions by example



Learns tubes of valid trajectories

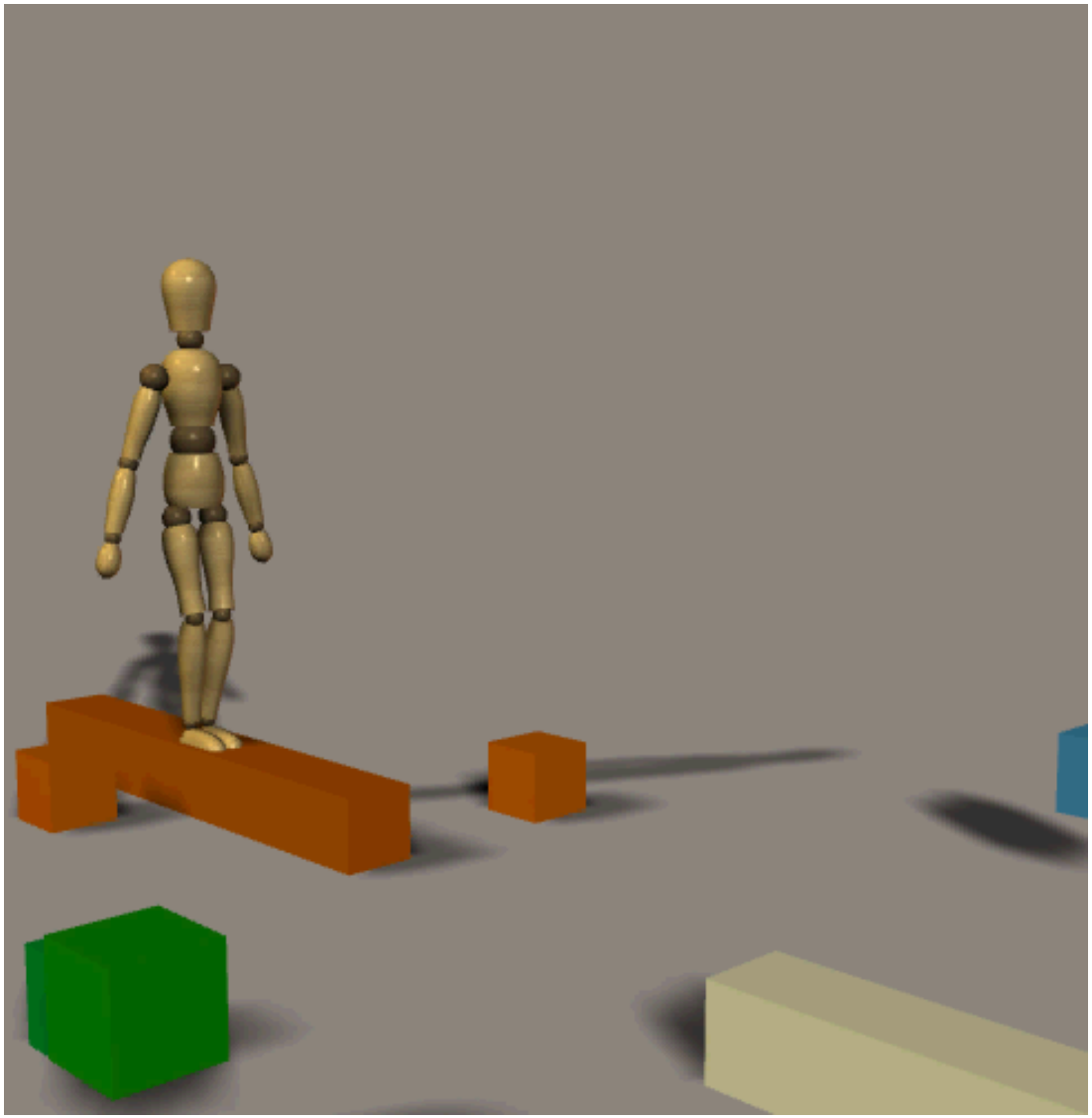
Collaborate with
Verbal commands

Athlete Demonstration – July, 2009

Robust, Model-based Execution (1): Coordination and Dynamic Scheduling

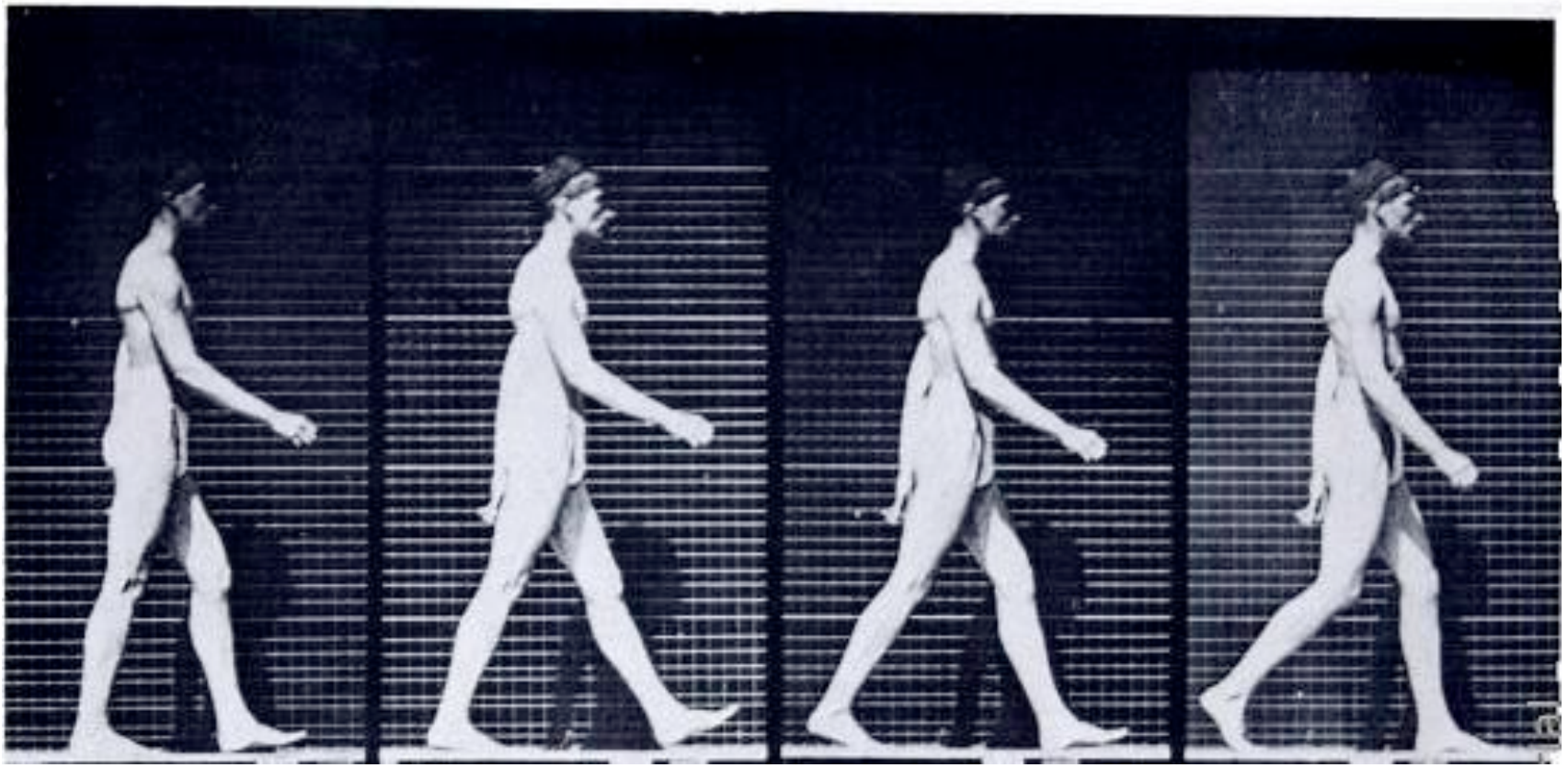
- Robust, model-based execution of time critical tasks.
- Task coordination through dynamic scheduling.
- Task coordination for under-actuated systems.
- Task coordination for multi-robot systems.

Dynamic Plan Execution for Under-actuated Systems



[Hofmann, Williams AAAI06]

Example: Describe Walking Tasks with Qualitative Poses



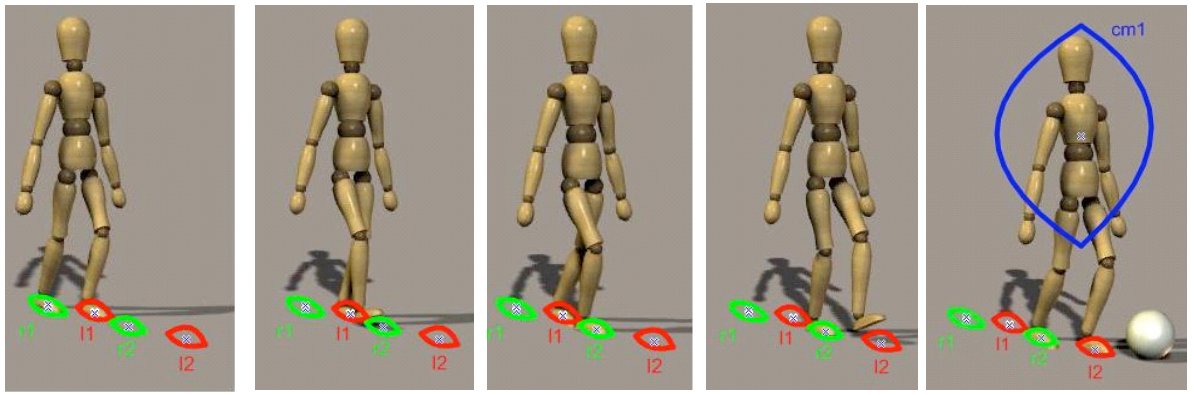
[Muybridge, 1955] Depicted gaits as sequences of distinct qualitative poses

Specify as temporal plan over qualitative states

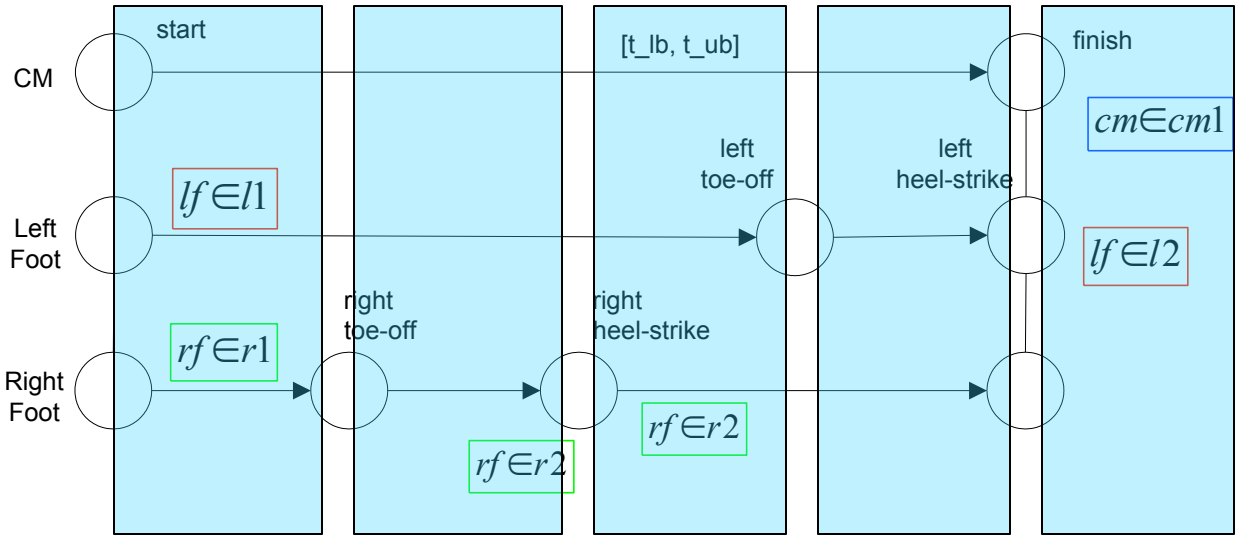
Supported by NASA

Qualitative State Plan

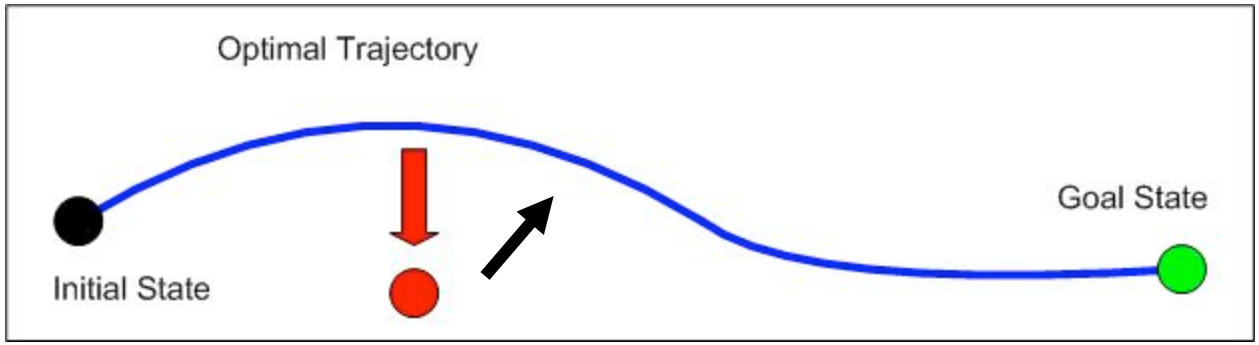
Muybridge



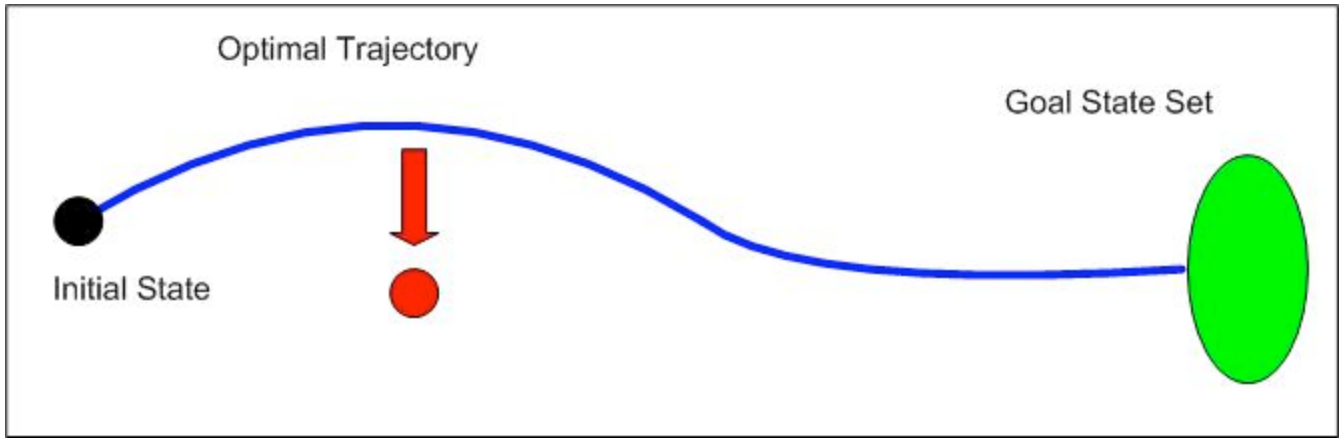
Input:
Qualitative
State Plan



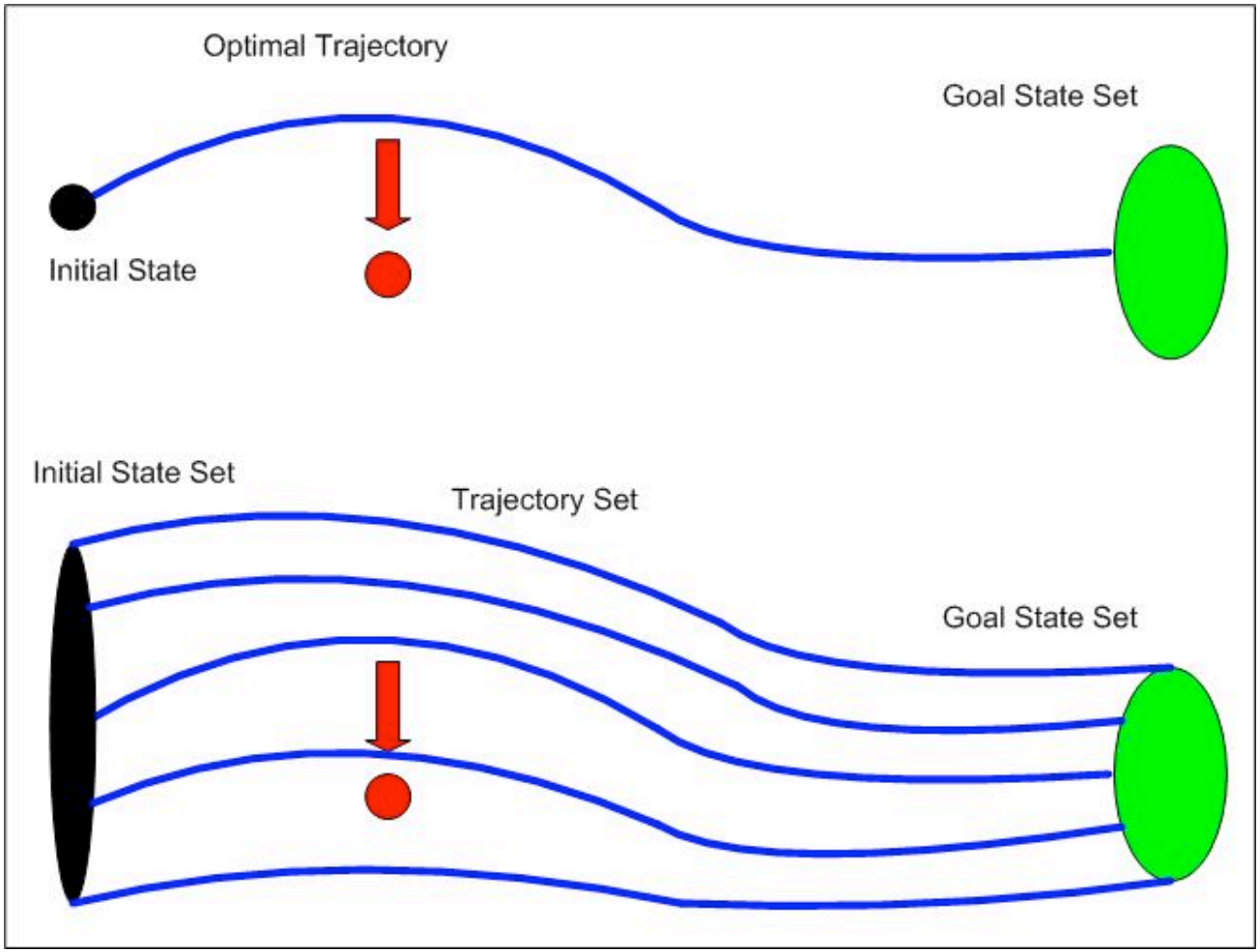
Traditional biped control tracks a reference trajectory



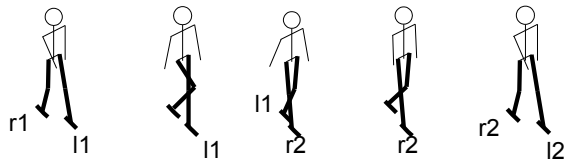
Executive achieves robustness by utilizing the flexibility of the Qualitative State Plan

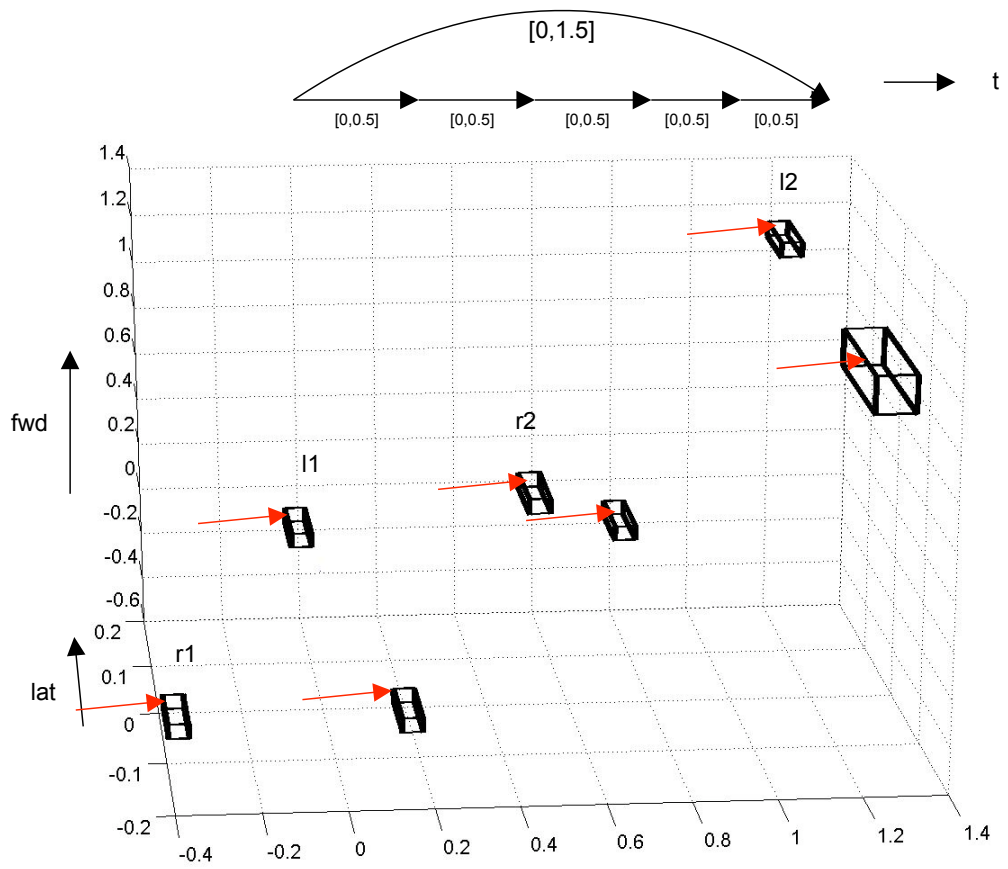


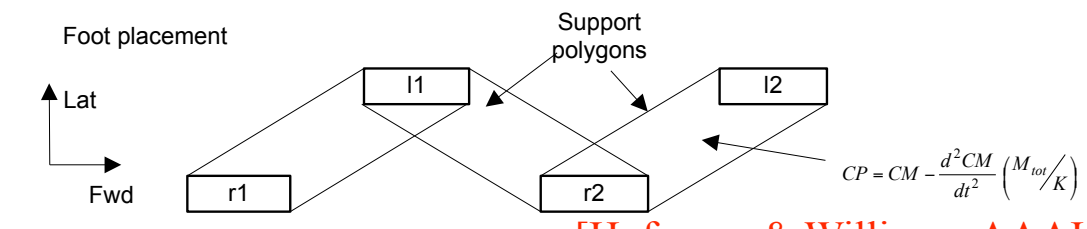
Executive achieves compliance by precomputing all feasible trajectories, not just one!



Feasible trajectories must go through goal regions

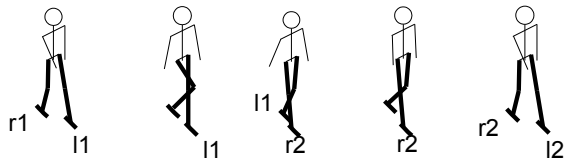






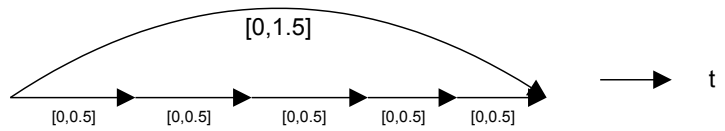
[Hofmann & Williams, AAAI 06; ICAPS 06]

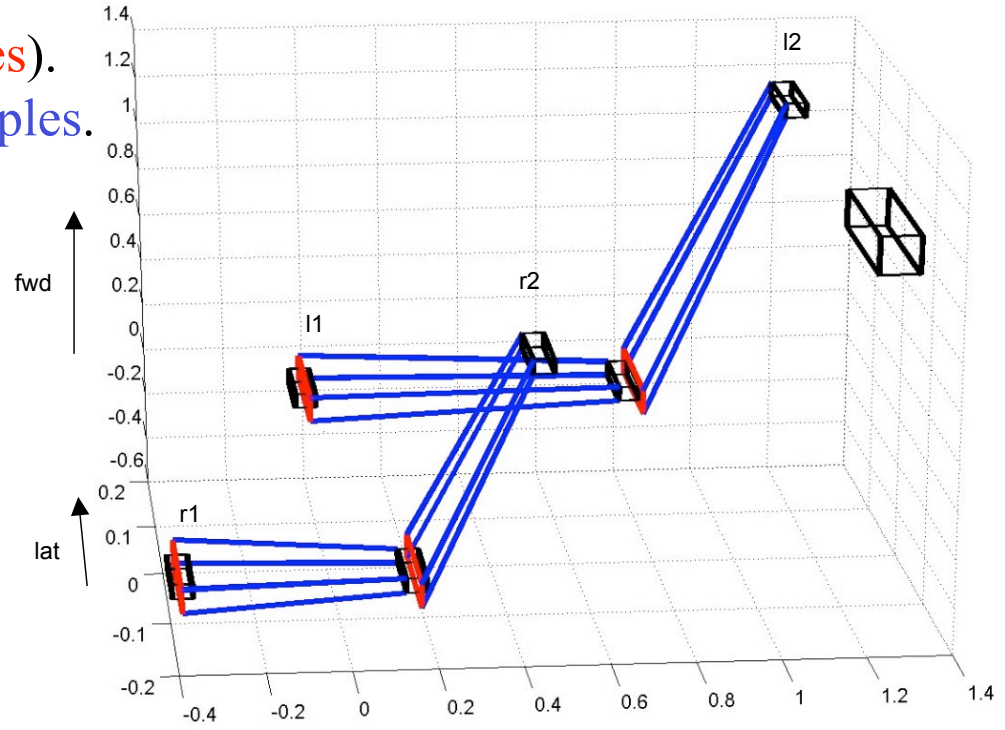
Feasible trajectories must go through goal regions

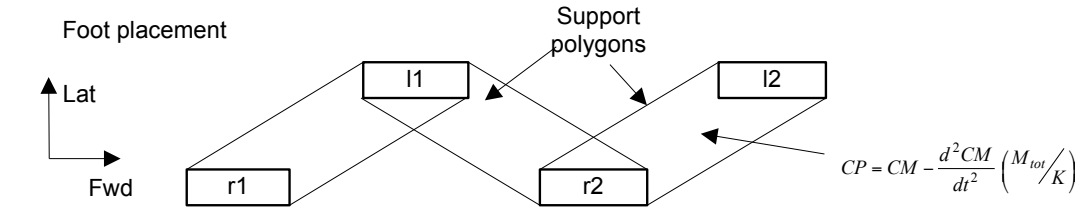


Compile Time:

- Construct all feasible trajectories (Flow Tubes).
- Learn tubes from examples.

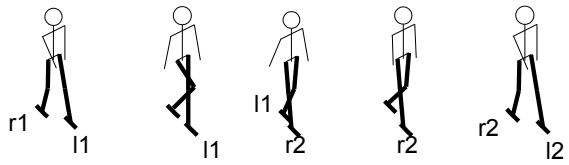






[Hofmann & Williams, AAAI 06; ICAPS 06]

Feasible trajectories must go through goal regions.

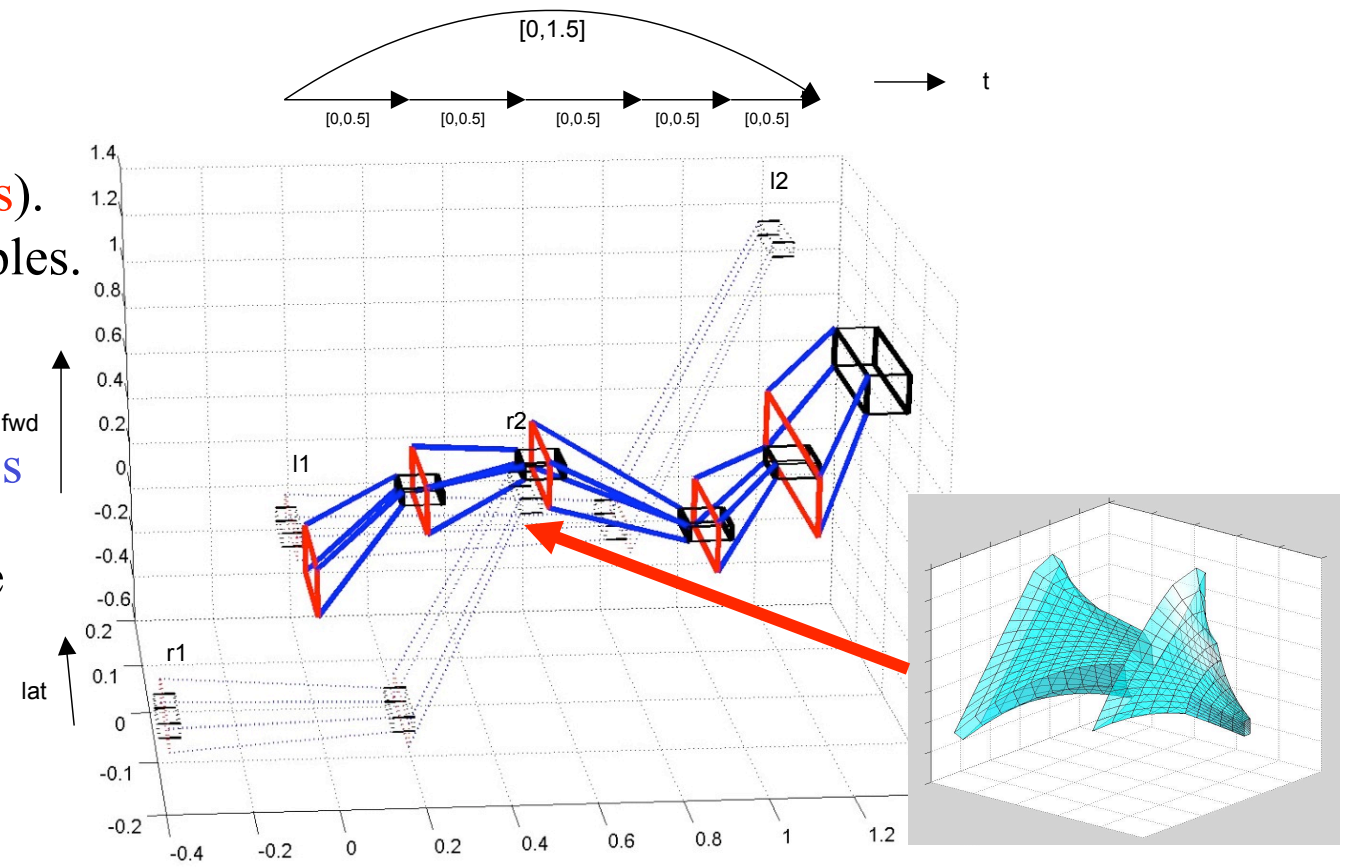


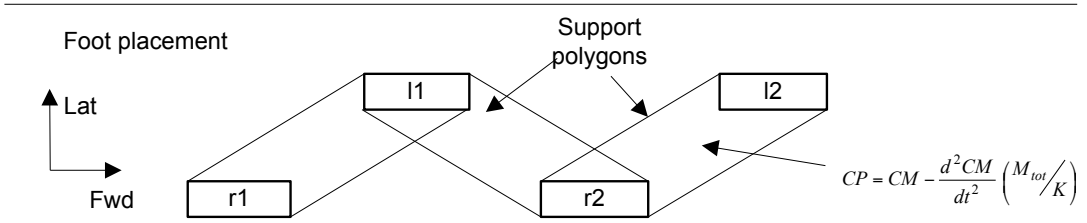
Compile Time:

- Construct all feasible trajectories (Flow Tubes).
- Learn tubes from examples.

Dynamics couples through center of mass

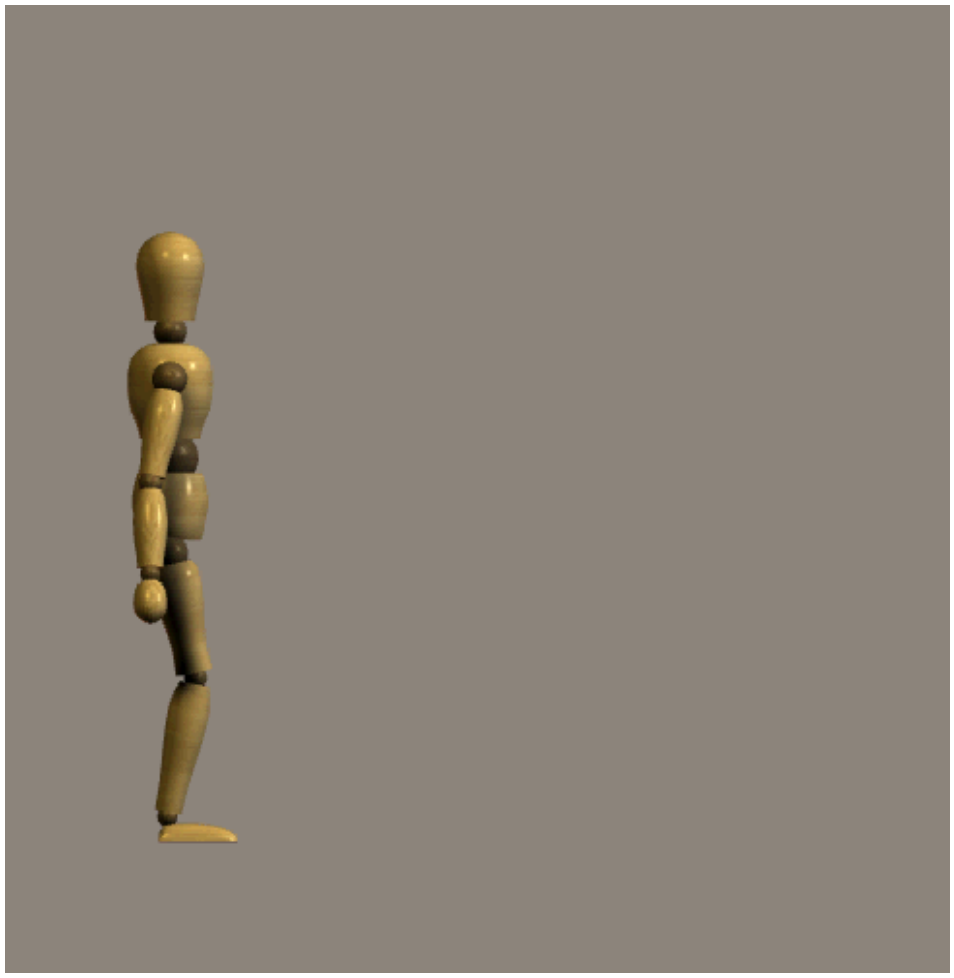
- Construct all feasible schedules for goals.



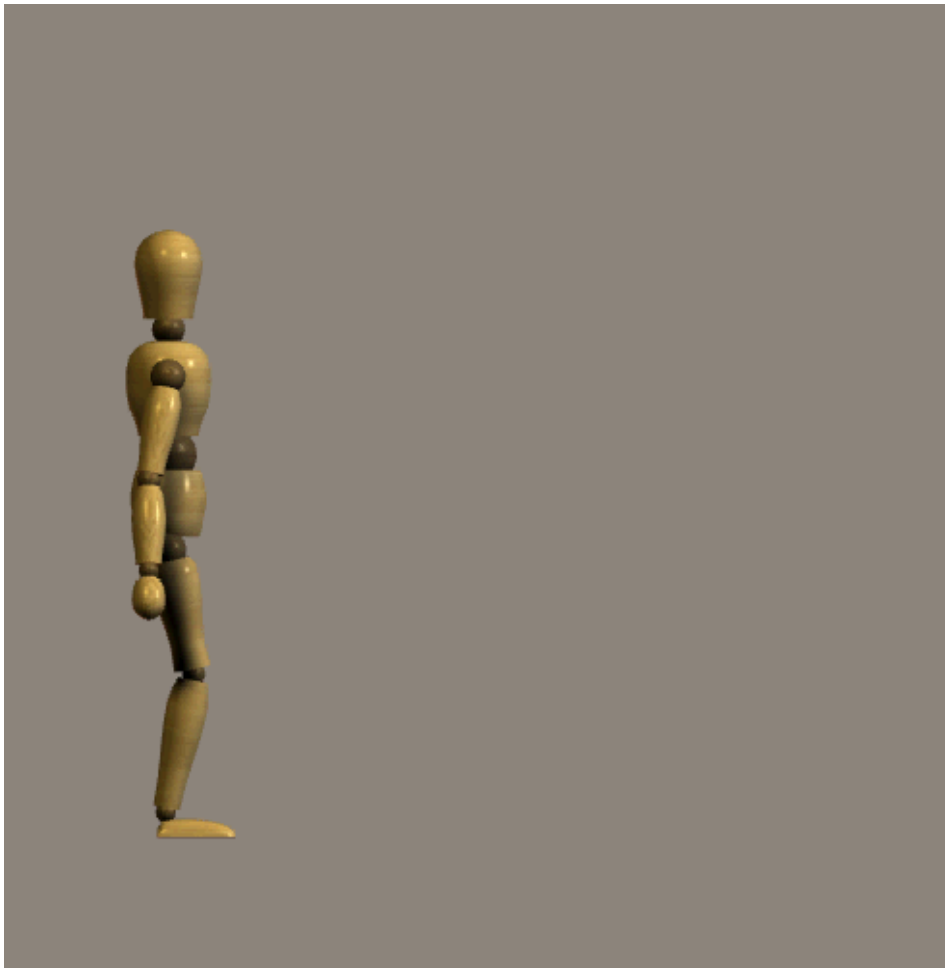


[Hofmann & Williams, AAAI 06; ICAPS 06]

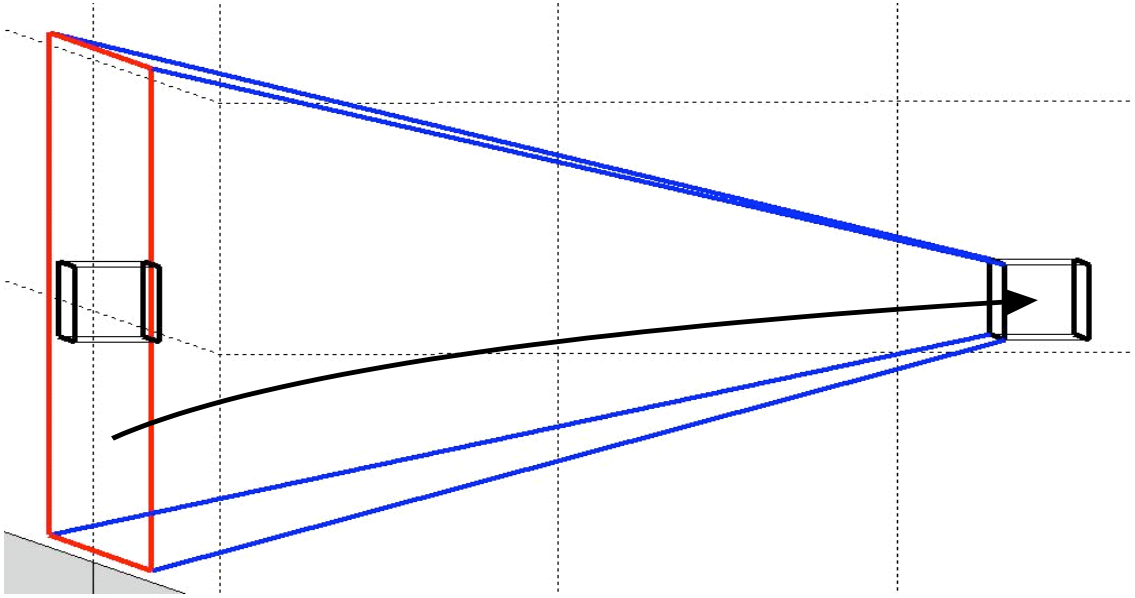
Robustness Requires Temporal Synchronization



Disturbance without temporal coordination



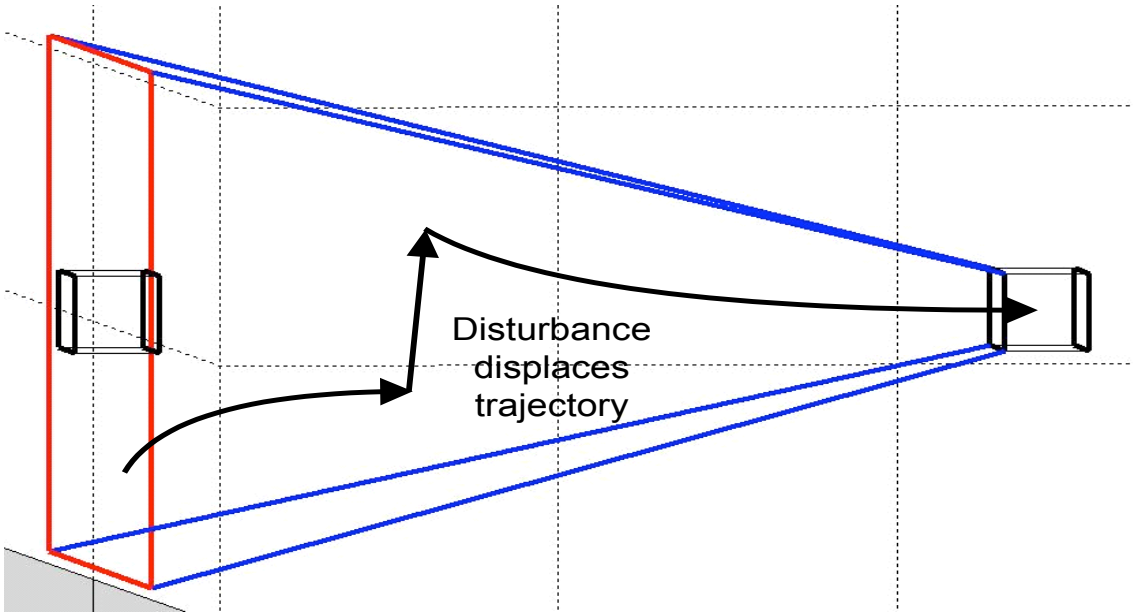
Disturbance with temporal coordination



Execution:

1. Select enabled tube.
2. Schedule goal arrival.
3. Execute control policy until goal achieved.

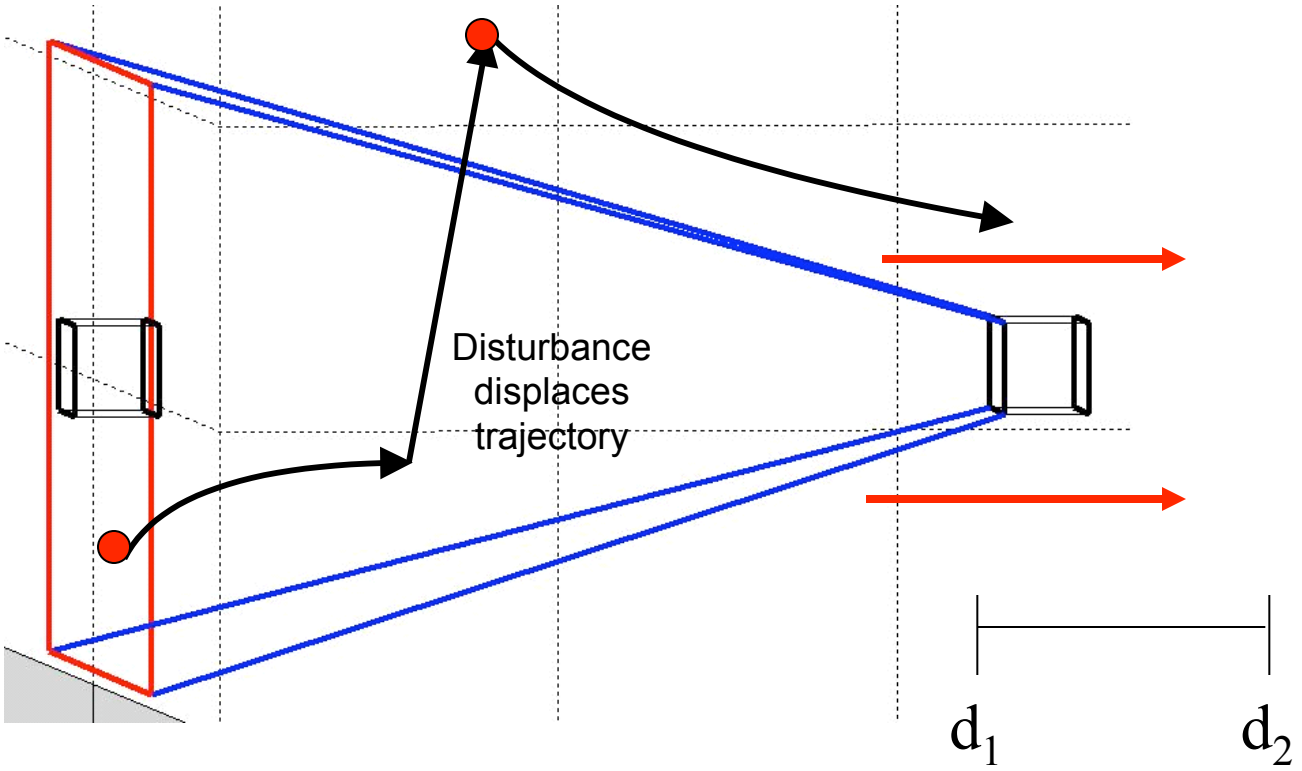
[Hofmann & Williams, AAAI 06; ICAPS 06]



Execution:

1. Select enabled tube.
2. Schedule goal arrival.
3. Execute control policy until goal achieved.
4. If displaced from tube, adjust control parameters or schedule.

[Hofmann & Williams, AAAI 06; ICAPS 06]

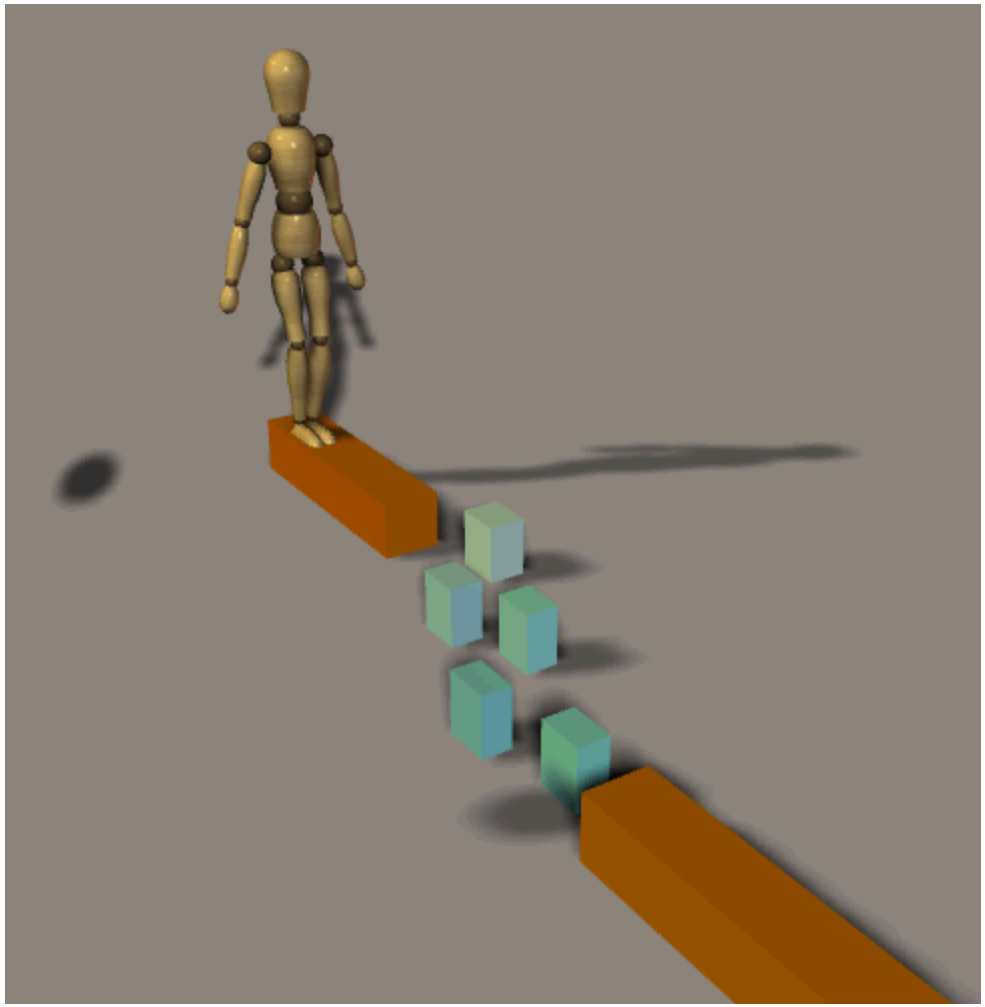


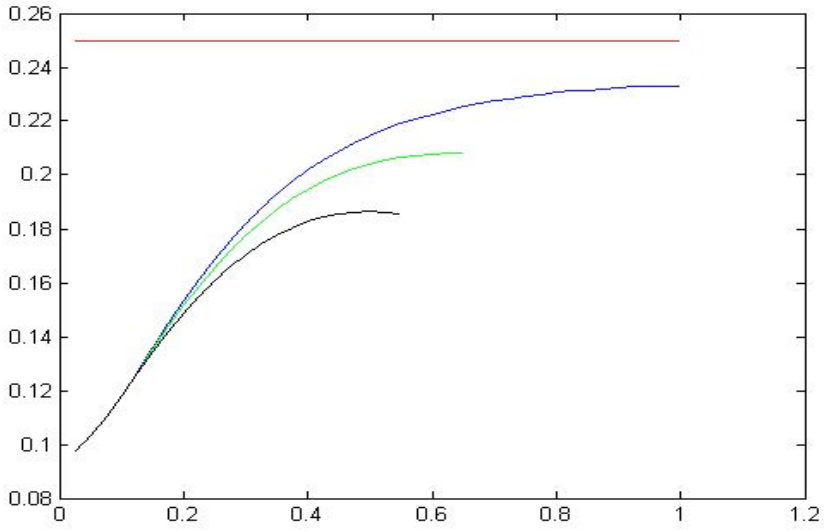
Execution:

1. Select enabled tube.
2. Schedule goal arrival.
3. Execute control policy until goal achieved.
4. If displaced from tube, adjust control parameters or schedule.
   - May require synchronization with other activities.
   - If unschedulable, switch plan.

[Hofmann & Williams, AAAI 06; ICAPS 06]

Compliance Results





Lateral CM with push disturbance

- Blue – 40 N
- Green – 35 N
- Black – 25 N
- Red – Max allowed displacement

Robust, Model-based Execution (1): Coordination and Dynamic Scheduling

- Robust, model-based execution of time critical tasks.
- Task coordination through dynamic scheduling.
- Task coordination for under-actuated systems.
- Task coordination for multi-robot systems.

A Good Human Teammate

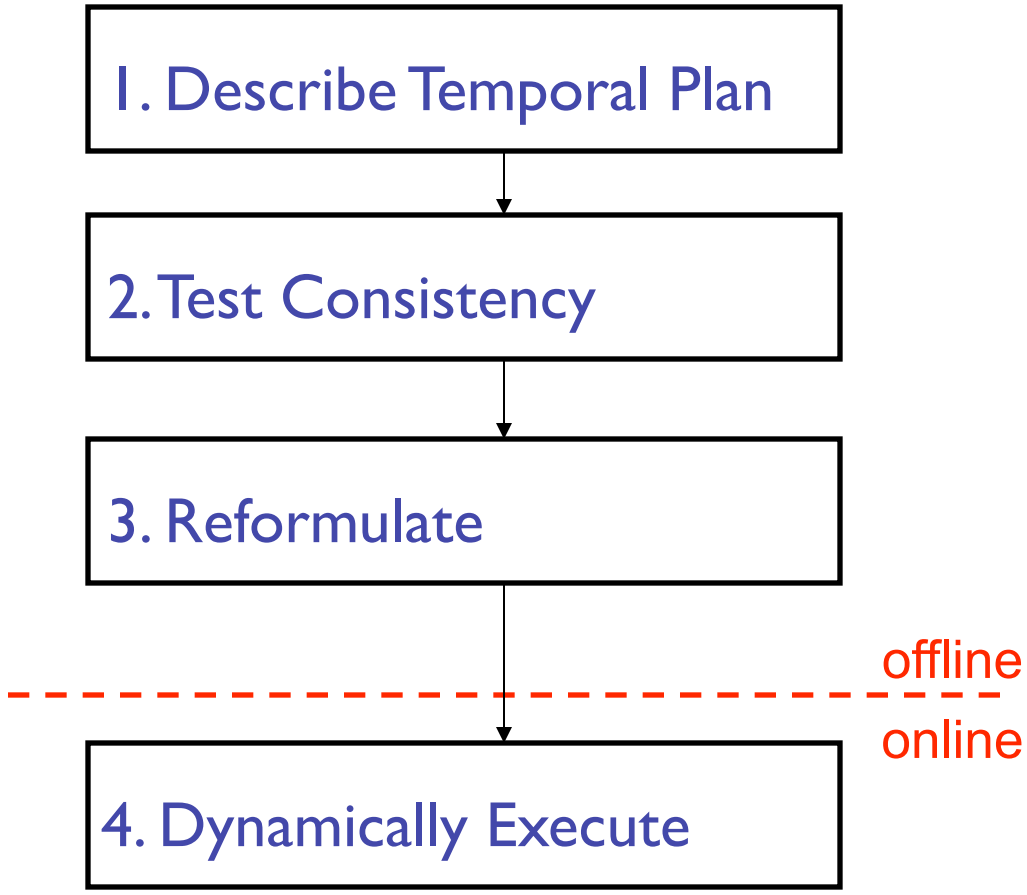
An effective Scrub Nurse:

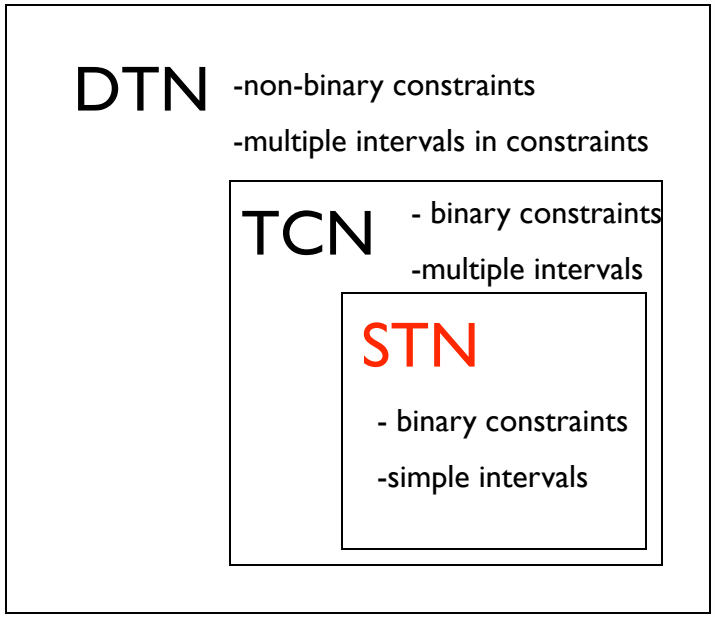
- works hand-to-hand, face-to-face with surgeon,
- assesses and anticipates needs of surgeon,
  - provides tools and assistance in order needed,
- responds quickly to changing circumstances,
- responds quickly to surgeon's cues and requests.

[Shah Ph D MIT]

To Execute a Temporal Plan

Part II: Schedule Online





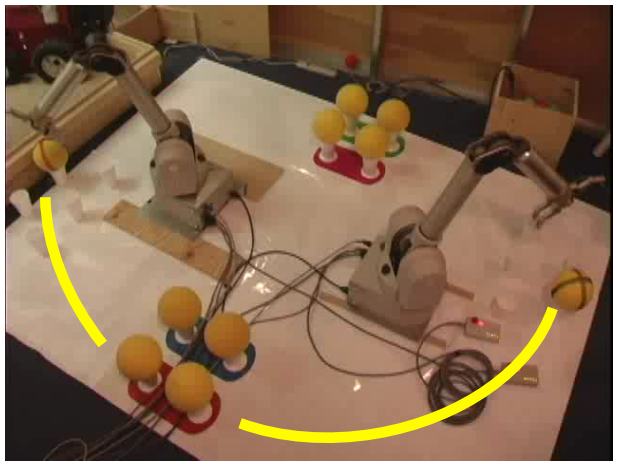
Exploit:

- Handling of Uncertainty
  (STNU, TPNU, TCNU, DTNU).
- Handling of Choice
  (TPN, TCN, DTN)

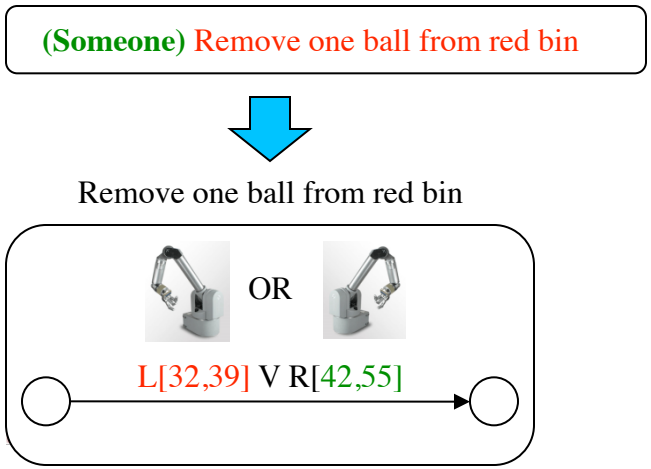
Tsamardinos, Pollack, M. Ganchev, ECP 01]
[Shah, Conrad, Williams ICAPS 09]
[Conrad, Shah, Williams ICAPS 09]

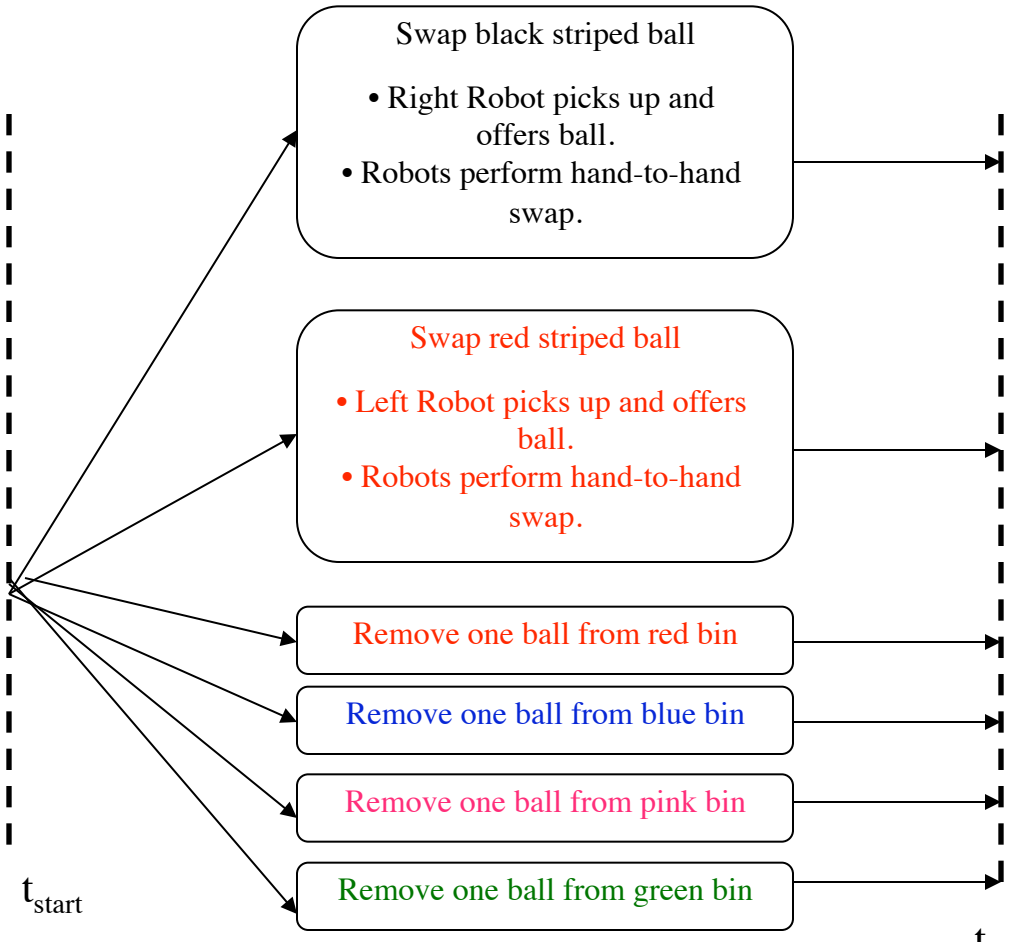
Multi-Robot Teamwork

Model-based Embedded & Robotic Systems



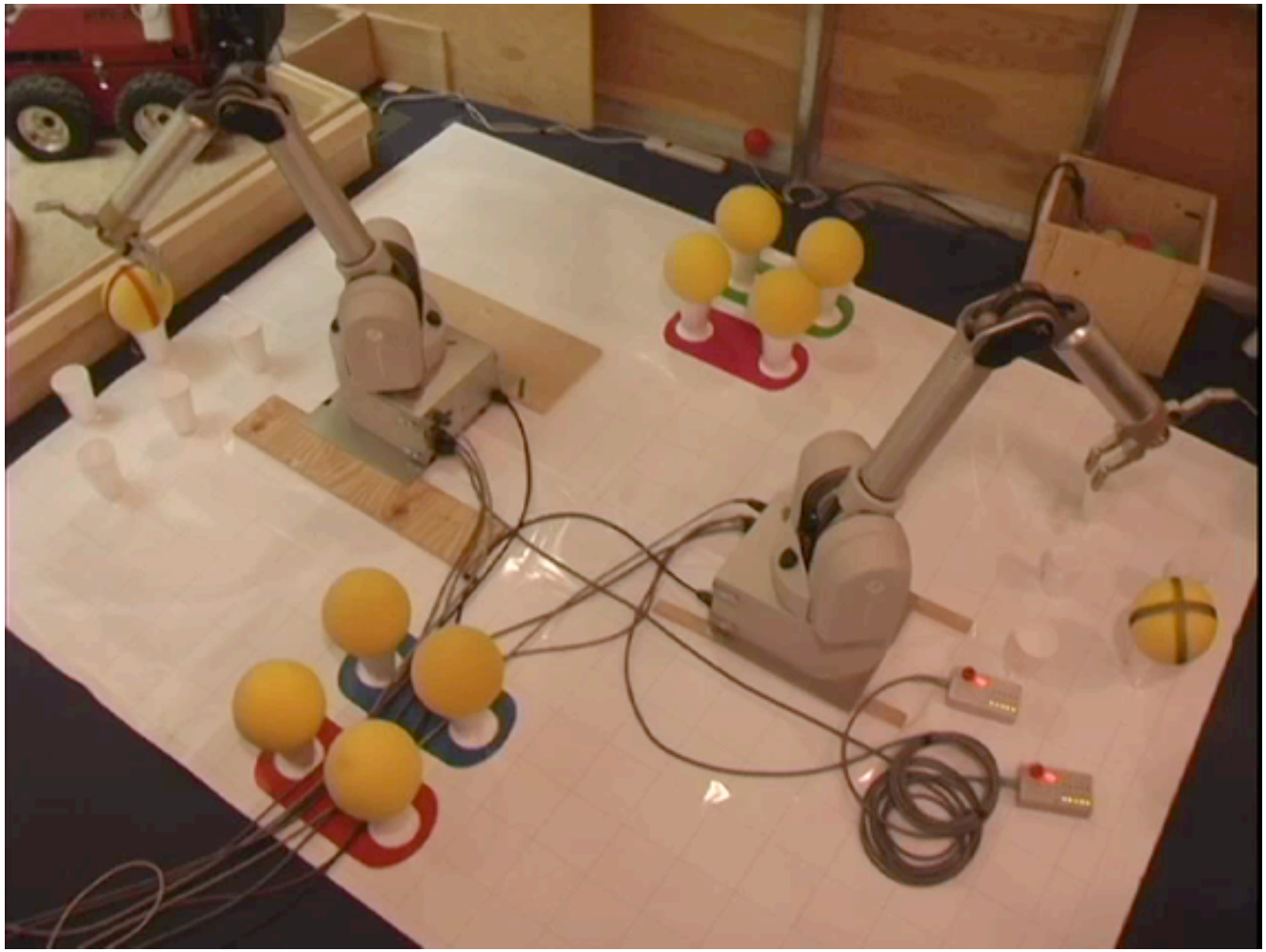
Agents choose and schedule activities





CSAIL

Multi-Robot Teamwork



- Off-nominal
- Partner adapts in response to teammate's failure.

Leader & Assistant

Model-based Embedded & Robotic Systems

Assistant waits to see what Leader will do before acting.

Leader

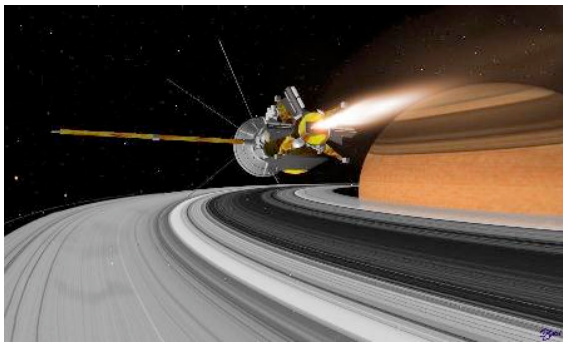
Assistant

Idea: model leader durations and assignments as uncontrollable (TPNU).

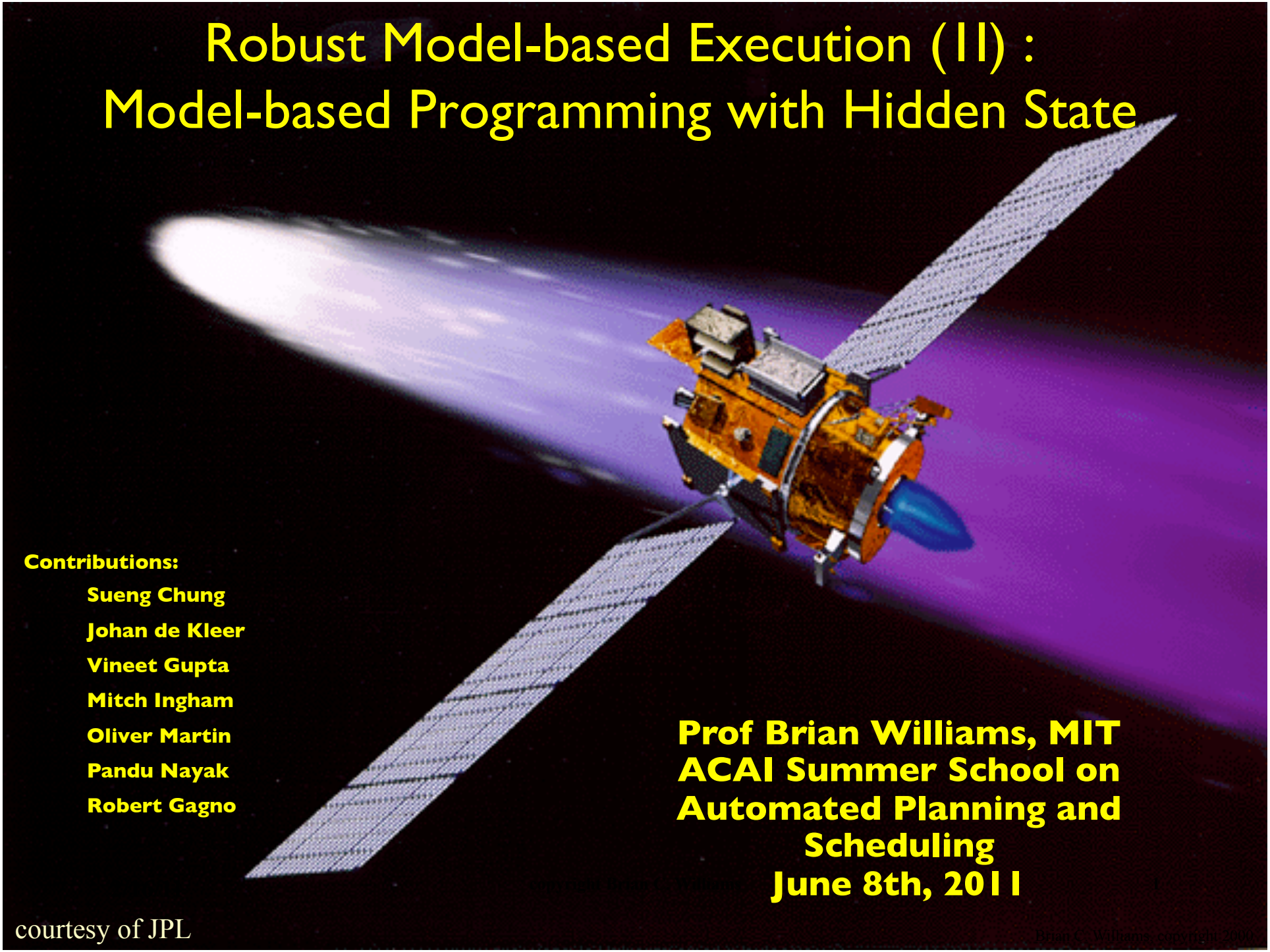
Model-based Execution

The development of autonomous systems that robustly perform complex tasks.

- Goal-directed: Tasks described qualitatively in terms of time-evolved goals.
- Real-time Decisions: Tasks executed using real-time decision making algorithms, based on observations.
- Model-based: Operates on heterogeneous models of the robot, user and environment.



Robust Model-based Execution (II) : Model-based Programming with Hidden State



Contributions:

Sueng Chung
Johan de Kleer
Vineet Gupta
Mitch Ingham
Oliver Martin
Pandu Nayak
Robert Gagno

Prof Brian Williams, MIT
ACAI Summer School on Automated Planning and Scheduling
June 8th, 2011

courtesy of JPL

Readings

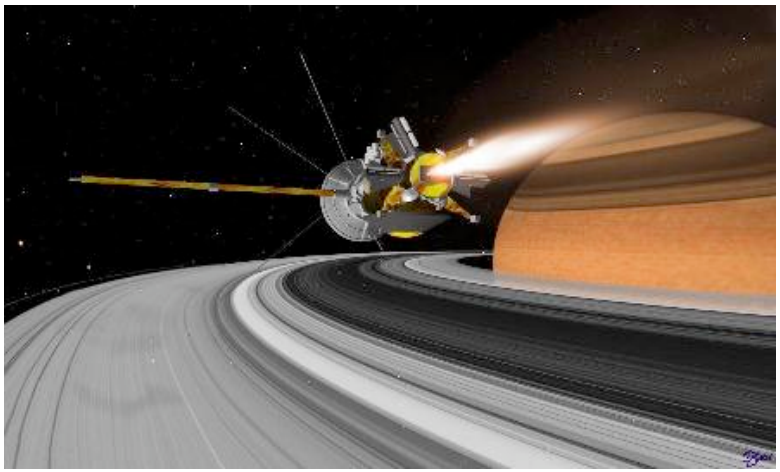
- Google “MIT OCW 16.412 Cognitive Robotics”
- mers.csail.mit.edu, click “Publications”
- Williams, B. C. et al., “Model-based Programming of Intelligent Embedded Systems and Robotic Explorers," *Proceedings of the IEEE* 91, no. 1, Special Issue on Modeling and Design of Embedded Software, pp. 212-237, 2003.
- B. C. Williams, M. Ingham, S. Chung, P. Elliott, and M. Hofbaur,"Model-based Programming of Fault-Aware Systems," *AI Magazine, vol. 24, no. 4, pp. 61-75, 2004.*
- B. C. Williams, and R. Ragno, "Conflict-directed A* and its Role in Model-based Embedded Systems," Special Issue on Theory and Applications of Satisfiability Testing, *Journal of Discrete Applied Math, January 2003.*
- J. de Kleer and B. C. Williams, "Diagnosing Multiple Faults," *Artificial Intelligence*, 32:100-117, 1987.
- Martin, O., B. C. Williams and M. Ingham, "Diagnosis as Approximate Belief State Enumeration for Probabilistic Concurrent Constraint Automata", in Proceedings of the Twentieth National Conference on Artificial Intelligence, Pittsburgh, PA, July 2005.
- Brian C. Williams and P. Pandurang Nayak, "A Reactive Planner for a Model-based Executive," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997, pp. 1178-85.
- L. Blackmore, S. Funiak, and B. C. Williams, “A Combined Stochastic and Greedy Hybrid Estimation Capability for Concurrent Hybrid Models with Autonomous Mode Transitions,” Journal of Robotic and Autonomous Systems, July, 2007.

Approach: Model-based Programming and Execution

- An embedded programming language elevated to operations on hidden state, and
- A language executive that achieves robustness by reasoning over constraint-based models.

Today: Coordination and dynamic scheduling.

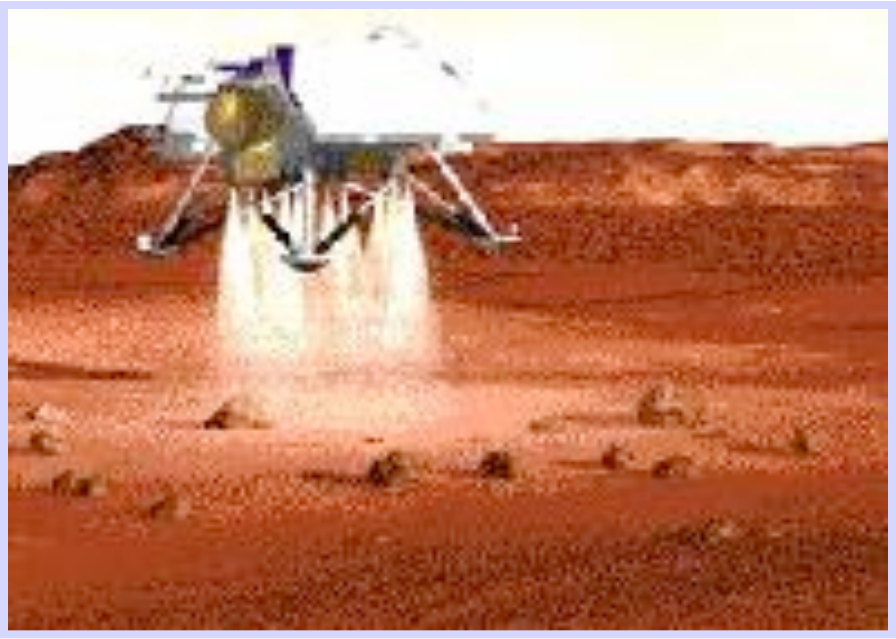
Wednesday: Model-based programming with hidden state.



Robust, Model-based Execution (2): Model-based Programming w Hidden States

- Model-based programming and execution.
- Control sequencing.
- Mode estimation.
- Mode reconfiguration and reactive planning.

Mars Polar Lander Failure



Leading Diagnosis:

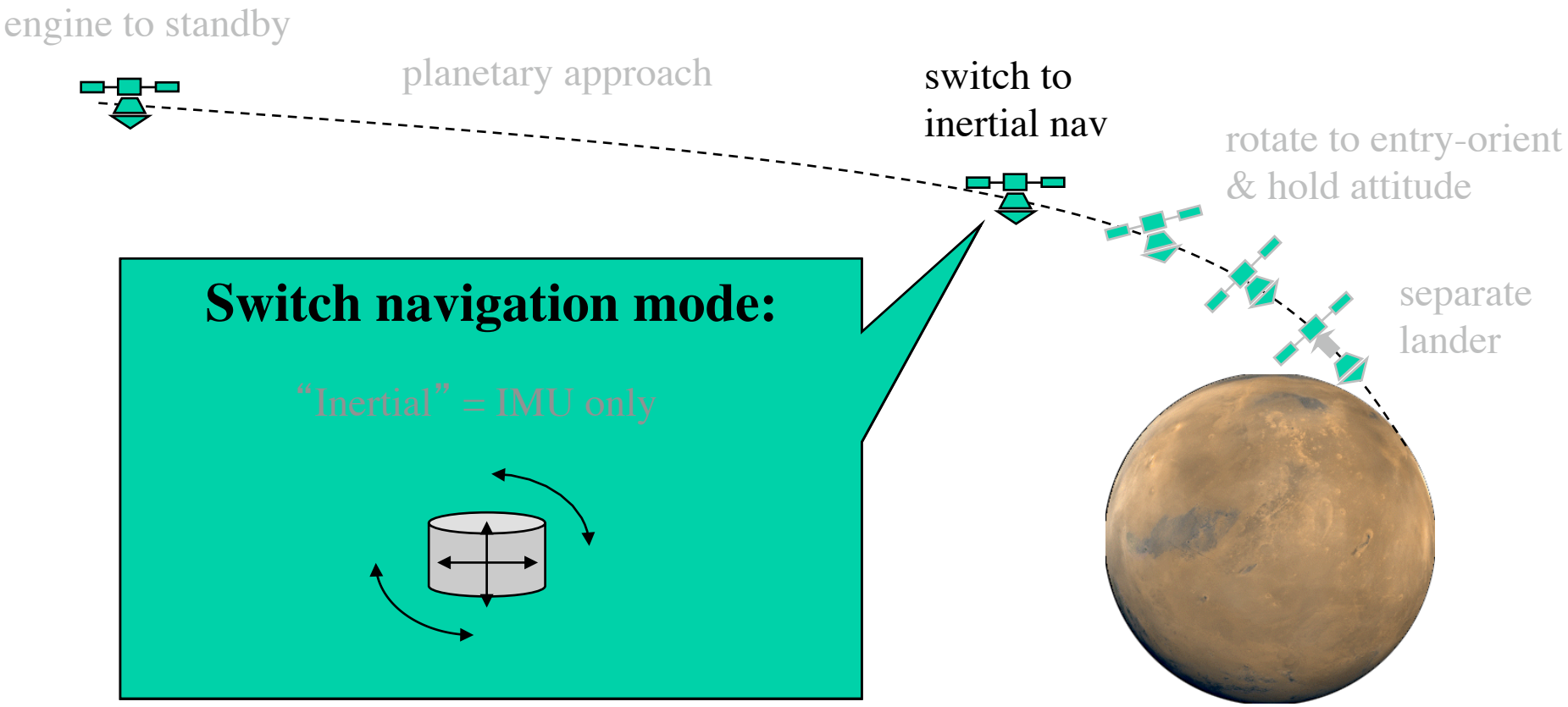
- Legs deployed during descent.
- Noise spike on leg sensors latched by software monitors.
- Laser altimeter registers 50ft.
- Begins polling leg monitors to determine touch down.
- Latched noise spike read as touchdown.
- Engine shutdown at ~50ft.

Fault Aware Systems:
Systems that reason and coordinate on the fly from models

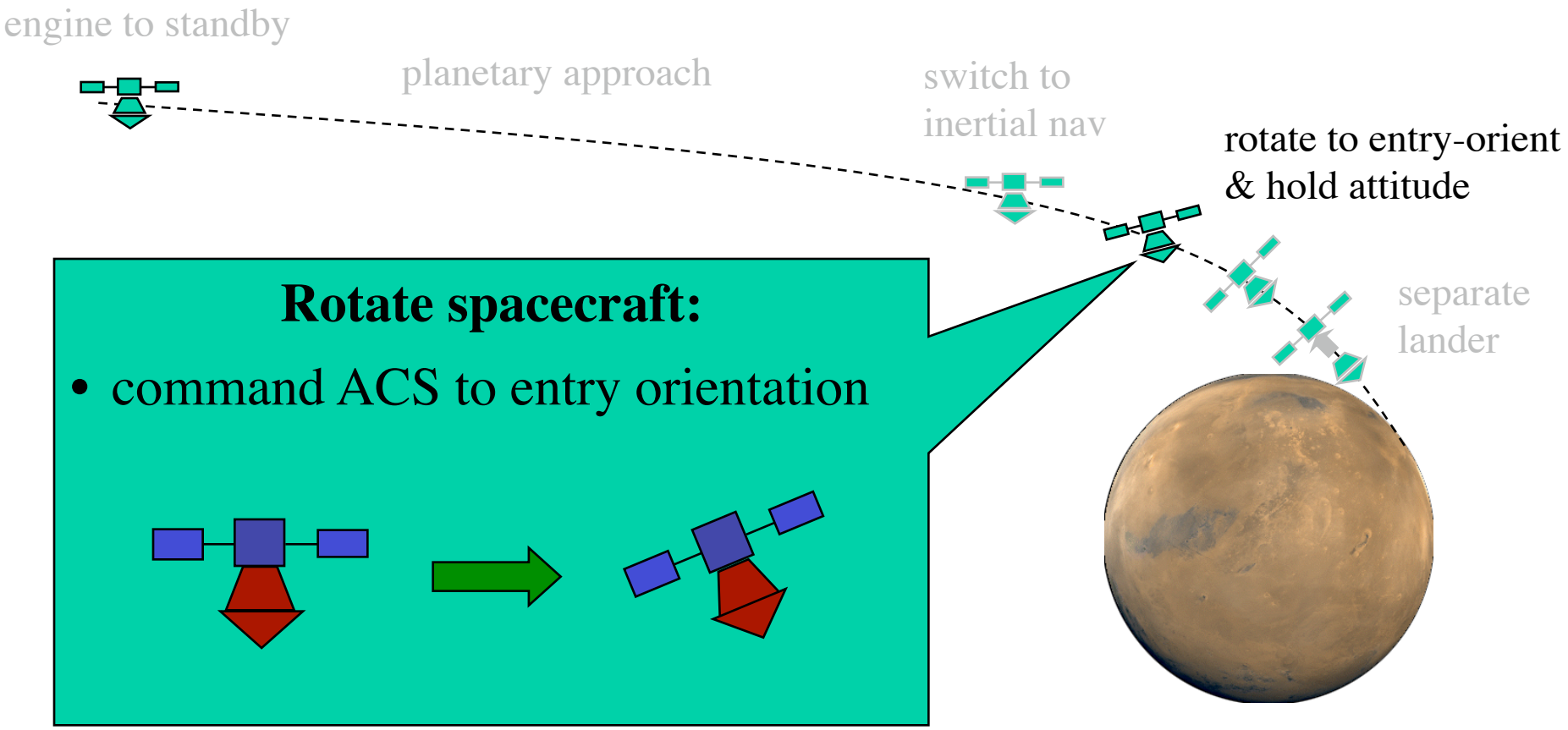
Model-based Programming:
Programming of fault-aware systems

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

Mission Storyboards Specify Evolving States



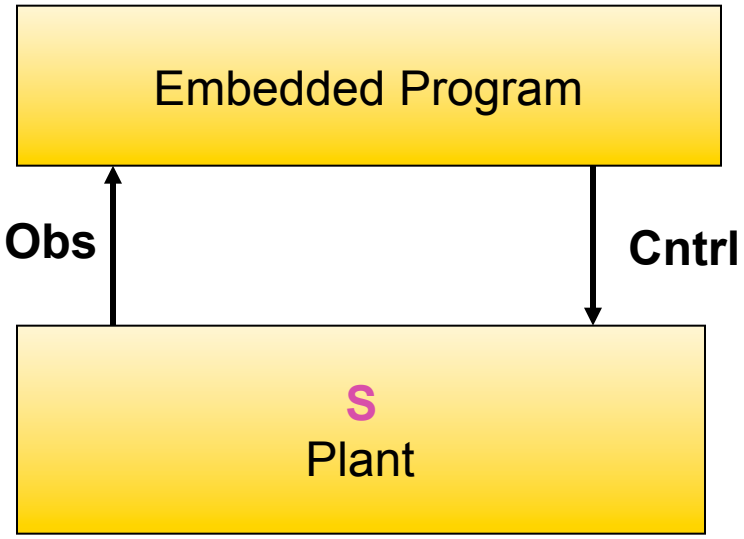
Mission Storyboards Specify Evolving States



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

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

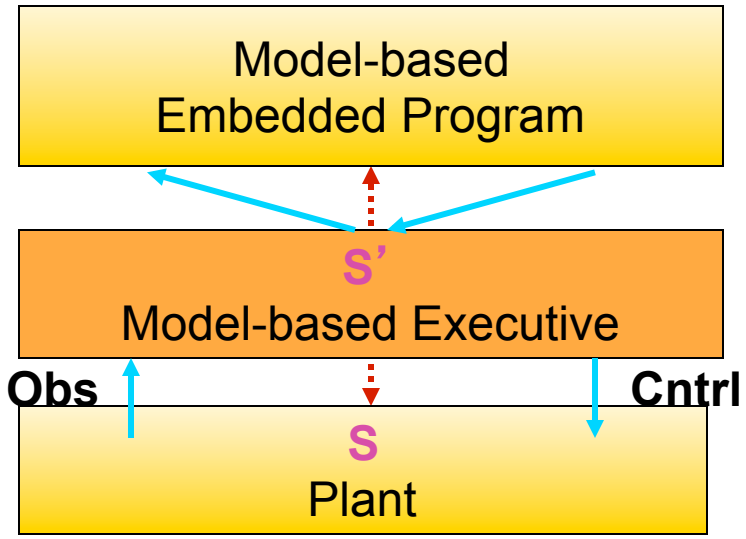
- Read sensors
- Set actuators



Programmer maps between state and sensors/actuators.

Model-based programs evolve abstract states through direct interaction:

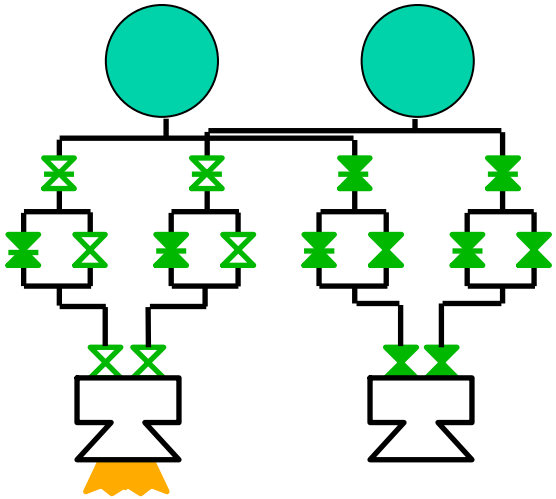
- Read abstract state
- Write abstract state



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

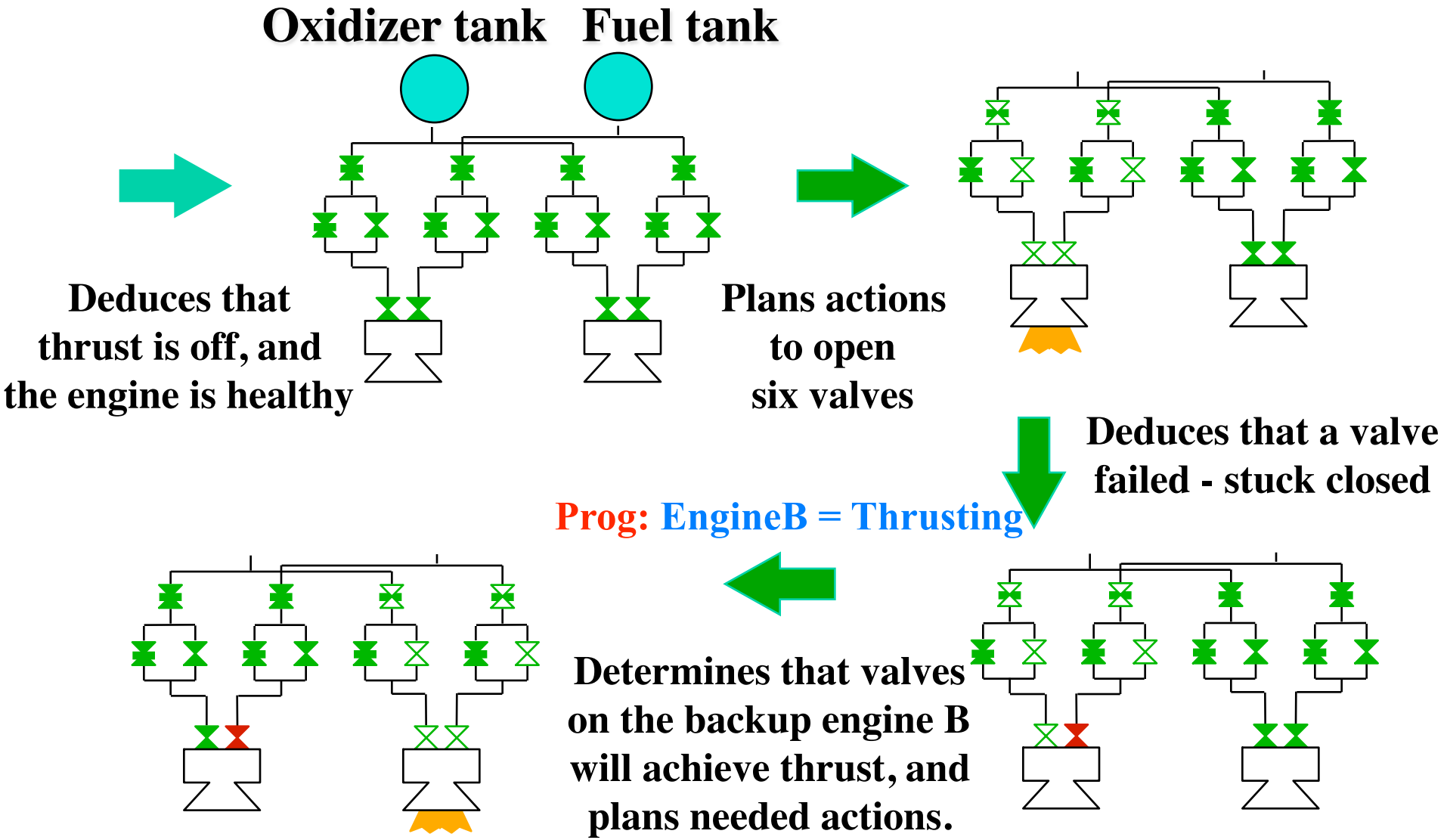
Model-based Programming of a Saturn Orbiter

Turn camera off and engine on




```
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}
```

The program assigns EngineA = Thrusting, and the model-based executive . . . .



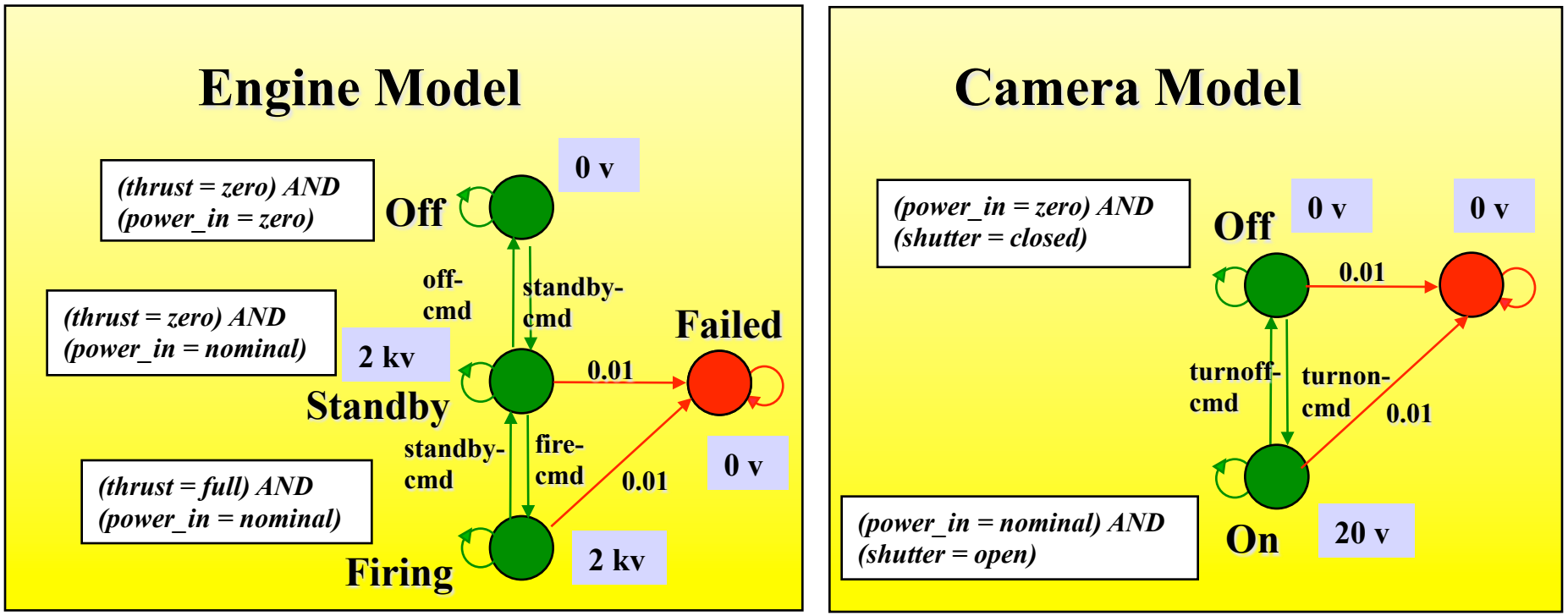
Plant Model: Probabilistic Constraint Automata (PCA)

component modes…

described by finite domain constraints on variables…

guarded deterministic and probabilistic transitions

cost / reward & prior distribution



one per component … operating concurrently

[Williams & Nayak 95, Williams et al. 01]

Modeling Reactive Behavior

Design Features	RMPL constructs:
• state constraints	• c[s]
• conditional execution	• If c[s] next A
• preemption	• Unless c[s] next A
• full concurrency	• A, B
• Iteration	• Always A
• Flexible time	• [l, u] A
• Reward	• Choose with reward
• Probability	• Choose with probability

A generalization of TCC combinators
[Saraswat, Gupta, et al.]

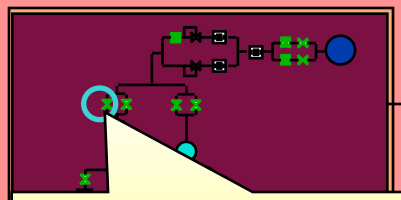
RMP

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

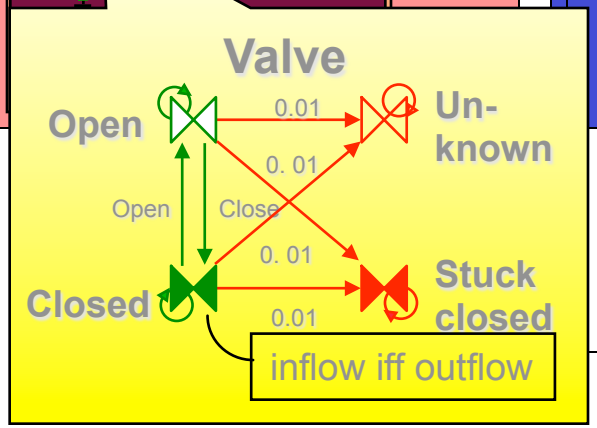


State estimates

State goals

Tracks likely plant states

Tracks least cost goal states



Observations

Commands

Plant

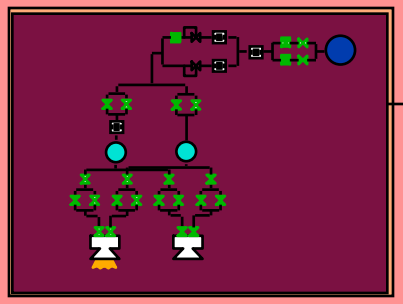
RMPL Model-based Program

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

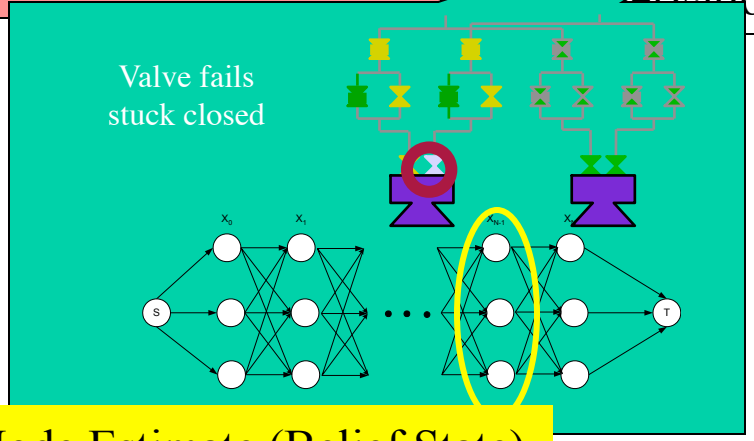
State goals

Mode Estimation:
Tracks likely States

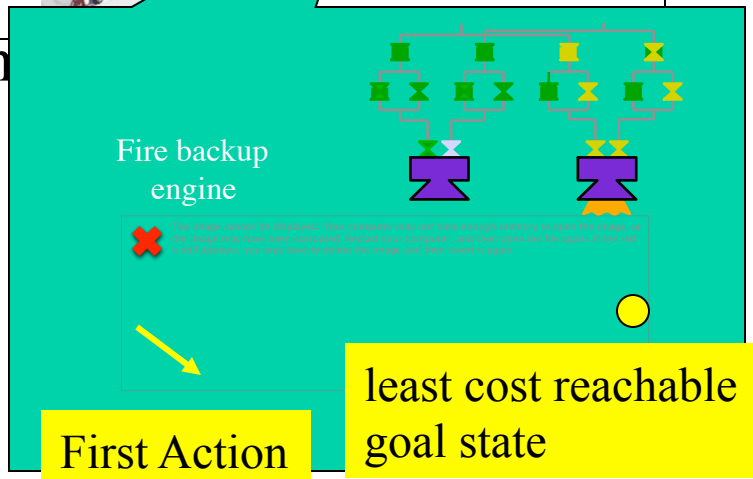
Mode Reconfiguration:
Tracks least-cost state goals

Commands

Plan



Mode Estimate (Belief State)



First Action

least cost reachable goal state

Mode Estimation:

Select a most likely set of next component modes that are consistent with the model and past observations.

Mode Reconfiguration:

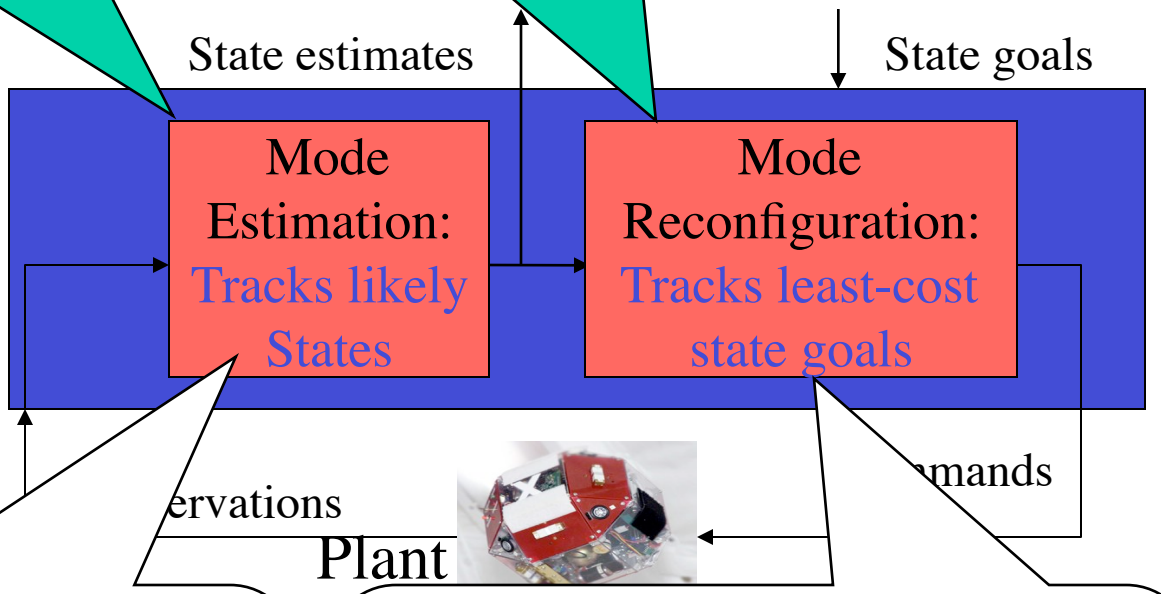
Select a least cost set of commandable component modes that entail the current goal, and are consistent.

Optimal CSP:

arg min f(x)

s.t. C(x) is satisfiable

D(x) is unsatisfiable



arg min $P_t(Y| Obs)$

s.t. $\Psi(X,Y) \wedge O(m')$ is consistent

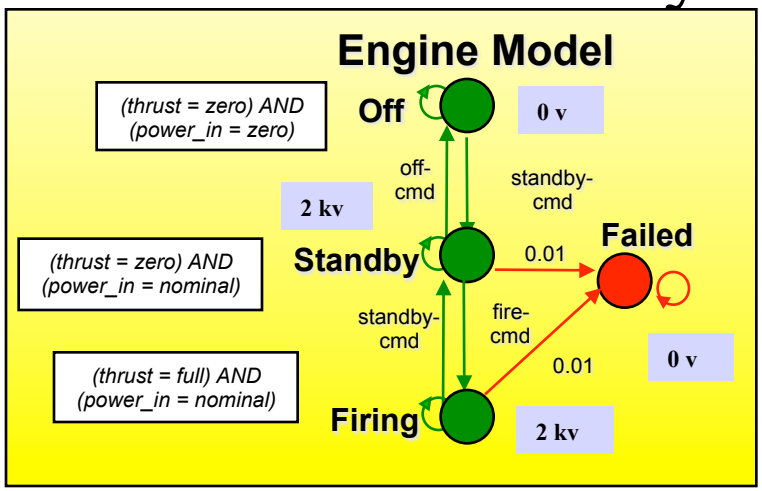
arg max $R_t(Y)$

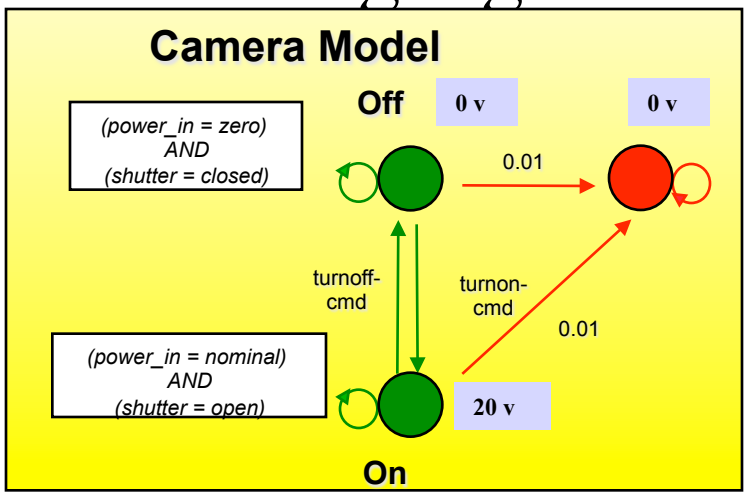
s.t. $\Psi(X,Y)$ entails $G(X,Y)$

s.t. $\Psi(X,Y)$ is consistent

s.t. Y is reachable

Variants on Probabilistic Constraint Automata define a Family of RMPL Languages





- Complex, discrete behaviors
  - modeled through concurrency, hierarchy and timed transitions.
- Anomalies and uncertainty
  - modeled by probabilistic transitions
- Physical interactions
  - modeled by discrete and continuous constraints

Robust, Model-based Execution (2): Model-based Programming w Hidden States

- Model-based programming and execution.
- Control sequencing.
- Mode estimation.
- Mode reconfiguration and reactive planning.

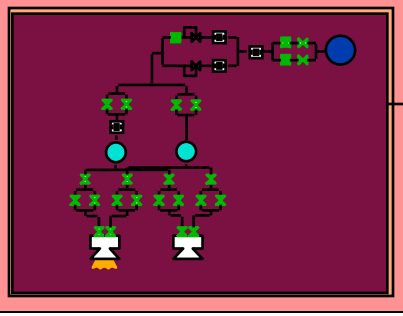
RMI

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}
```

Control Sequencer

System Model



State estimates

State goals

Deductive Controller

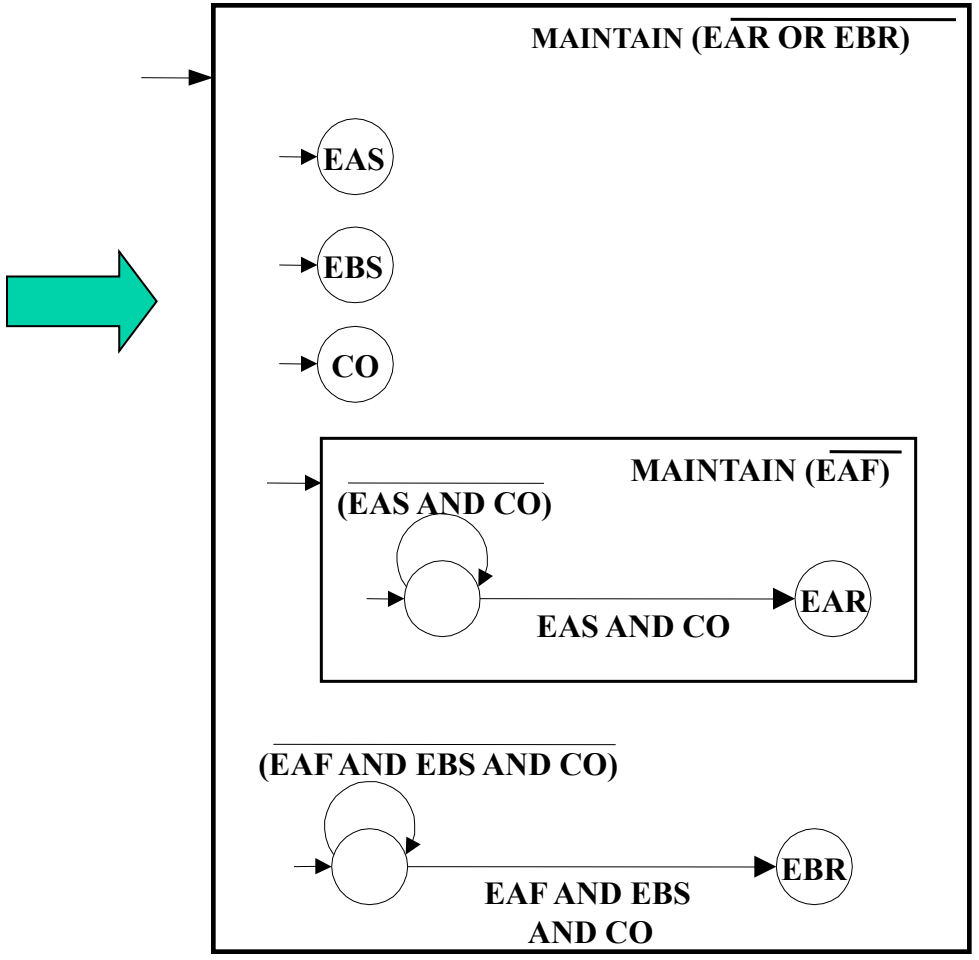
Observations

Commands

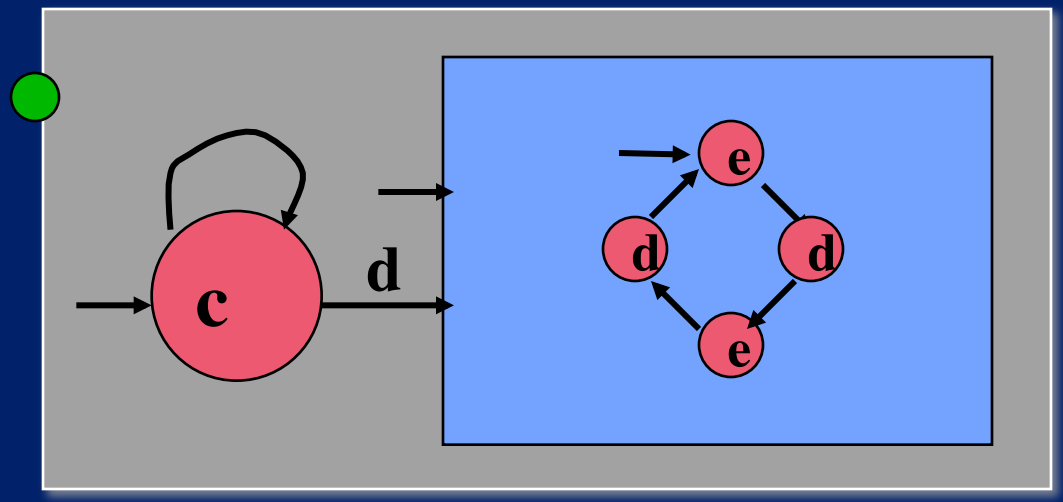
Plant

Map RMPL to Hierarchical Constraint Automaton

```
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}
```



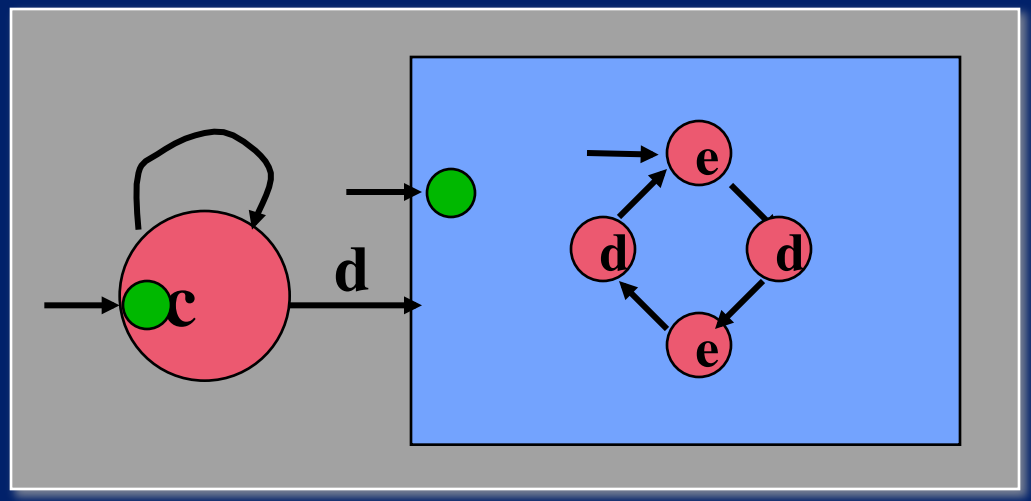
Executing Deterministic HCA



Given marking:

- Mark start locations of all newly marked composite locations.

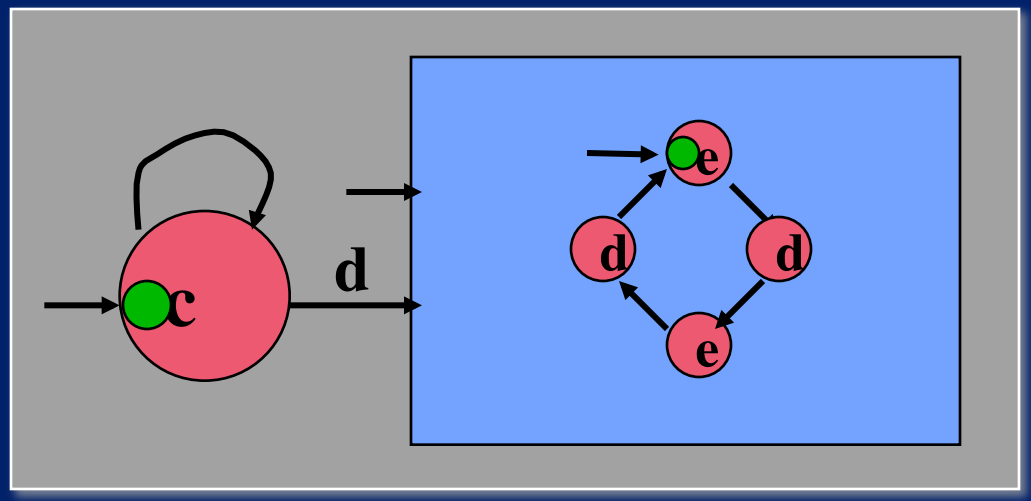
Executing Deterministic HCA



Given marking:

- Mark start locations of all newly marked composite locations.

Executing Deterministic HCA

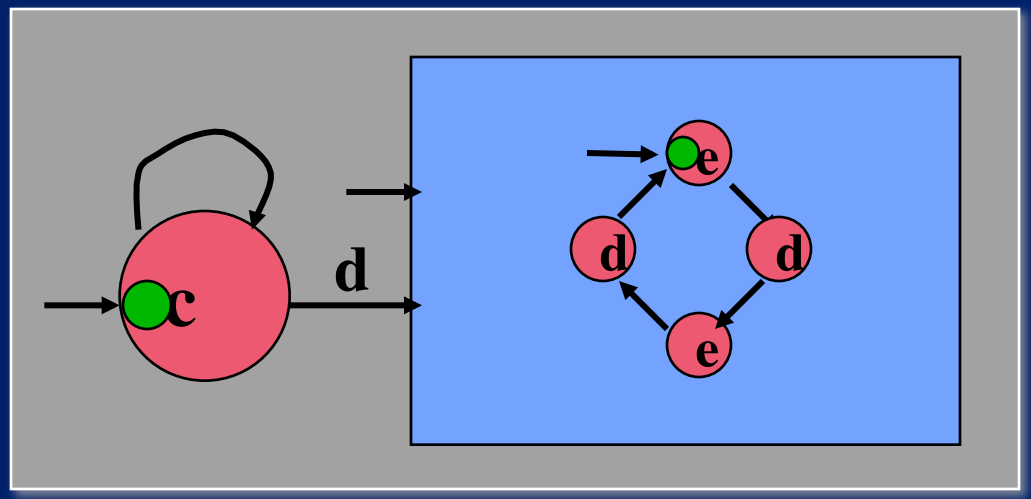


Given marking:

- Mark start locations of all newly marked composite locations.

Executing Deterministic HCA

$\theta = \{c,e\}$



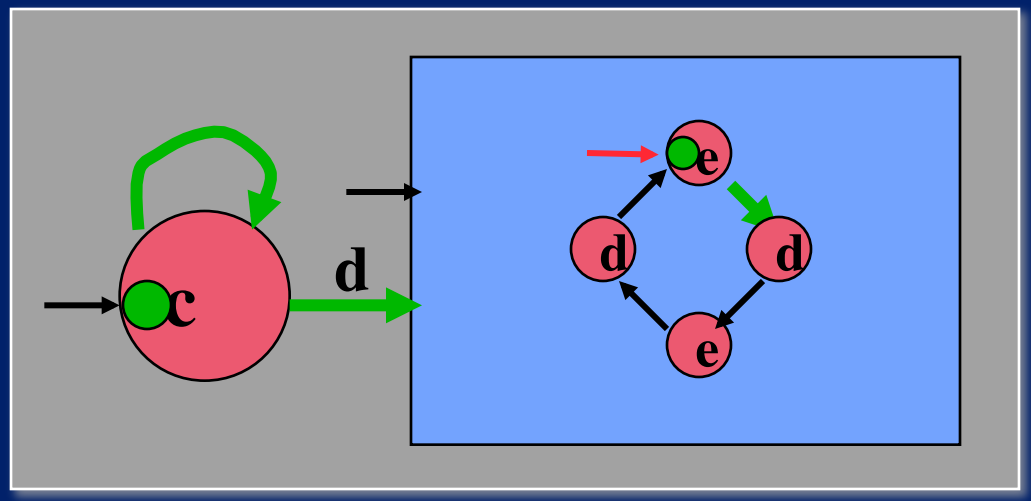
Given marking:

- Mark start locations of all newly marked composite locations.
- Achieve goal constraints θ of marked locations (mode reconfiguration).

Executing Deterministic HCA

θ = {c,e}

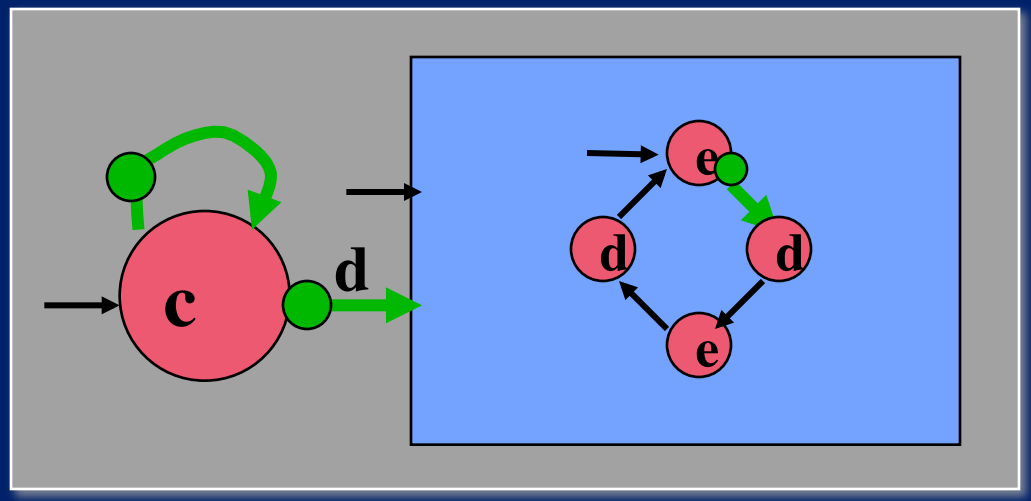
d estimated



Given marking:

- Mark start locations of all newly marked composite locations.
- Achieve goal constraints θ of marked locations (mode reconfiguration).
- Find enabled transitions of marked locations, using model and observations (mode estimation).

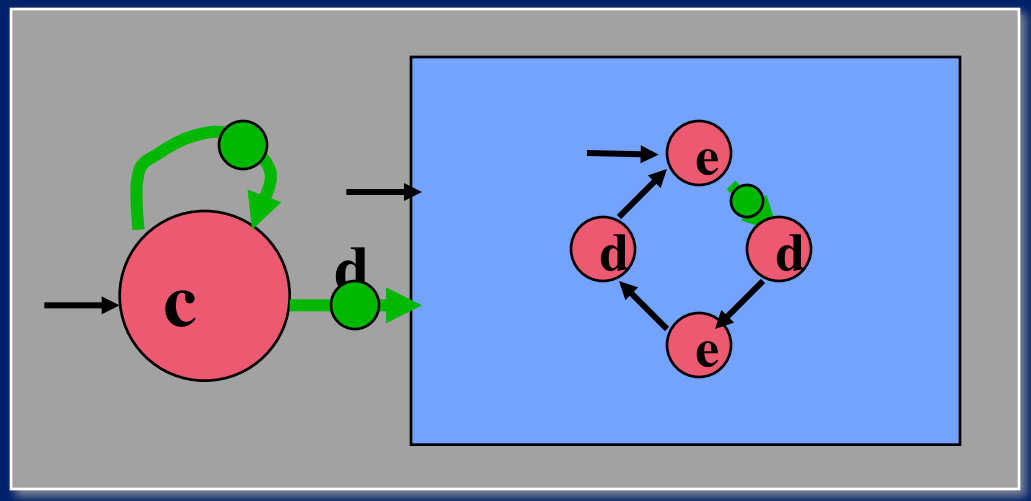
Executing Deterministic HCA



Given marking:

- Mark start locations of all newly marked composite locations.
- Achieve goal constraints θ of marked locations (mode reconfiguration).
- Find enabled transitions of marked locations, using model and observations (mode estimation).
- Take enabled transitions.

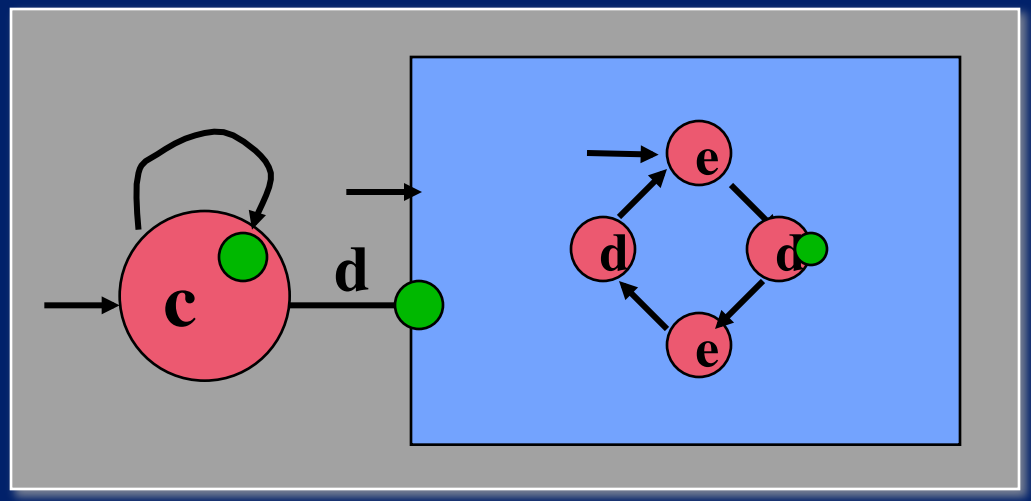
Executing Deterministic HCA



Given marking:

- Mark start locations of all newly marked composite locations.
- Achieve goal constraints θ of marked locations (mode reconfiguration).
- Find enabled transitions of marked locations, using model and observations (mode estimation).
- Take enabled transitions.

Executing Deterministic HCA

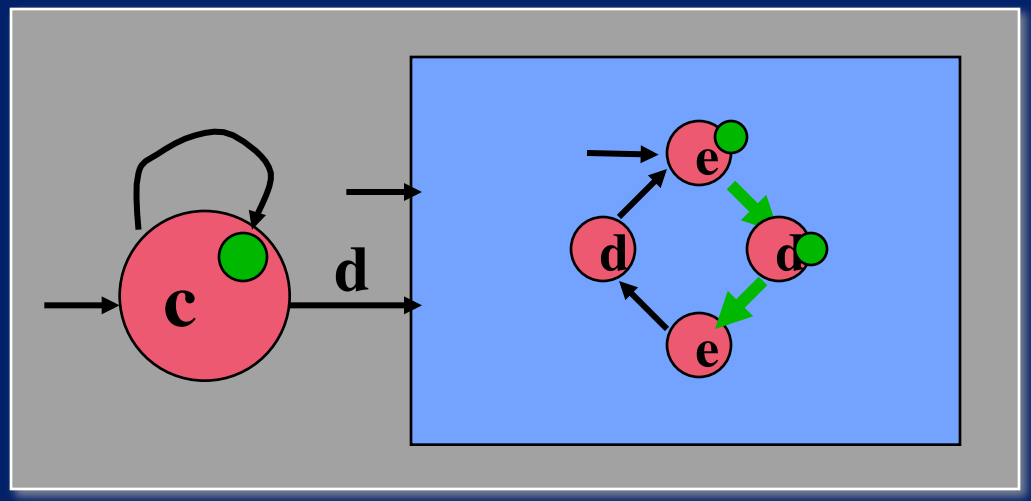


Given marking:

- Mark start locations of all newly marked composite locations.
- Achieve goal constraints θ of marked locations (mode reconfiguration).
- Find enabled transitions of marked locations, using model and observations (mode estimation).
- Take enabled transitions.

Executing Deterministic HCA

$\theta = \{c,d,e\}$



Given marking:

- Mark start locations of all newly marked composite locations.
- Achieve goal constraints θ of marked locations (mode reconfiguration).
- Find enabled transitions of marked locations, using model and observations (mode estimation).
- Take enabled transitions.

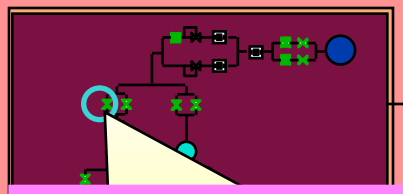
RMF

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))))
```

Control Sequencer

System Model

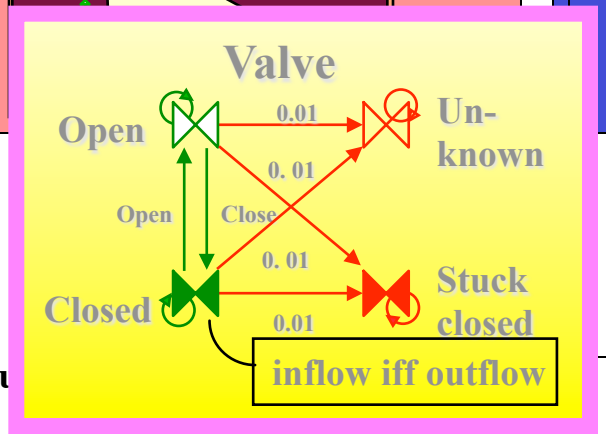


State estimates

State goals

Mode Estimation

Mode Reconfiguration



Observations

Ju

Plant

Optimal CSP:

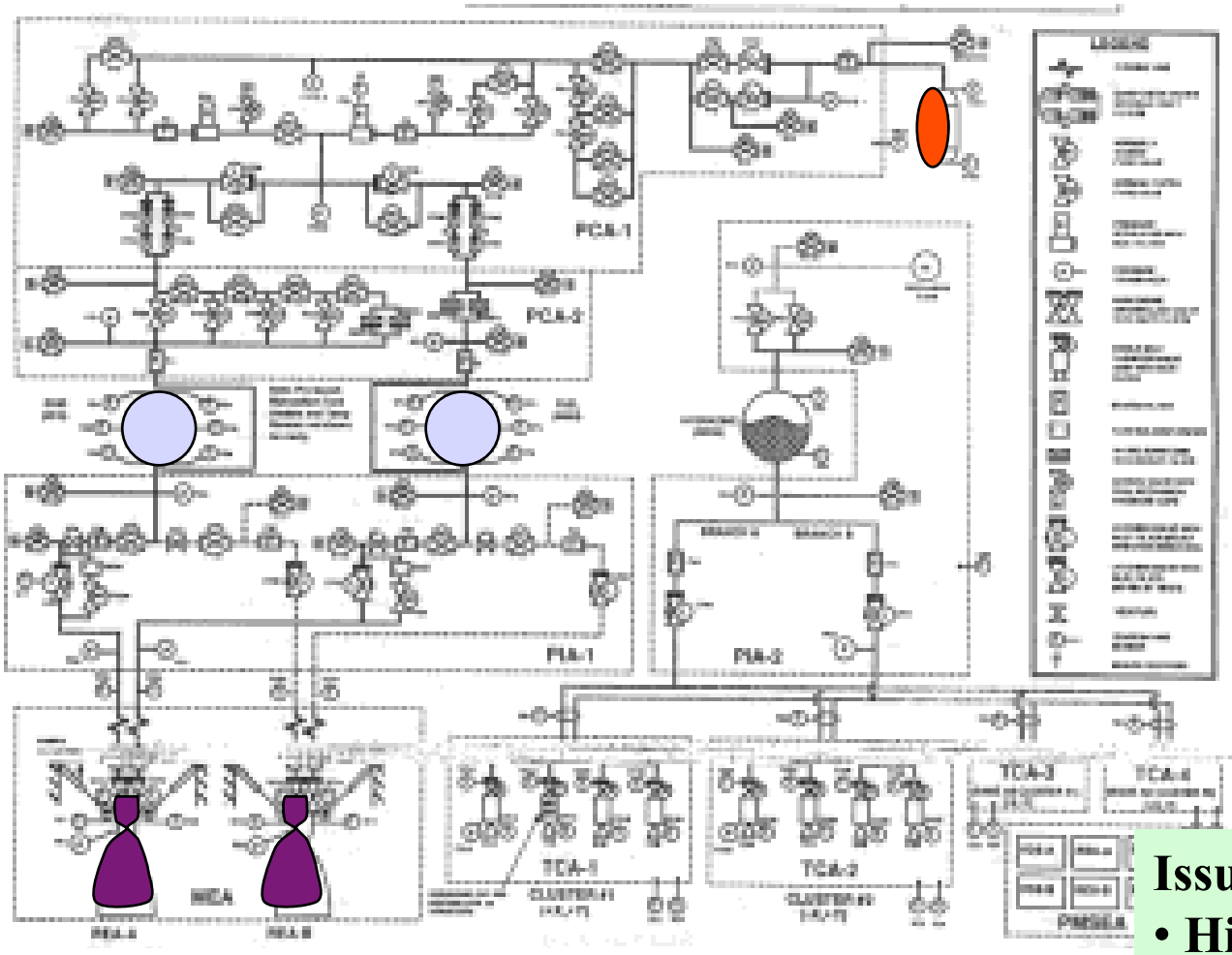
arg min f(x)

s.t. C(x) is satisfiable

D(x) is unsatisfiable

Robust, Model-based Execution (2): Model-based Programming w Hidden States

- Model-based programming and execution.
- Control sequencing.
- Mode estimation.
  - Estimates and kernels.
  - By divide and conquer (GDE).
  - Likely estimates (Conflict-directed A*).
  - Estimating probabilistic constraint automata
- Mode reconfiguration and reactive planning.

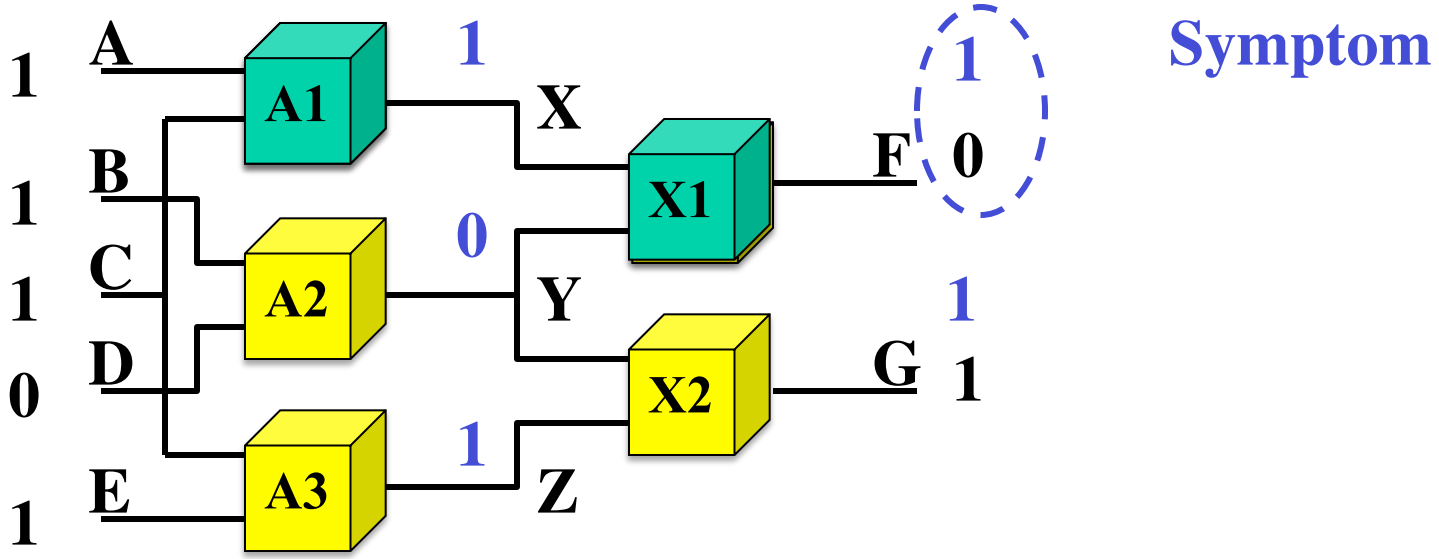


Issues:
- **Hidden failures**
- **Novel failures**
- **Multiple faults**

Model-based Diagnosis

Input: Observations of a system with symptomatic behavior, and a model Φ of the system.

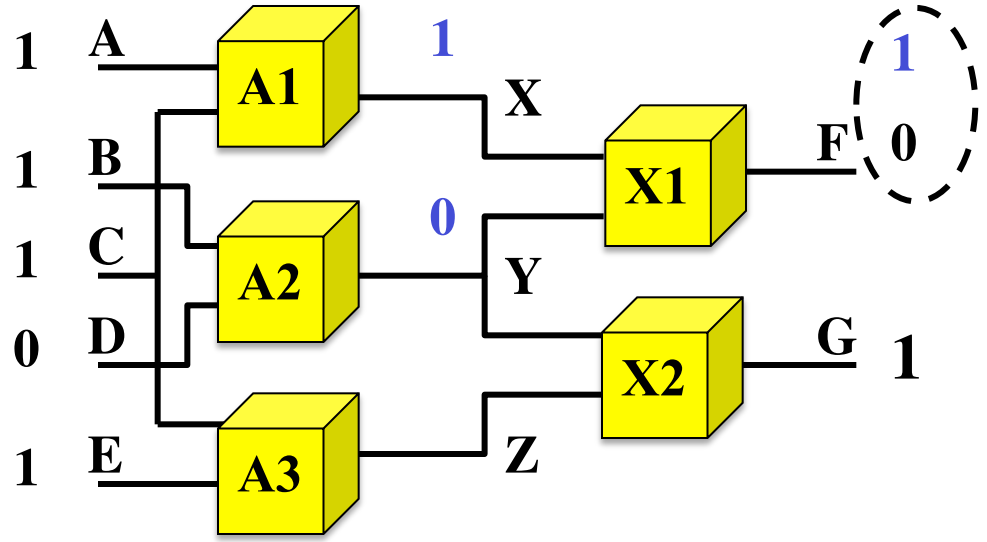
Output: Diagnoses that account for the symptoms.



How Should Diagnoses Account for Novel Symptoms?

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

Suspending Constraints: For novel faults, make no presumption about faulty component behavior.



Symptom

[deKleer & Brown, 83]

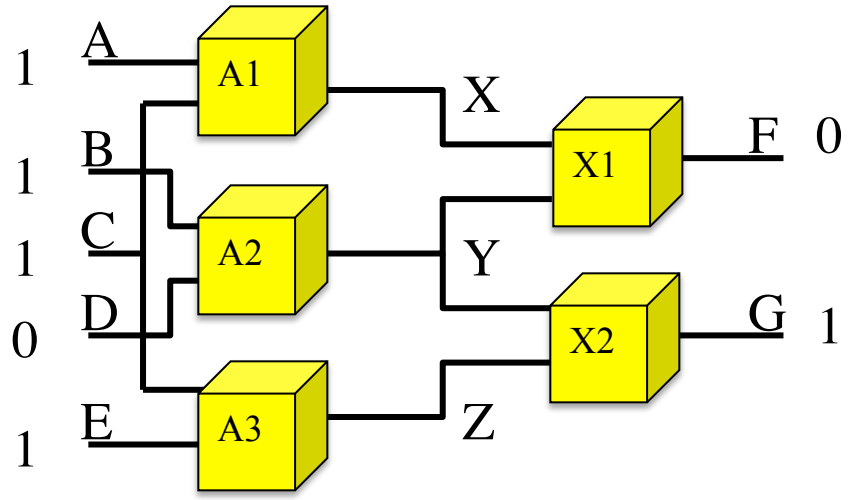
[Davis, 84]

[Geneserth, 84]

Multiple Faults: Identify all Combinations of Consistent "Unknown" Modes

And(i):

- G(i):
  Out(i) = In1(i) AND In2(i)
- U(i): *No Constraint*



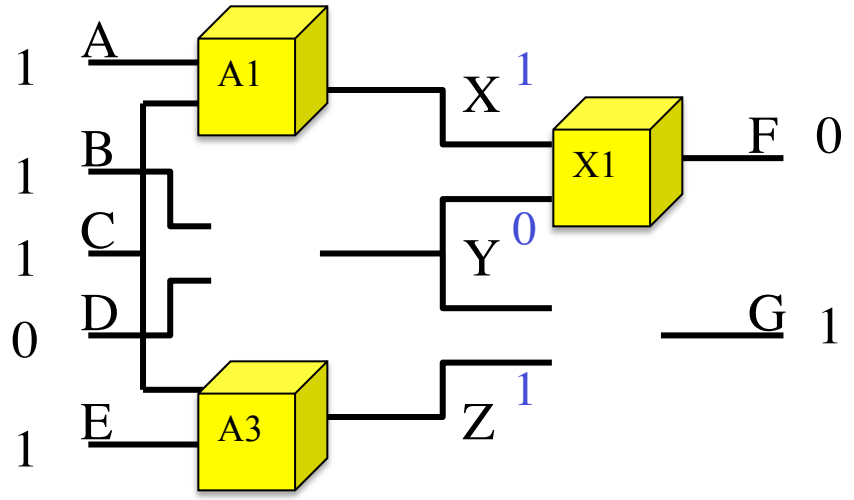
Candidate = {A1=G, A2=G, A3=G, X1=G, X2=G}

- Candidate: Assignment of G or U to each component.

Multiple Faults: Identify all Combinations of Consistent "Unknown" Modes

And(i):

- G(i):
  Out(i) = In1(i) AND In2(i)
- U(i): *No Constraint*

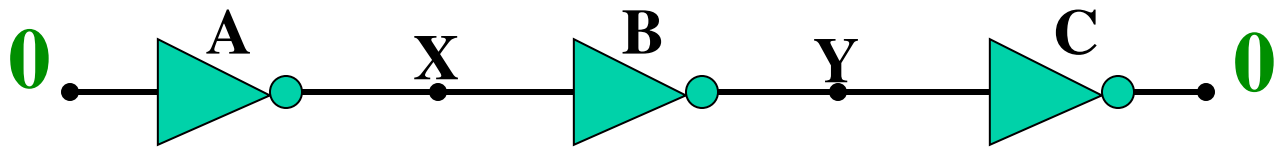


Diagnosis = {A1=G, A2=U, A3=G, X1=G, X2=U}

- Candidate: Assignment of G or U to each component.
- Diagnosis: Candidate consistent with model and observations.

Incorporating (Failure) Modes: Mode Estimation

Sherlock
[de Kleer & Williams, IJCAI 89]



Inverter(i):

- G(i): Out(i) = not(In(i))
- S1(i): Out(i) = 1
- S0(i): Out(i) = 0
- U(i):

- Isolates unknown.
- Explains.

Nominal, Fault and Unknown Modes

Summary: Mode Estimation

Input:

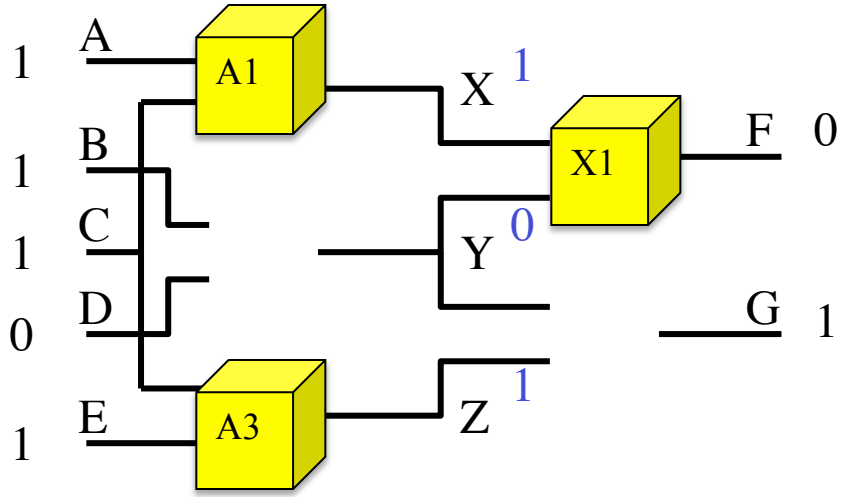
- Mode, State, Observation Variables: X, Y, O⊆ Y
- Obs = assignment to O
- Model: Φ(X,Y) = components + structure

And(i):

G(i):

Out(i) = In1(i) AND In2(i)

U(i): *No Constraint*



• All component behaviors are associated with modes.

• All components have "unknown Mode" U, whose assignment is never mentioned in any constraint.

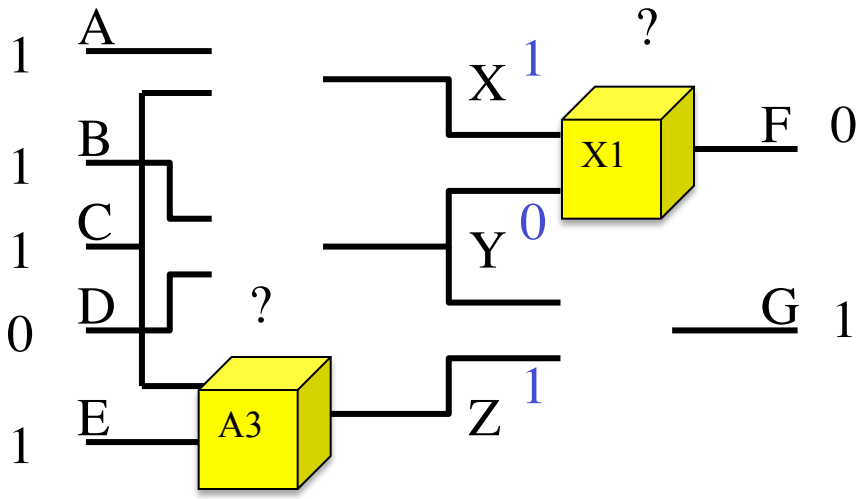
Output: All *mode estimates*:

$$M_{\Phi,obs} \equiv \{X \in D_X \mid \exists Y \in D_X \text{ st Obs} \wedge \Phi(X,Y)\}$$

Compact Encoding: Partial Diagnoses

Partial Diagnosis

{A1=U, A2=U, X2=U}



Partial Diagnosis:

A partial mode assignment M,

that "removes all symptoms."

- All full extensions of M are diagnoses.
  - $\Phi \wedge$ Obs is consistent.
  - M entails $\Phi \wedge$ Obs. *(implicant)*

Extensions (Diagnoses):

{A1=U, A2=U, A3=G, X1=G, X2=U}

{A1=U, A2=U, A3=G, X1=U, X2=U}

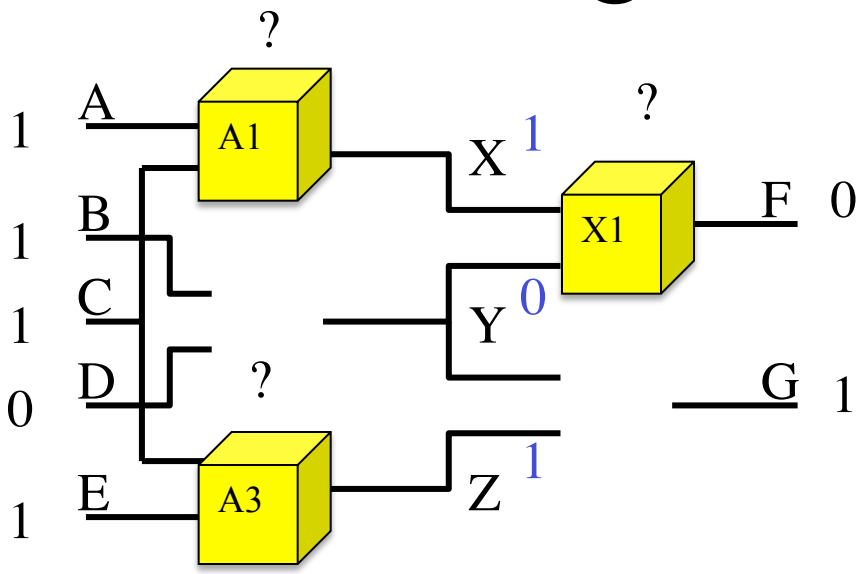
{A1=U, A2=U, A3=U, X1=G, X2=U}

{A1=U, A2=U, A3=U, X1=U, X2=U}

Compact Encoding: Kernel Diagnoses

Kernel Diagnosis

{A2=U, X2=U}



Partial Diagnosis:

A partial mode assignment M, that removes all symptoms.

- M entails $\Phi \wedge$ Obs. *(implicant)*

Kernel Diagnosis:

A partial diagnosis K, no subset of which is a partial diagnosis.

- K is a *prime implicant* of $\Phi \wedge$ Obs.

Robust, Model-based Execution (2): Model-based Programming w Hidden States

- Model-based programming and execution.
- Control sequencing.
- Mode estimation.
  - Estimates and kernels.
  - By divide and conquer (GDE).
  - Likely estimates (Conflict-directed A*).
  - Estimating probabilistic constraint automata
- Mode reconfiguration and reactive planning.

Modes Estimation by Divide and Conquer

Given model Φ and observations Obs,

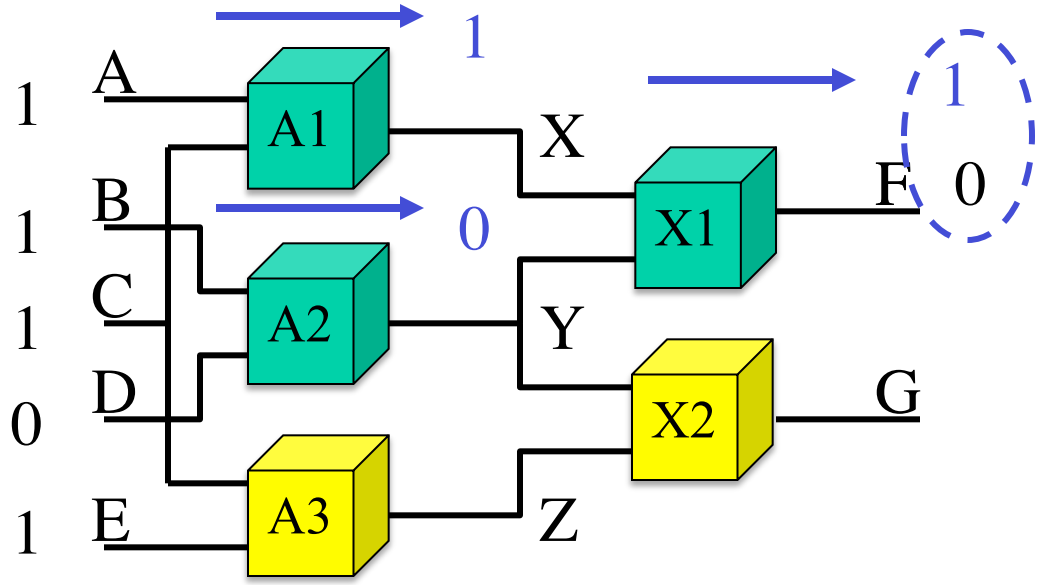
1. Find all symptoms.
2. Diagnose each symptom separately
   (each generates a conflict).
3. Merge diagnoses
   (set covering → kernel diagnoses).

Conflict Recognition

Candidate Generation

General Diagnostic Engine
[de Kleer & Williams, 87]

Conflicts Explain How to Remove Symptoms



Symptom:
F is observed 0, but predicted to be 1 if A1, A2 and X1 are okay.

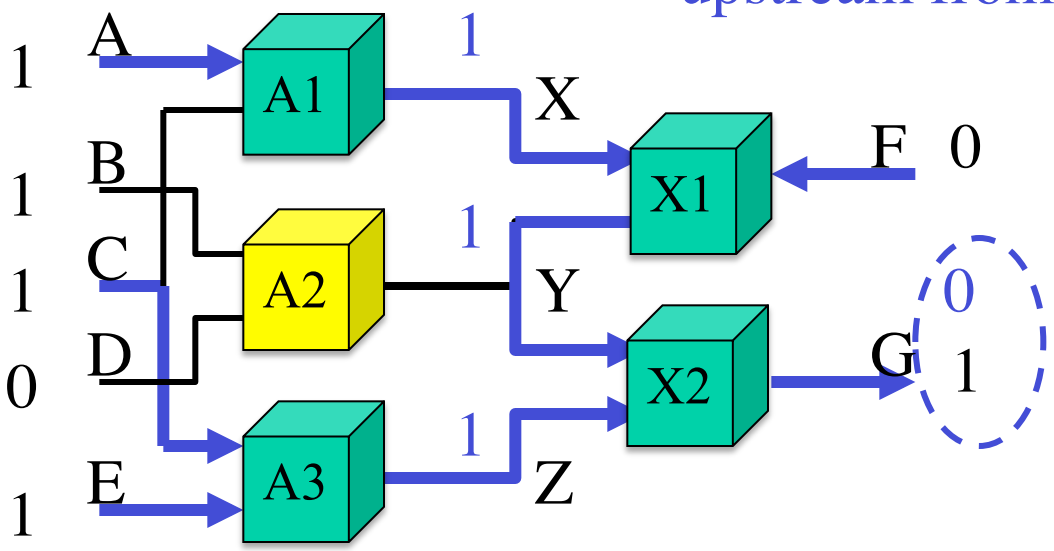
Conflict 1: {A1=G, A2=G, X1=G} is inconsistent.

→ One of A1, A2 or X1 must be broken.

Conflict: An inconsistent partial assignment to mode variables X.

Second Conflict

Conflicting modes aren't always upstream from symptom.

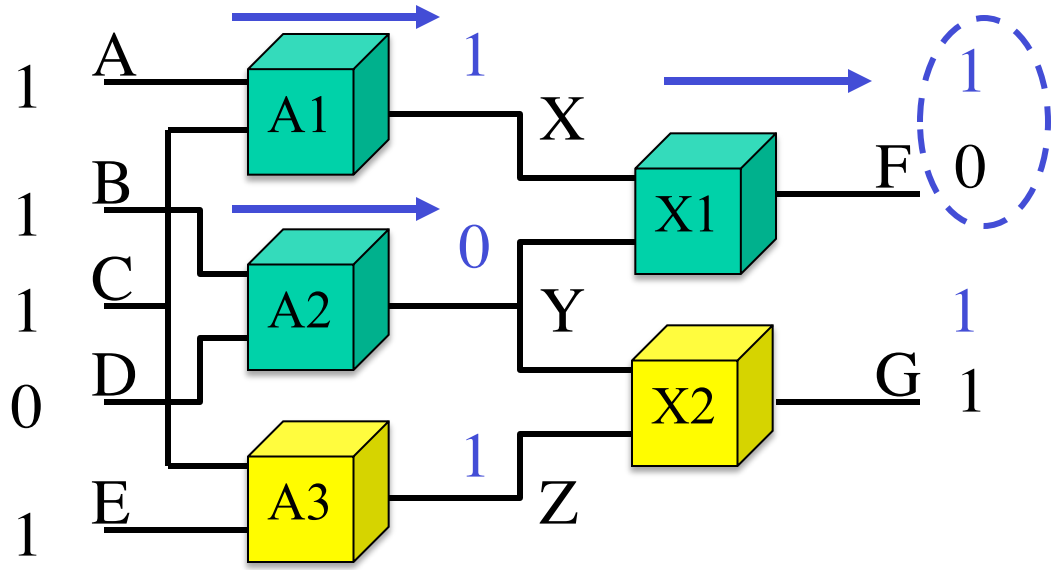


Symptom: G is observed 1, but predicted 0.

Conflict 2: {A1=G, A3=G, X1=G, X2=G} is inconsistent.

→ One of A1, A3, X1 or X2 must be broken.

Summary: Conflicts

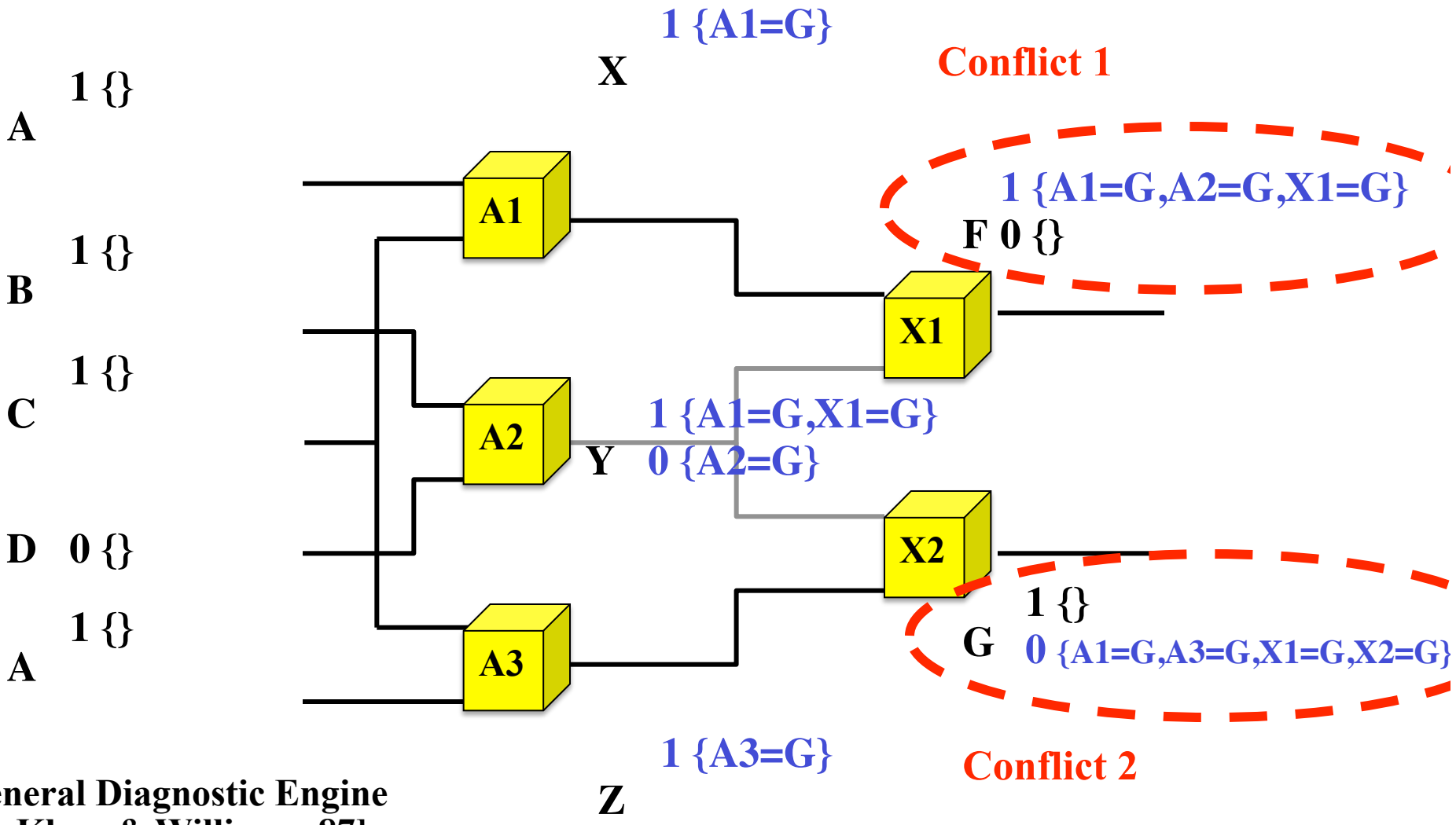


Conflict: A partial mode assignment M that is inconsistent with the model and observations.

Properties:

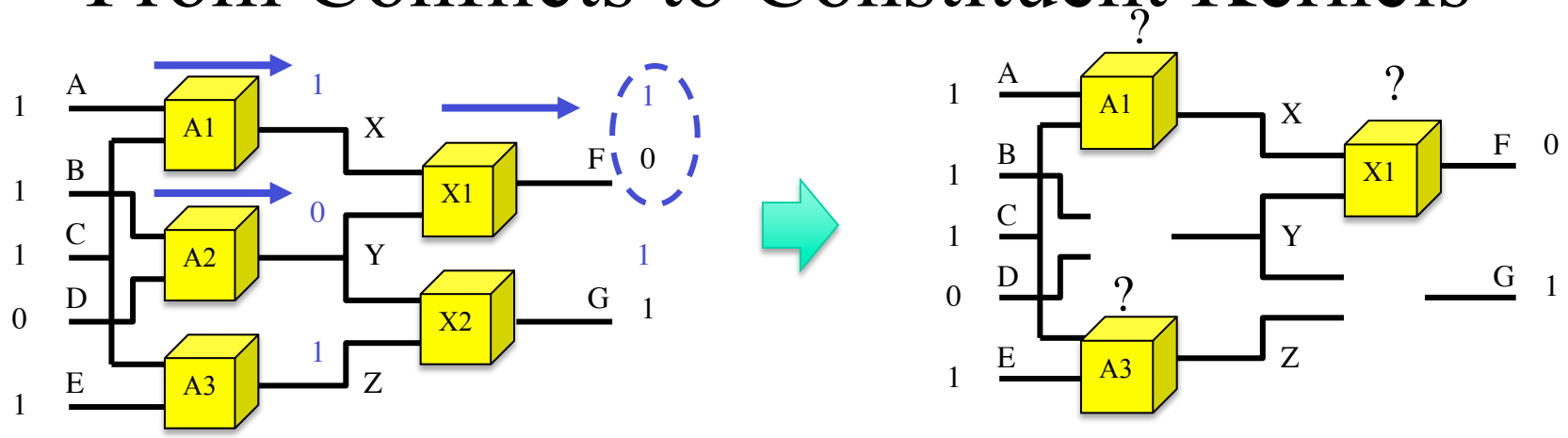
- $\Phi \wedge Obs$ implies $\neg M$
- Every superset of a conflict is a conflict.
- Only need conflicts that are minimal under subset.

Conflict Recognition: Propagating Environments



General Diagnostic Engine
[de Kleer & Williams, 87]

Candidate Generation: From Conflicts to Constituent Kernels



Constituent Kernel: An assignment **a** that "resolves" one conflict C_i.

Conflict: {A1=G, A3=G, X1=G, X2=G}.

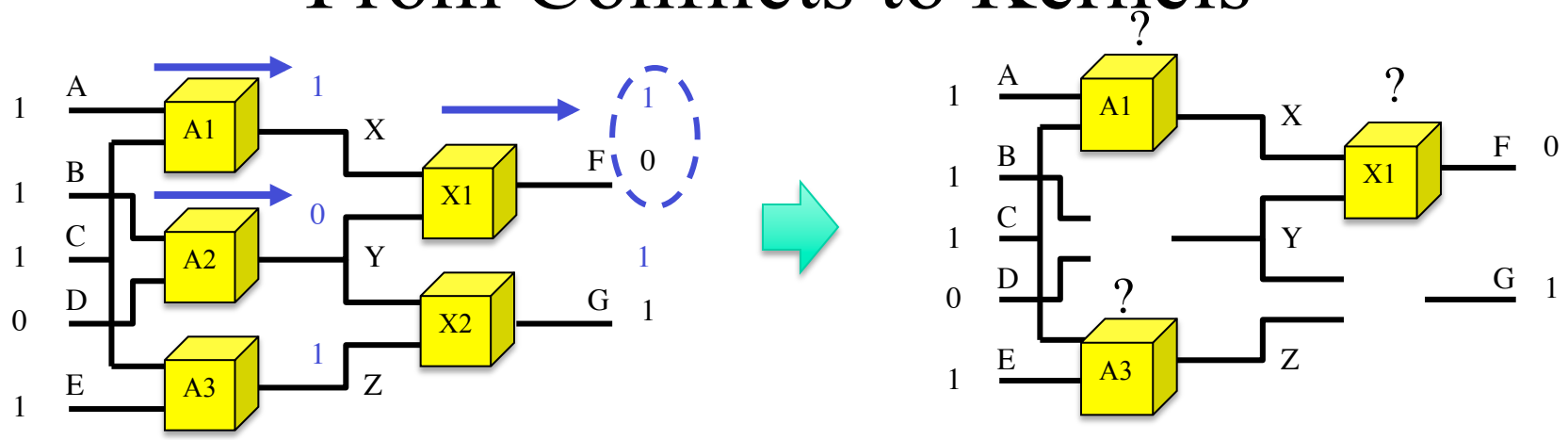
Constituent Kernels: {A1=U, A3=U, X1=U, X2=U}

"resolves" = prevents conflict C_i from being true.

= entails not C_i.

= alternative value of variable mentioned in conflict.

Candidate Generation: From Conflicts to Kernels



Constituent Kernel: An assignment **a** that "resolves" one conflict C_i.

Kernel: Minimal set of assignments **A** that "resolves" all conflicts C.

⇨Pick constituent from each conflict using minimal set covering.

{A2=U, X2=U} resolves {A1=G, A3=G, X1=G, X2=G}, and
{A2=U, X2=U} resolves {A1=G, A2=G, X1=G}.

Candidate Generation: Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses =

"Smallest" sets of modes that remove all conflicts.

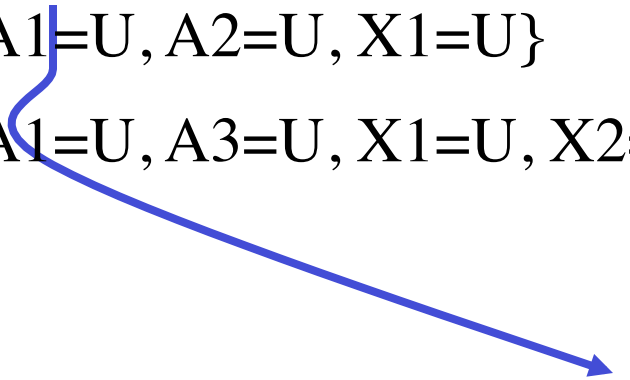
Candidate Generation: Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.



Kernel Diagnoses = {A1=U}

1. Compute cross product.
2. Remove supersets.
   - Old subset New.
   - New subset Old.

"Smallest" sets of modes that remove all conflicts.

Candidate Generation: Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses = ~~{A1=U, A3=U}~~
{A1=U}

1. Compute cross product.
2. Remove supersets.
   - Old subset New.
   - New subset Old.

"Smallest" sets of modes that remove all conflicts.

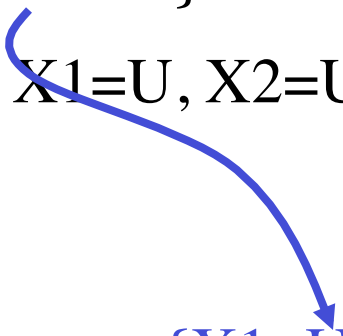
Candidate Generation: Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.



Kernel Diagnoses = {X1=U}
{A2=U, X2=U}
{A2=U, A3=U}
{A1=U}

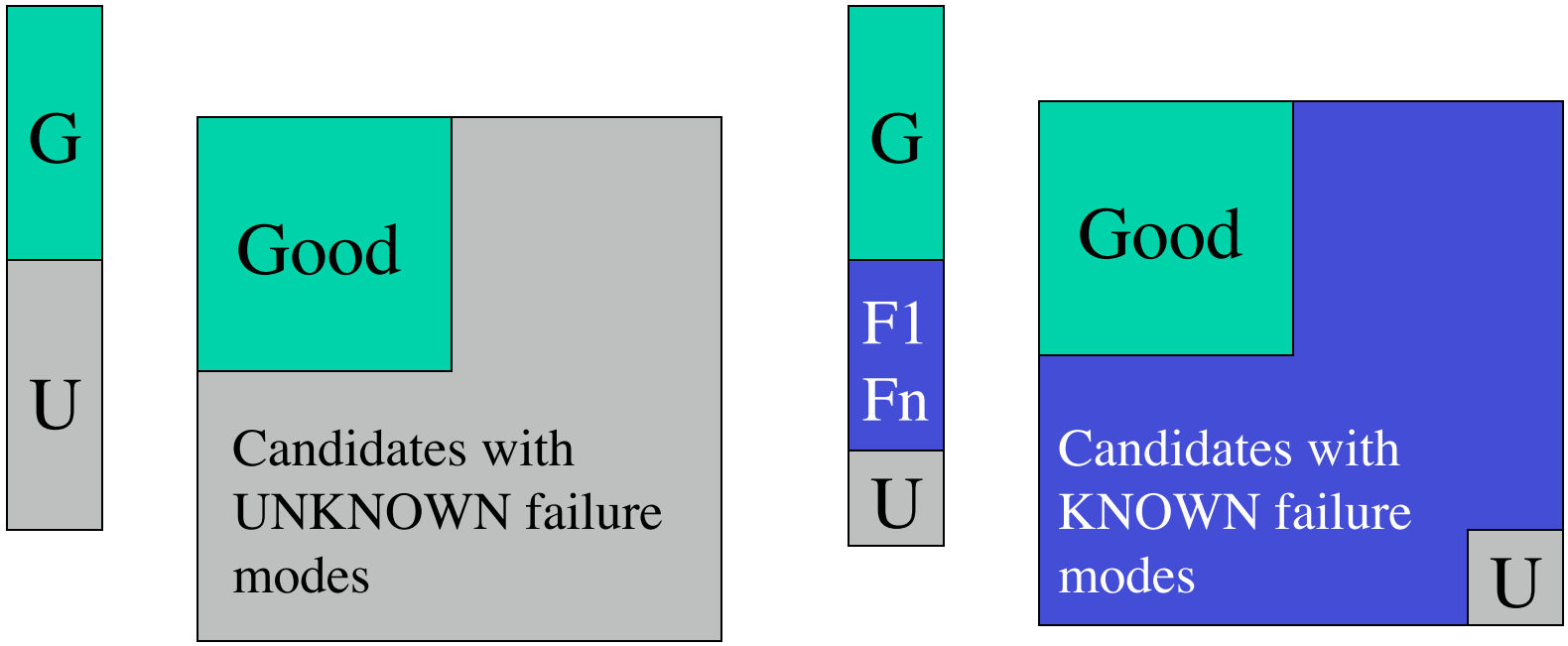
1. Compute cross product.
2. Remove supersets.
   - Old subset New.
   - New subset Old.

"Smallest" sets of modes that remove all conflicts.

Robust, Model-based Execution (2): Model-based Programming w Hidden States

- Model-based programming and execution.
- Control sequencing.
- Mode estimation.
  - Estimates and kernels.
  - By divide and conquer (GDE).
  - Likely estimates (Conflict-directed A*).
  - Estimating probabilistic constraint automata
- Mode reconfiguration and reactive planning.

Due to the unknown mode, there tends to be an exponential number of mode estimates.



Fault models alone don't help.

But most mode estimates represent a small fraction of the probability density space.

⇨ Most of the density space may be represented by enumerating the few most likely modes P(X | Obs.)

Simple Probabilistic Mode Estimation

Input:

- Mode X, State Y and Observation O variables with finite domains.
- Model $\Phi(X;Y)$.
- Observations obs.
- Prior distribution $P(X_i)$ for each component i.

Output:

- $P(X \mid obs)$ Posterior, given observations.

$$P(X \mid obs) = \alpha P(obs \mid X) P(X)$$

- Assume modes are *a priori* independent:

$$P(X) = \prod_{X_i \in X} P(X_i)$$

- Assume consistent observations* are equally likely for a given mode assignment:

* or consistent models

$$P(obs \mid X) = \begin{cases} 0 & \text{if } \Phi \wedge obs \wedge X \text{ is inconsistent} \\ 1/n & \text{else } n = \left| \{ obs_i \mid \Phi \wedge obs_i \wedge X \text{ is consistent} \} \right| \end{cases}$$

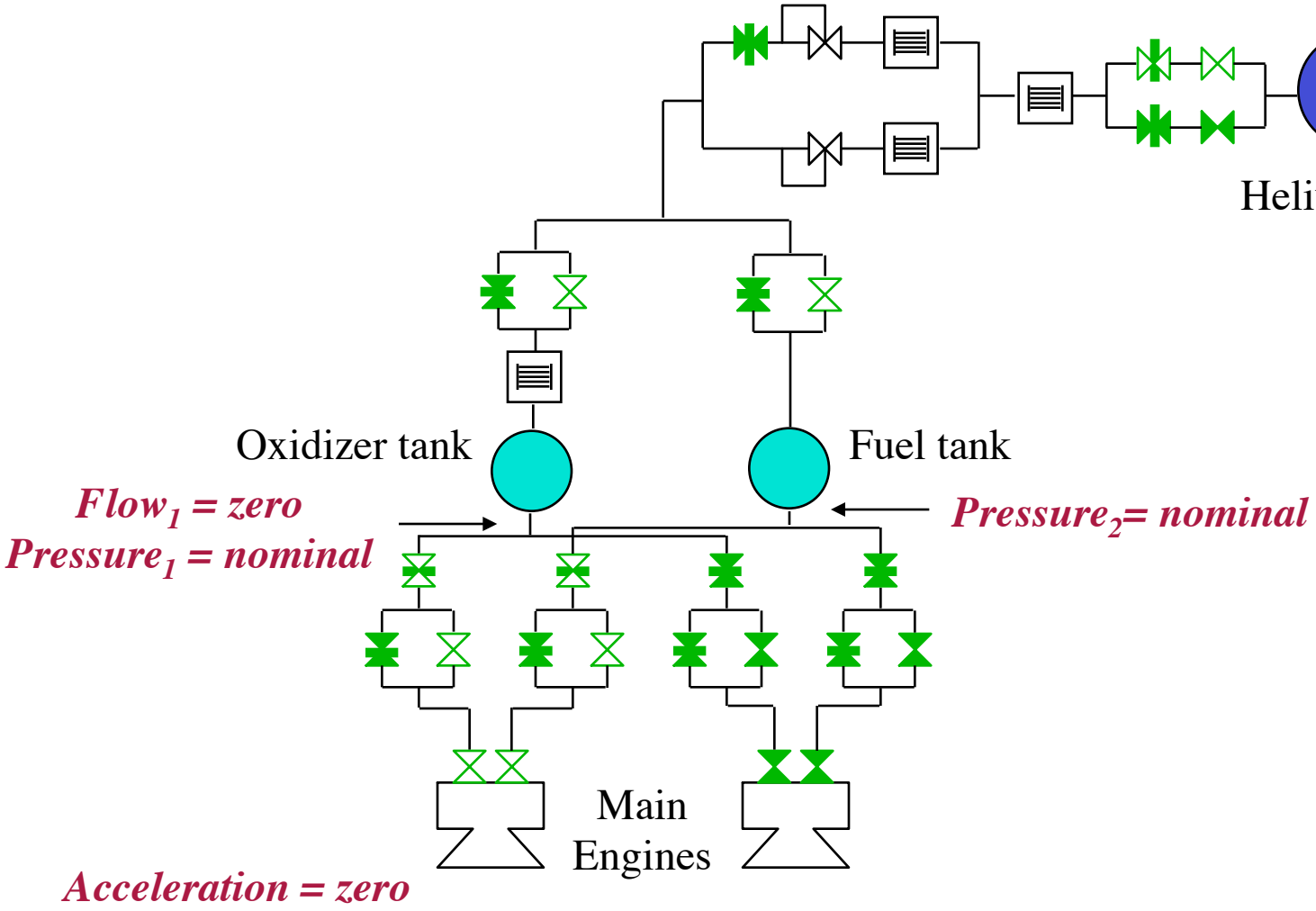
Mode Estimation as Conflict-directed Best First Search

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

- Sherlock Holmes. The Sign of the Four.

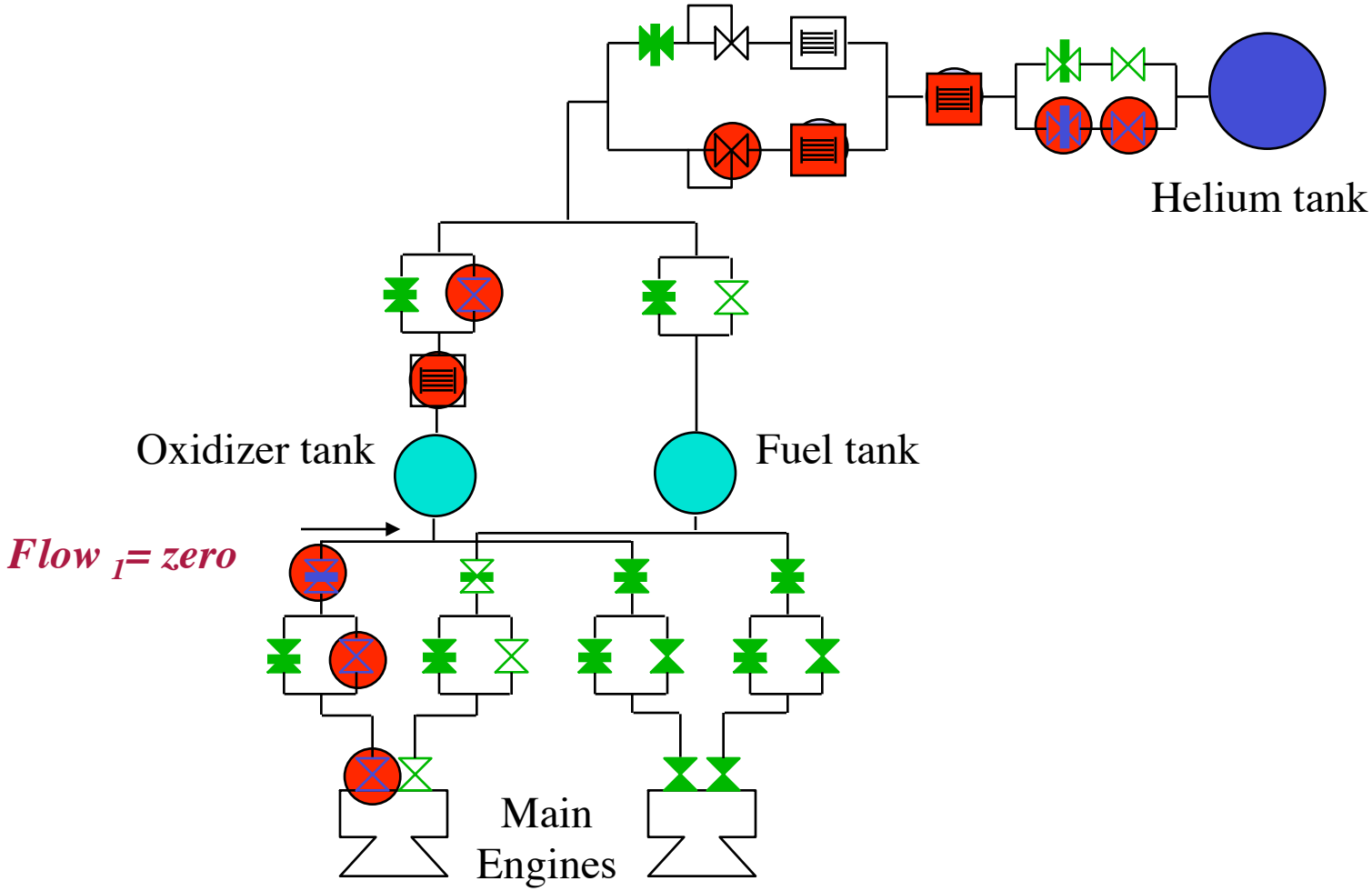
1. Generate most likely hypothesis.
2. Test hypothesis.
3. If inconsistent, learn reason for inconsistency (a conflict).
4. Use conflicts to leap over similarly infeasible options to next best hypothesis.

Compare Most Likely Hypothesis to Observations



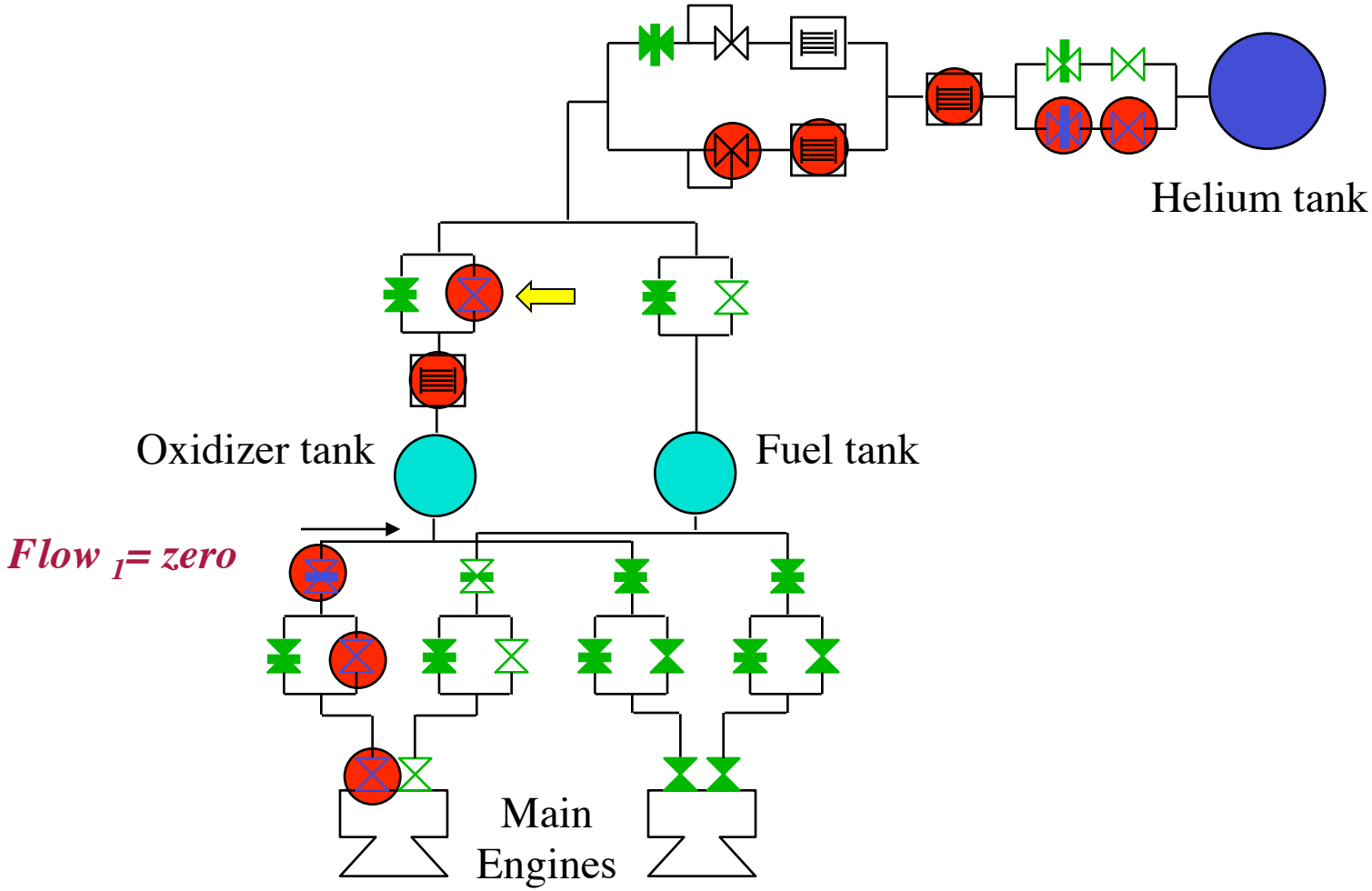
It is most likely that all components are okay.

Isolate Conflicting Information



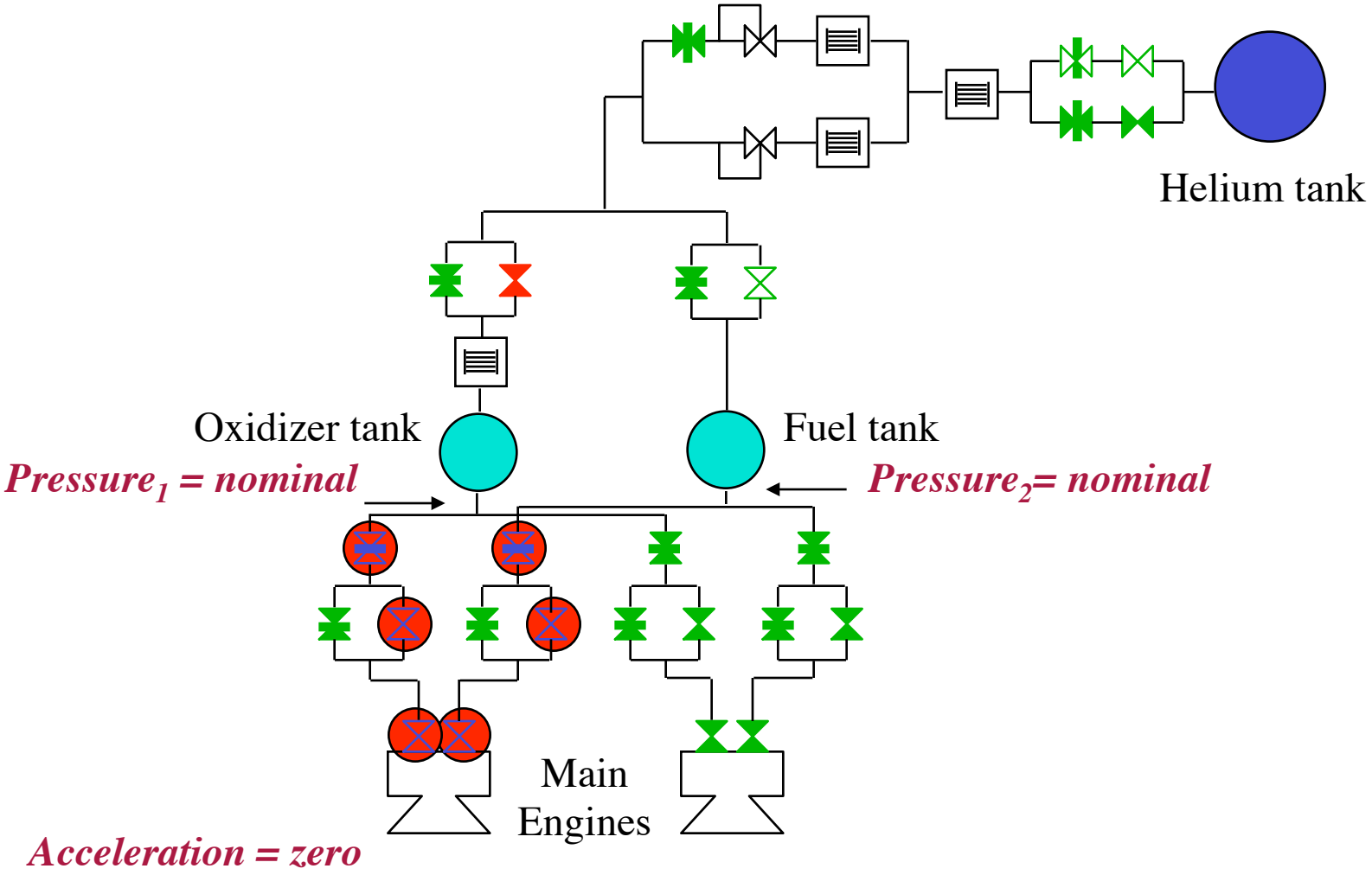
The red component modes *conflict* with the model and observations.

Leap to the Next Most Likely Hypothesis that Resolves the Conflict



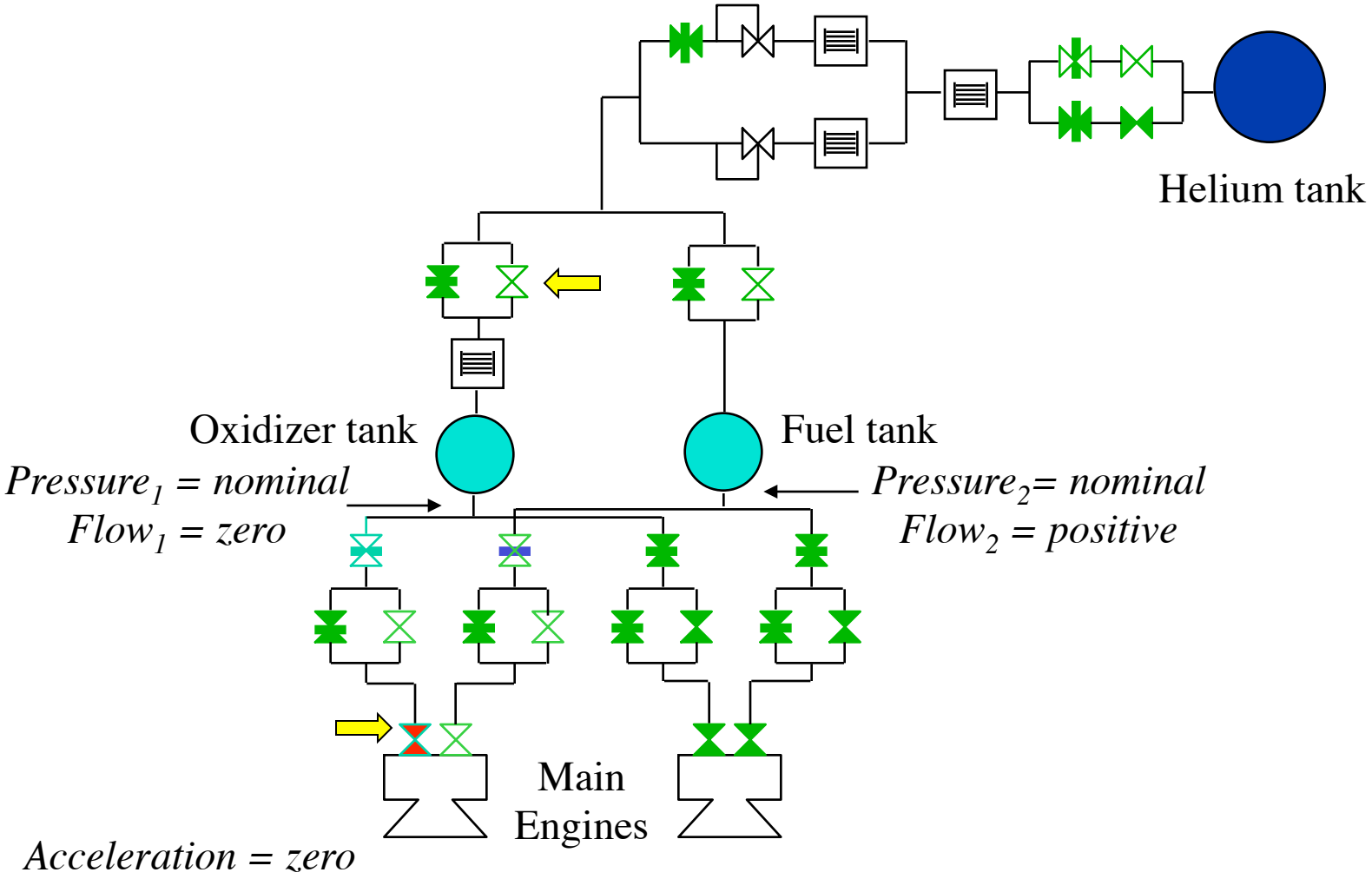
The next hypothesis must remove the conflict.

New Hypothesis Exposes Additional Conflicts



Another conflict, try removing both.

Final Hypothesis Resolves all Conflicts



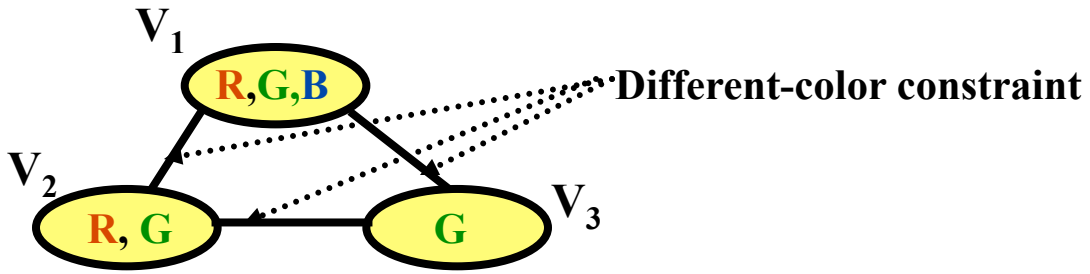
Implementation: Optimal CSPs and Conflict-directed A*.

Constraint Satisfaction Problem

CSP = <Y, D_Y,C>

- variables Y, with domain D_Y.
- Constraints C: $D_Y \rightarrow$ {True, False}.

Problem: Find Y in D_Y s.t. C(Y) .



Optimal CSP

Input: <X, *g*, CSP>

- X are decision variables with domain D_X.
- g: $D_X \rightarrow \Re$ is a utility function.
- CSP over variables <X;Y>.

Output: Find leading arg max *g*(X)

$$X \in D_X$$

$$\text{s.t. } \exists\, Y \in D_Y \,.\, C(X;Y).$$

➔ g() is a multi-attribute utility function that is *preferentially independent*.

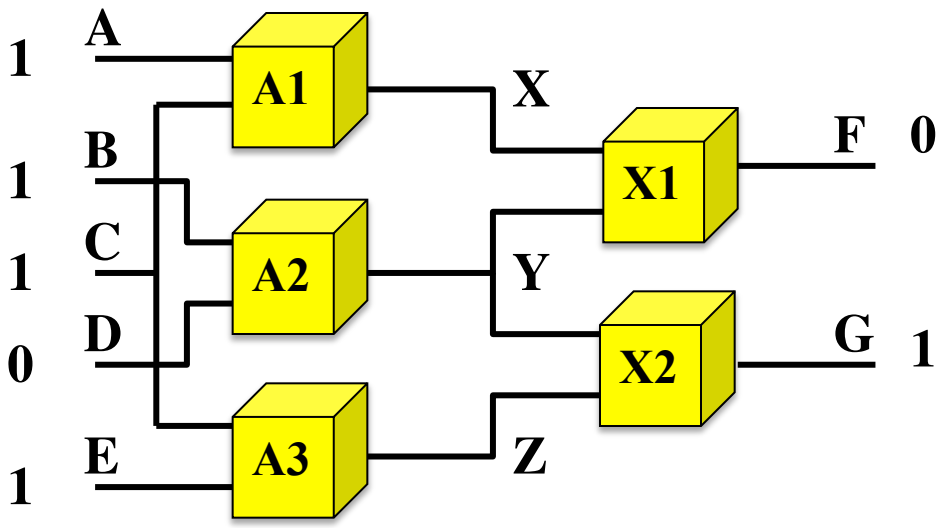
➔ Encode C in propositional state logic.

Mode Estimation

$$\text{Find leading arg max}_{X \in D_X} \; \alpha P(obs \mid X) \prod_{X_i \in X} P(X_i)$$

$$\text{s.t. } \exists\, Y \in D_Y \,.\, X \wedge \Phi(X,Y) \wedge \text{obs.}$$

Probabilities for Boolean Polycell



- Assume independent failures.
- Assign P such that:
  - $P_{Xi=G} >> P_{Xi=U}$
  - $P_{single} >> P_{double}$
  - $P_{A2=U} > P_{A1=U} > P_{A3=U} > P_{X1=U} > P_{X2=U}$

Mutual Preferential Independence (MPI) Example: Mode Estimation

Our preference for the assignment of one variable is independent of the assignments to the other variables.

If A1 = G is more likely than A1 = U,

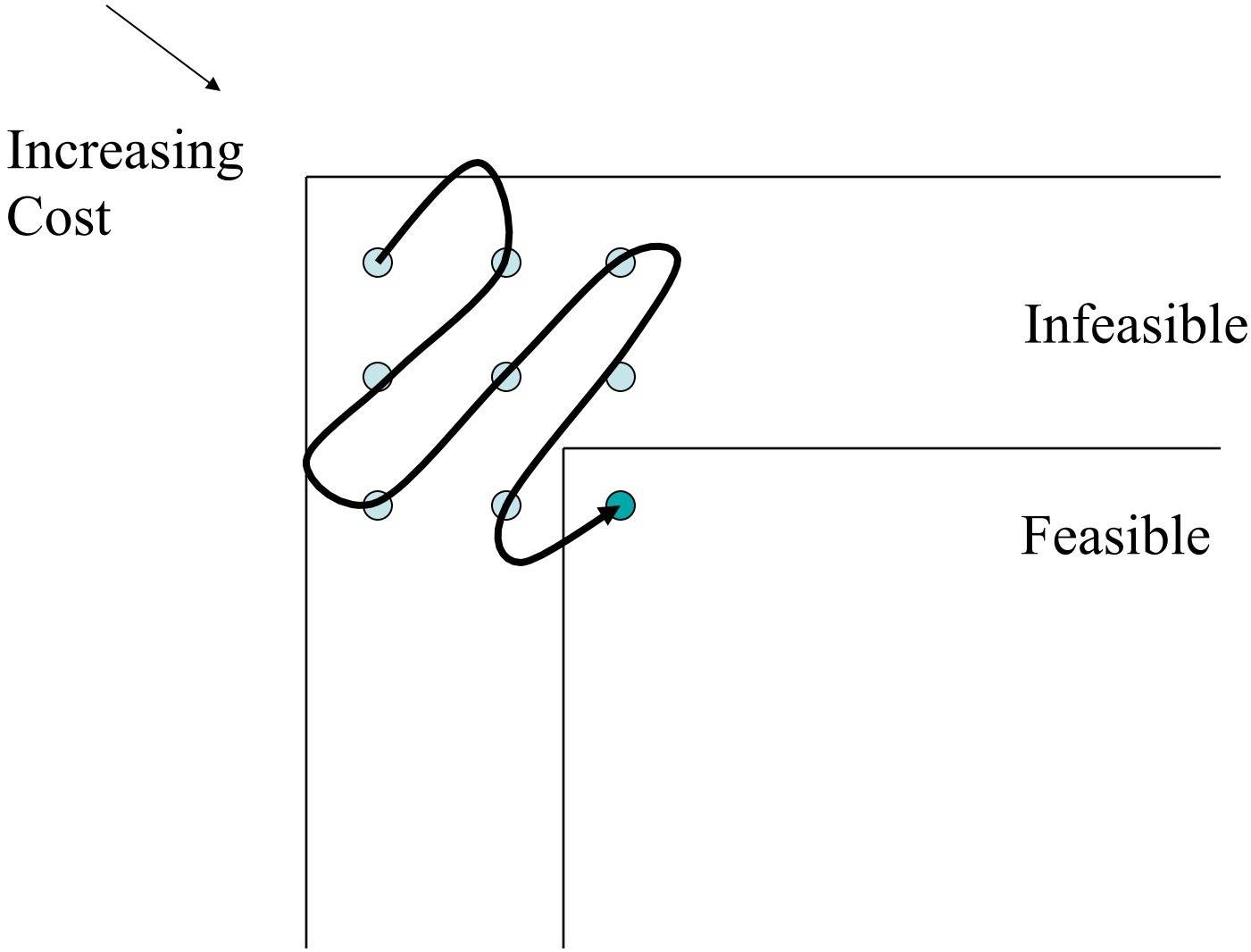
Then

{A1 = G, A2 = G, A3 = U, X1 = G, X2 = G}

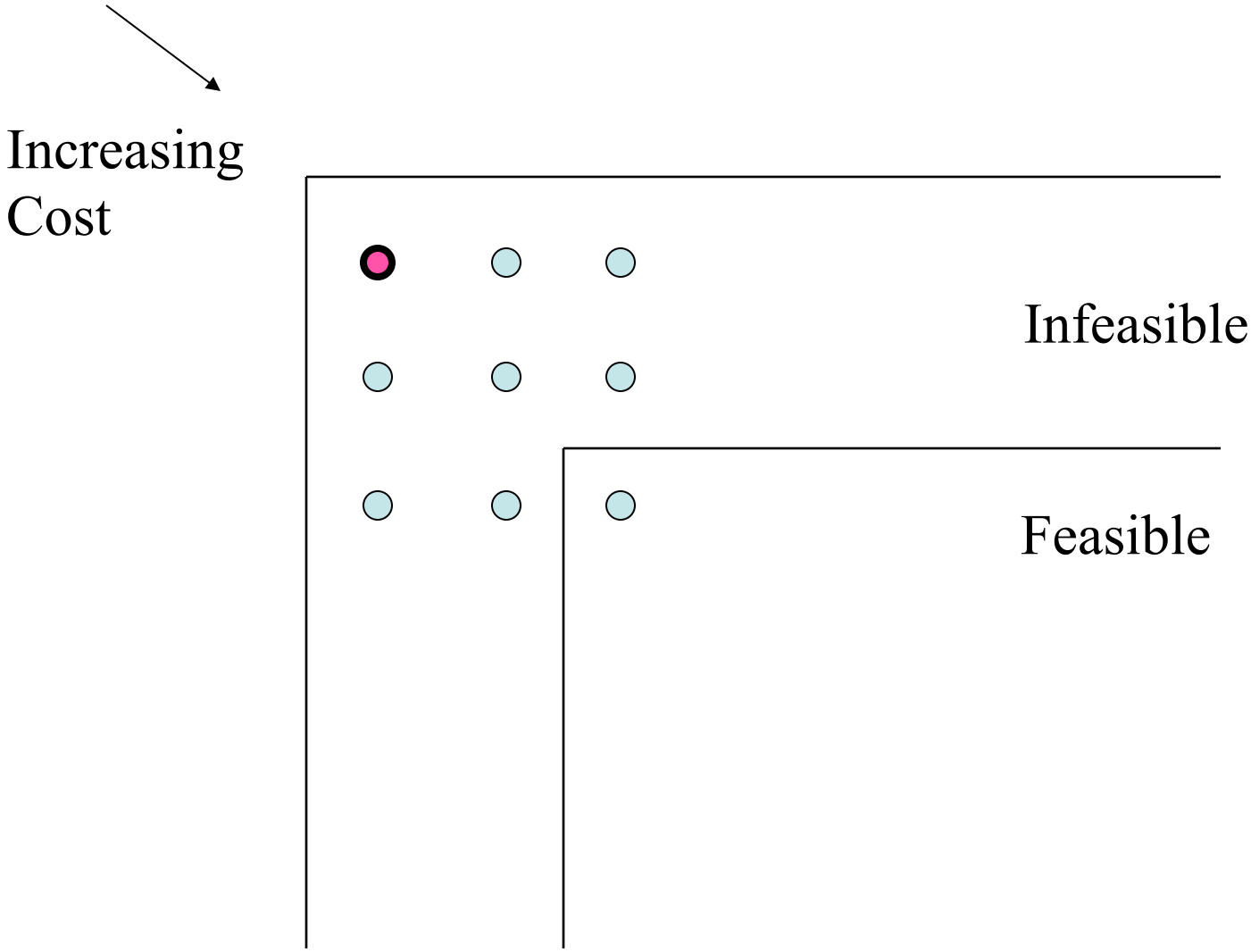
is preferred to

{A1 = U, A2 = G, A3 = U, X1 = G, X2 = G}.

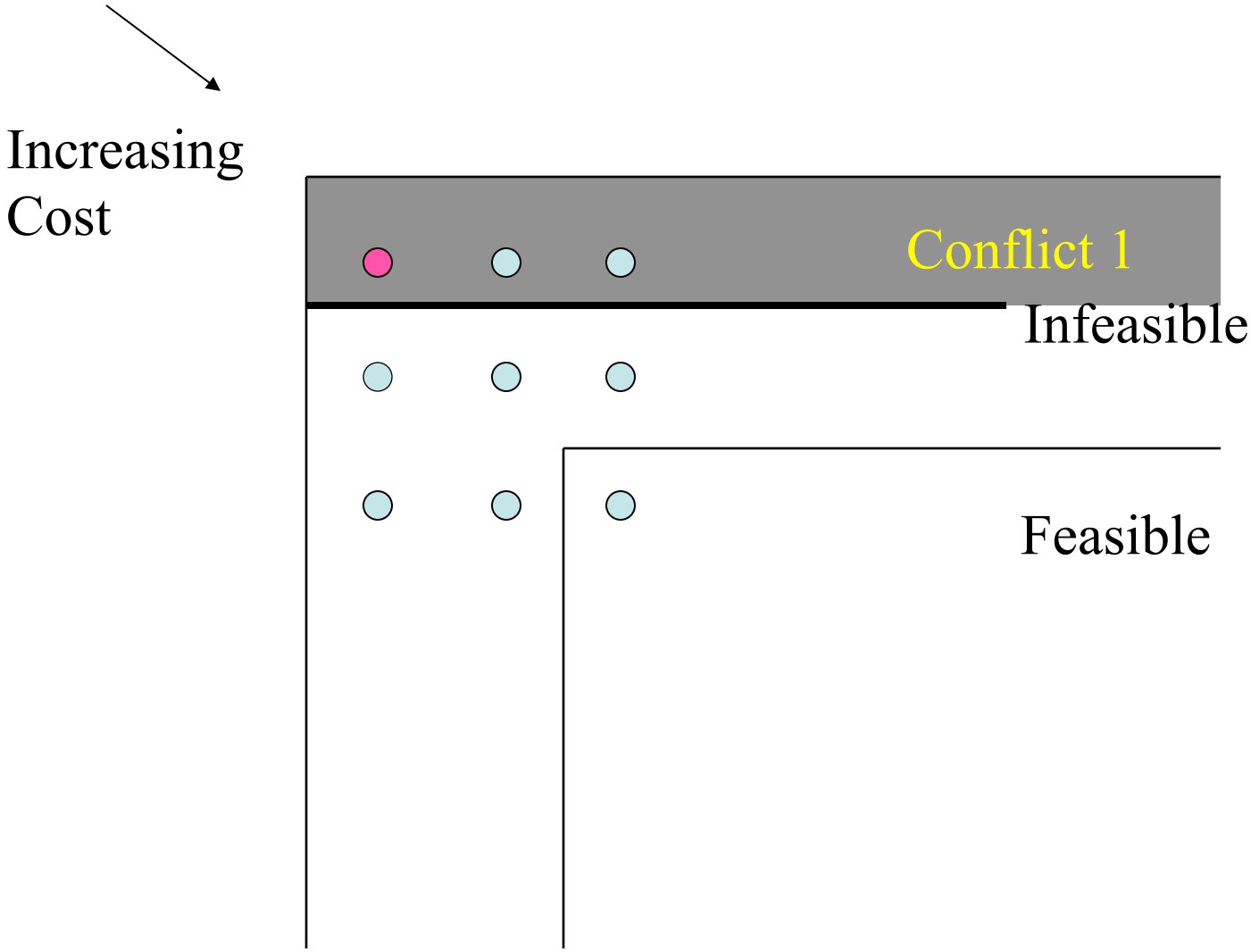
A*



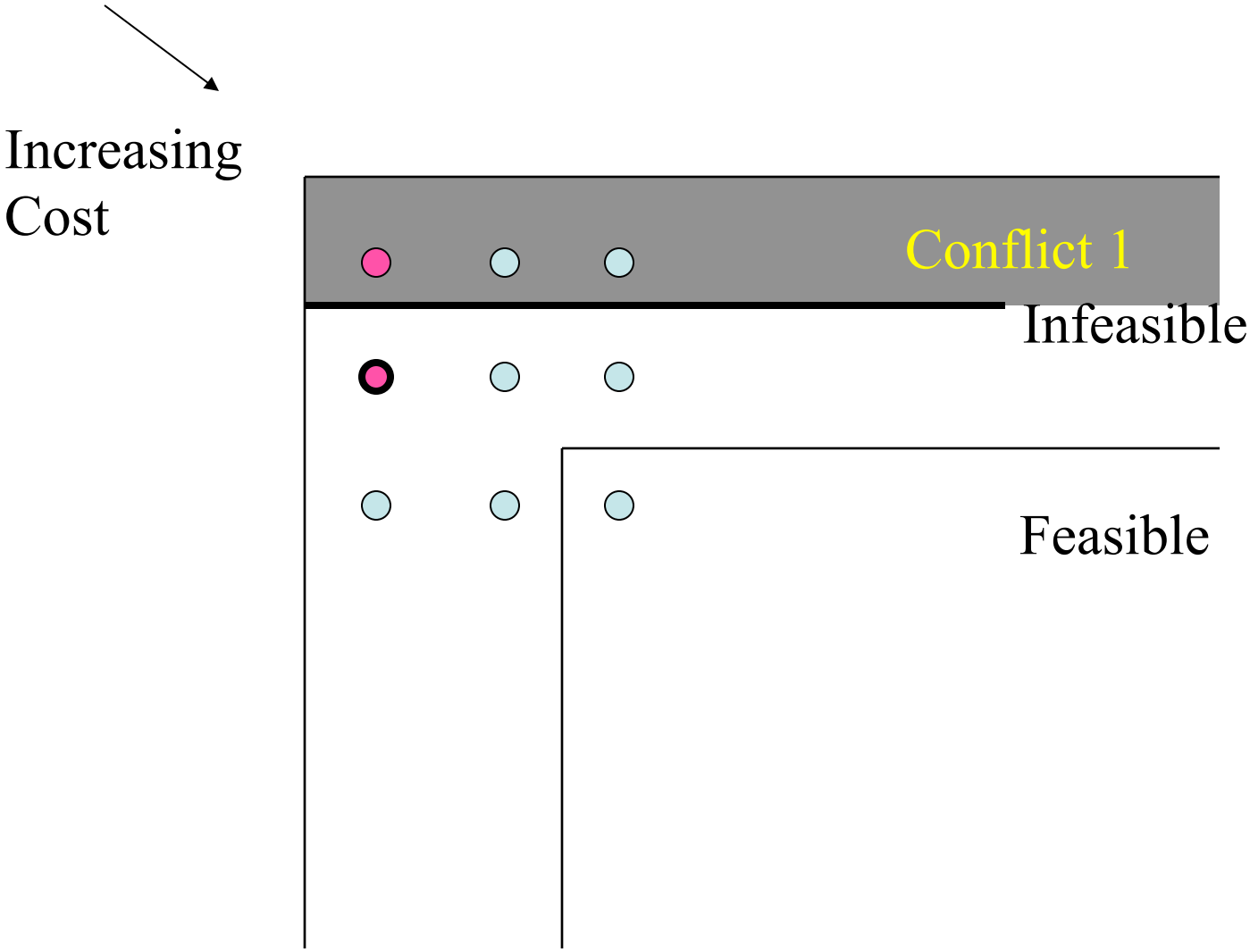
Conflict-directed A*



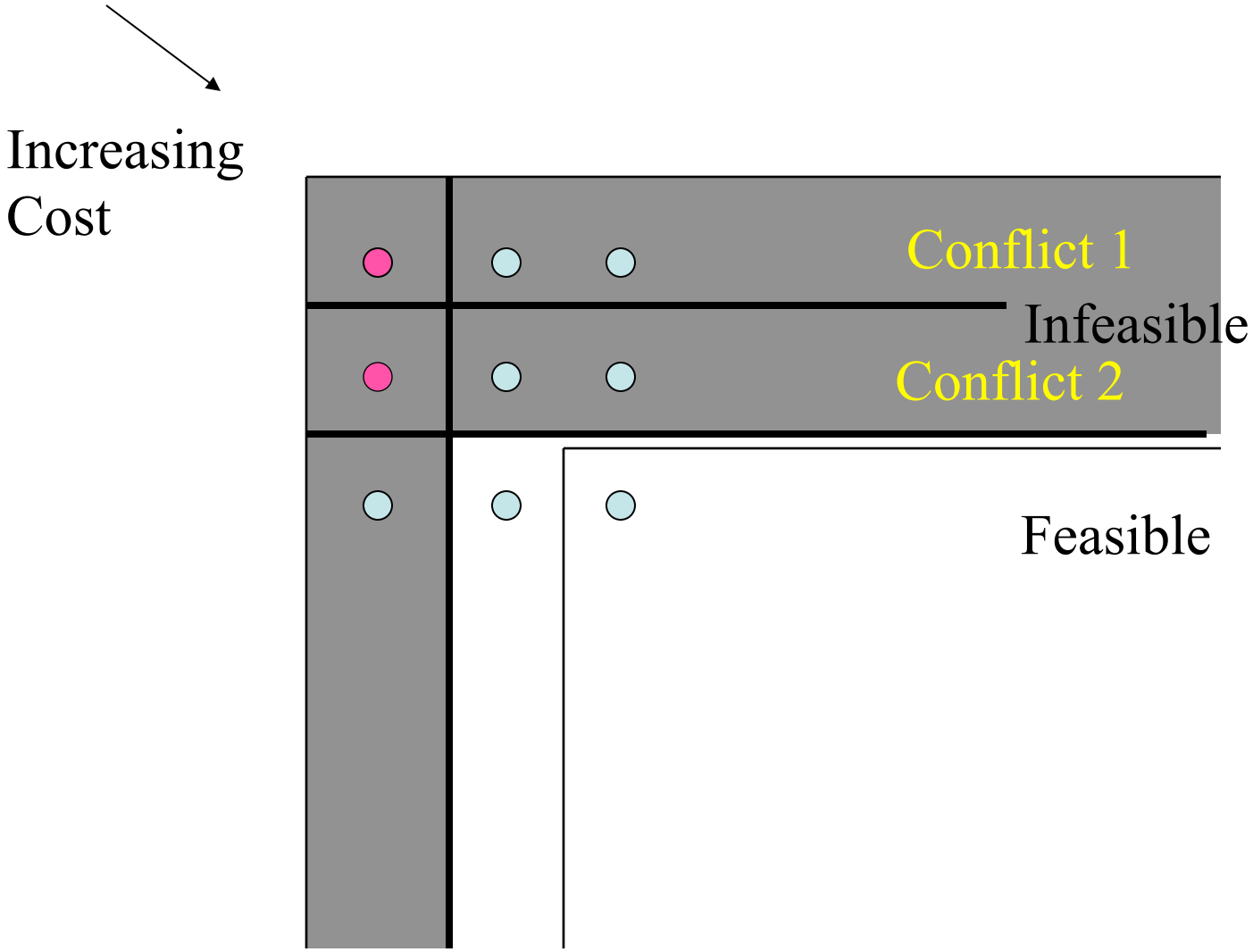
Conflict-directed A*



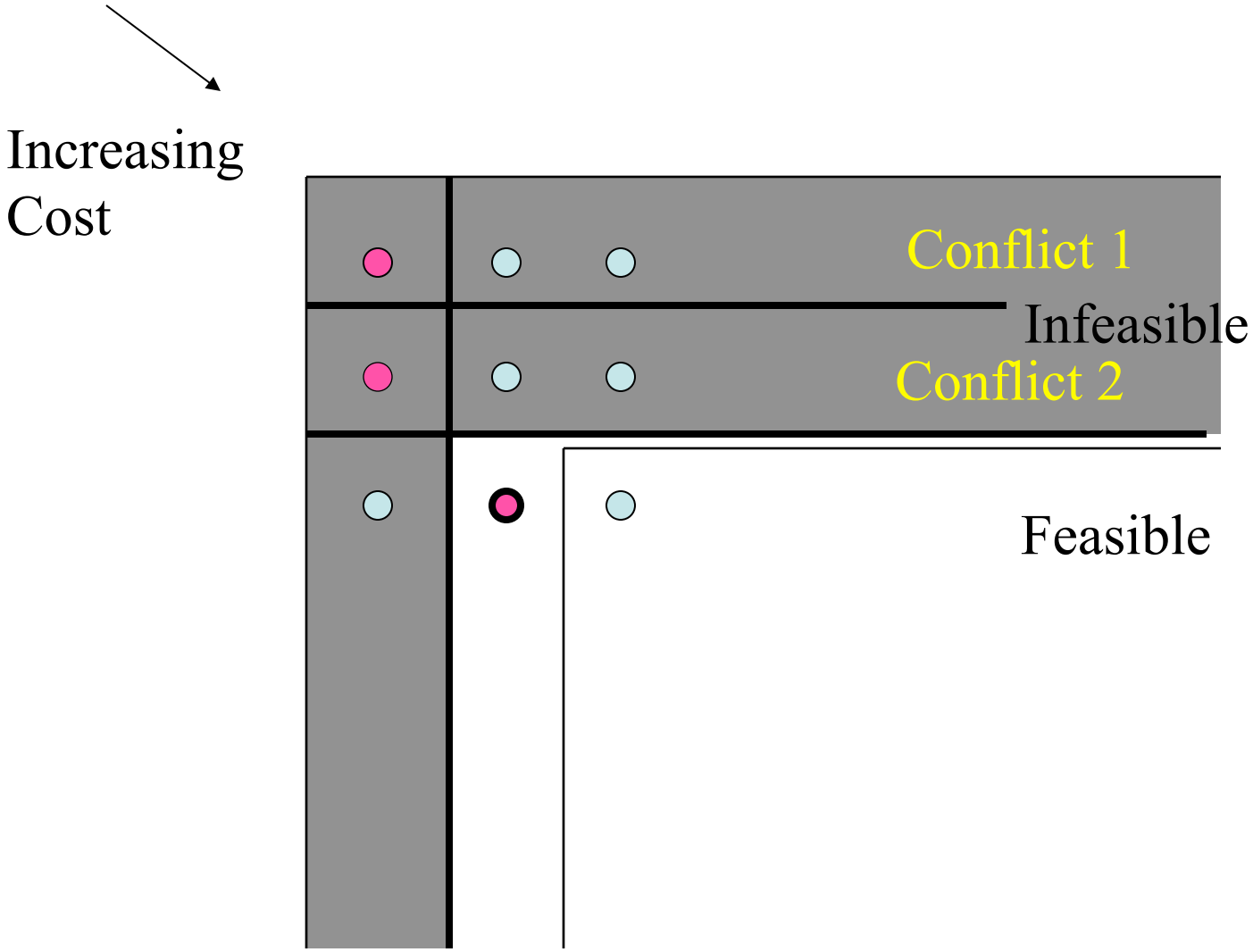
Conflict-directed A*



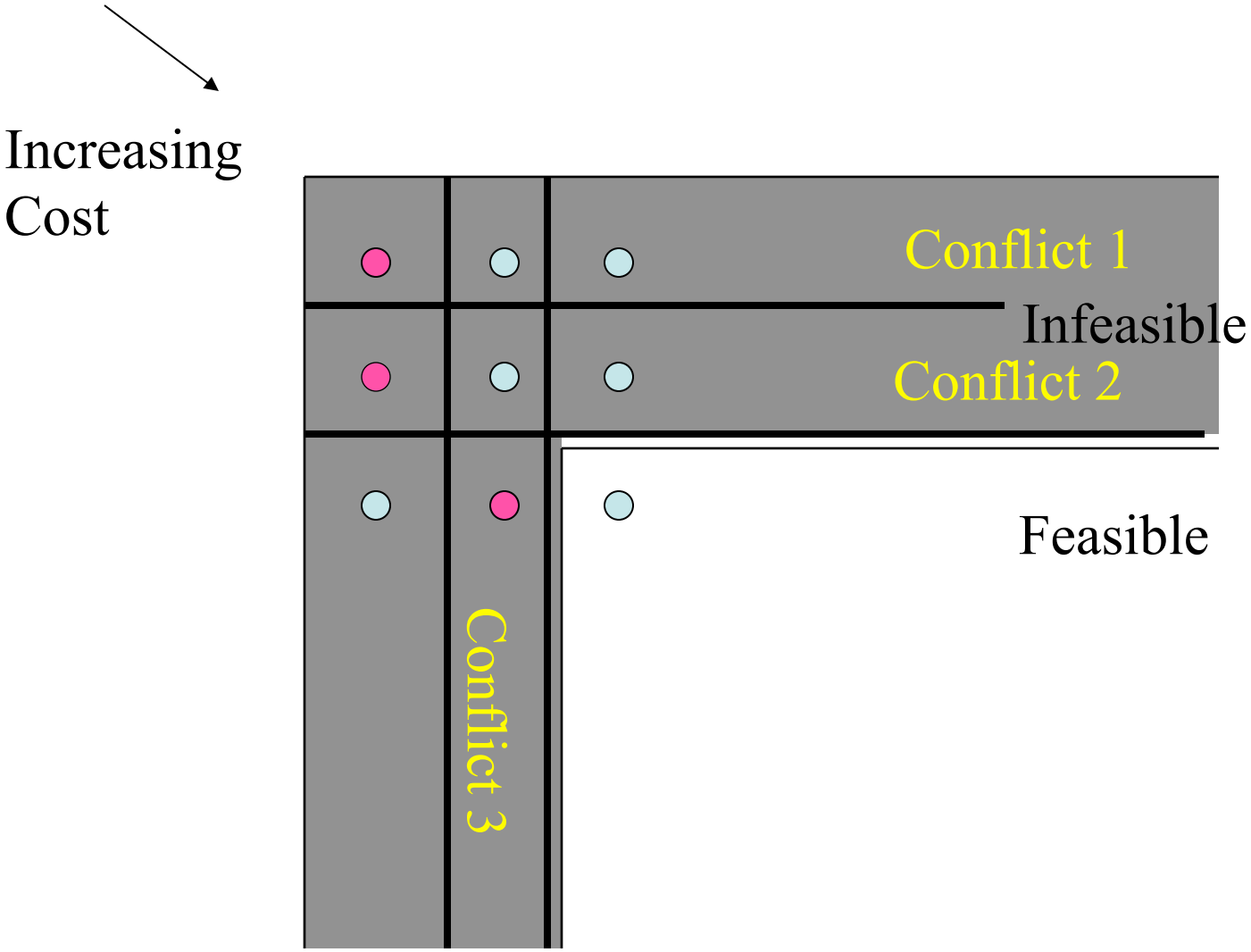
Conflict-directed A*



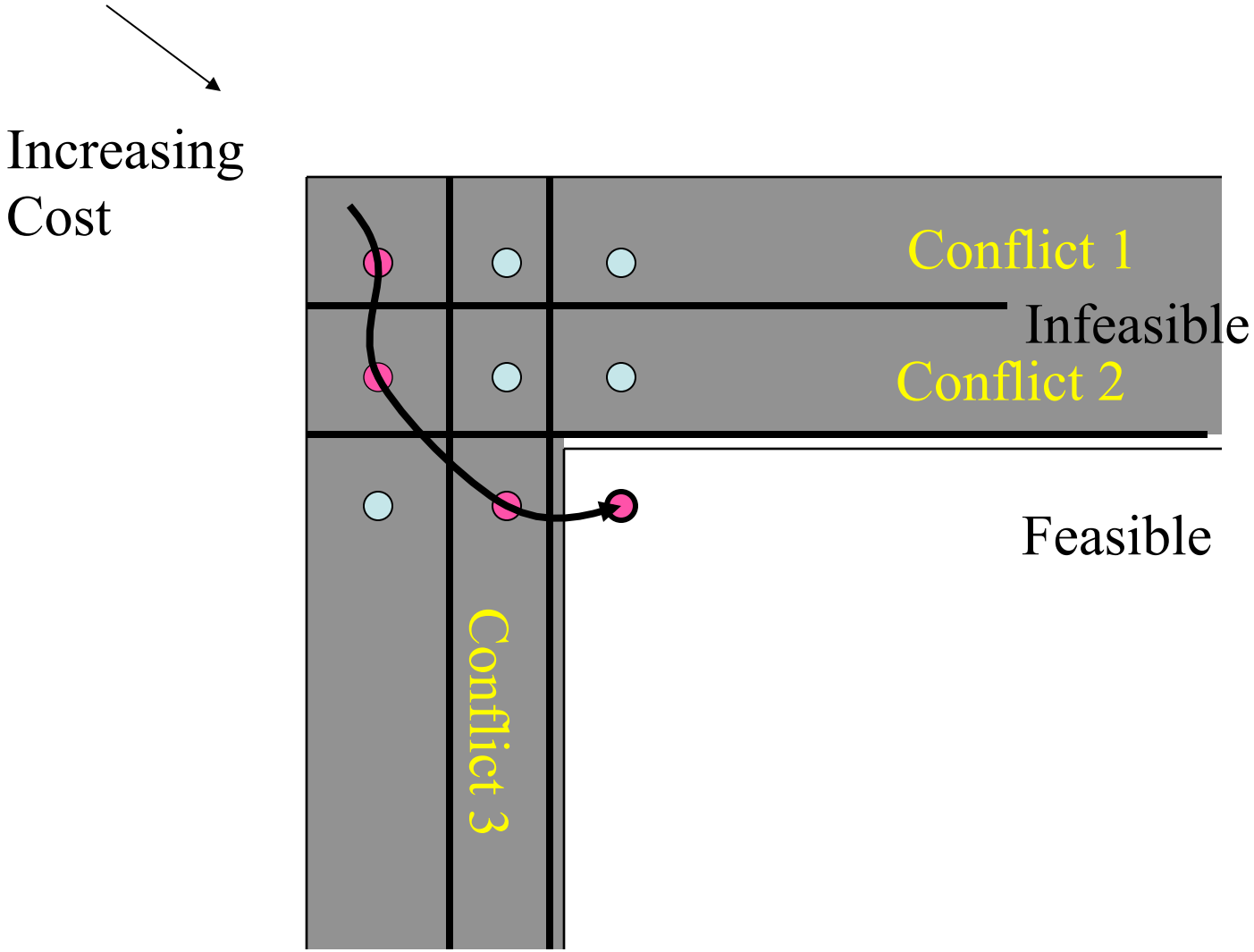
Conflict-directed A*



Conflict-directed A*



Conflict-directed A*

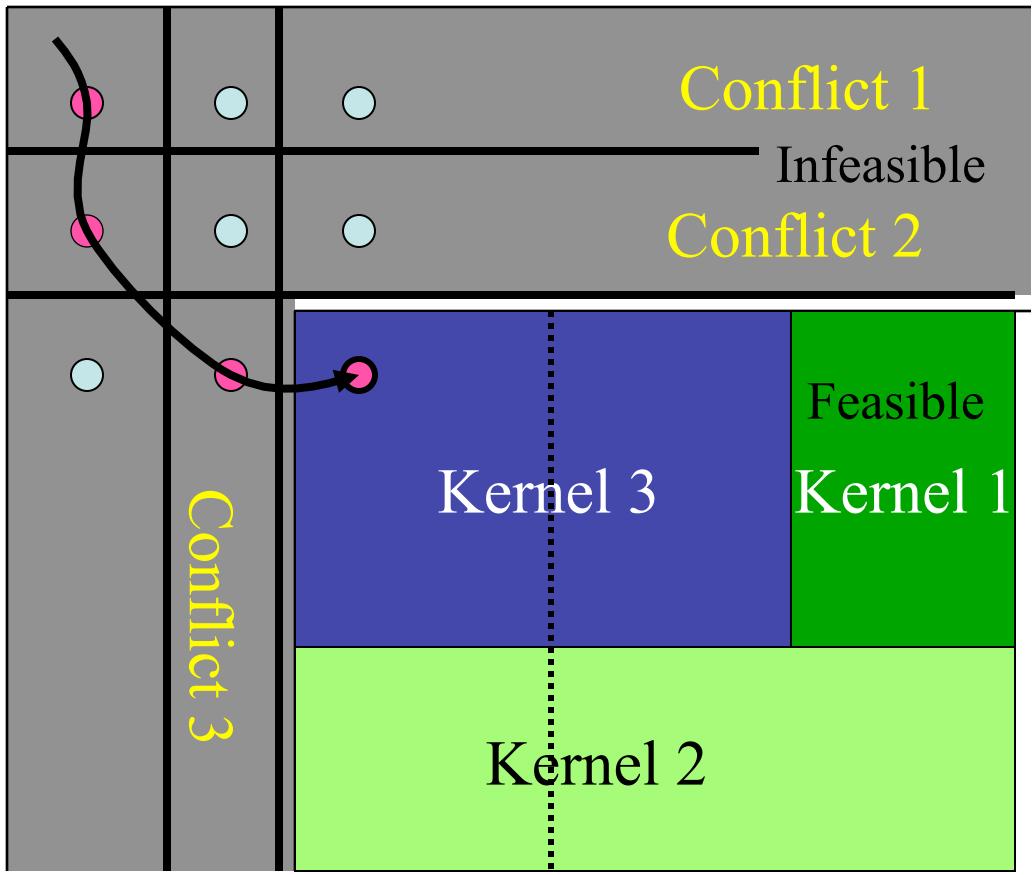


Conflict-directed A*

- Each feasible subregion described by a kernel assignment.

⇨ Approach: Use conflicts to search for kernel assignment containing the best cost candidate.

Increasing Cost



Extracting a Kernel's Best State

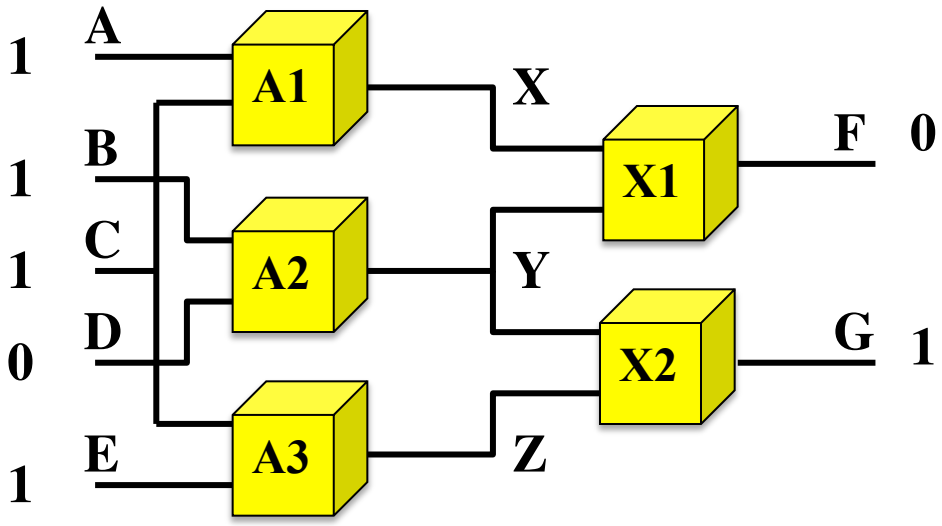
{A2=U}

A1=? ∧ A2=U ∧ A3=? ∧ X1=? ∧ X2=?

A1=G ∧ A2=U ∧ A3=G ∧ X1=G ∧ X2=G

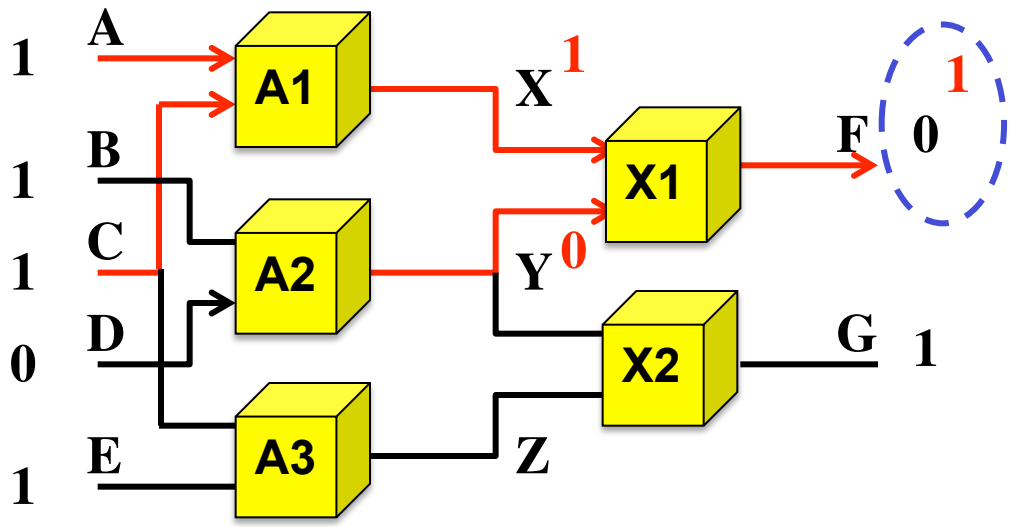
Idea: Select best value for each unassigned variable.

Example: First Iteration



- Conflicts / Constituent Kernels
  - none
- Best Kernel:
  - {}
- Best Candidate:
  - A1=G ∧ A2=G ∧ A3=G ∧ X1=G ∧ X2=G

Test: A1=G ∧ A2=G ∧ A3=G ∧ X1=G ∧ X2=G



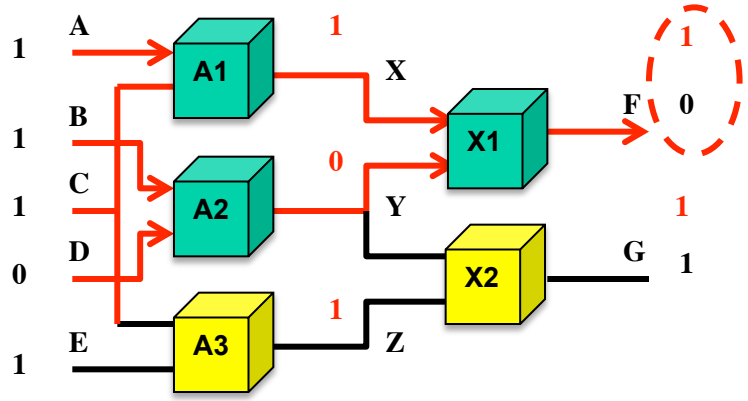
- Extract Conflict and Constituent Kernels:

¬ [A1=G ∧ A2=G ∧ X1=G]

A1=U ∨ A2=U ∨ X1=U

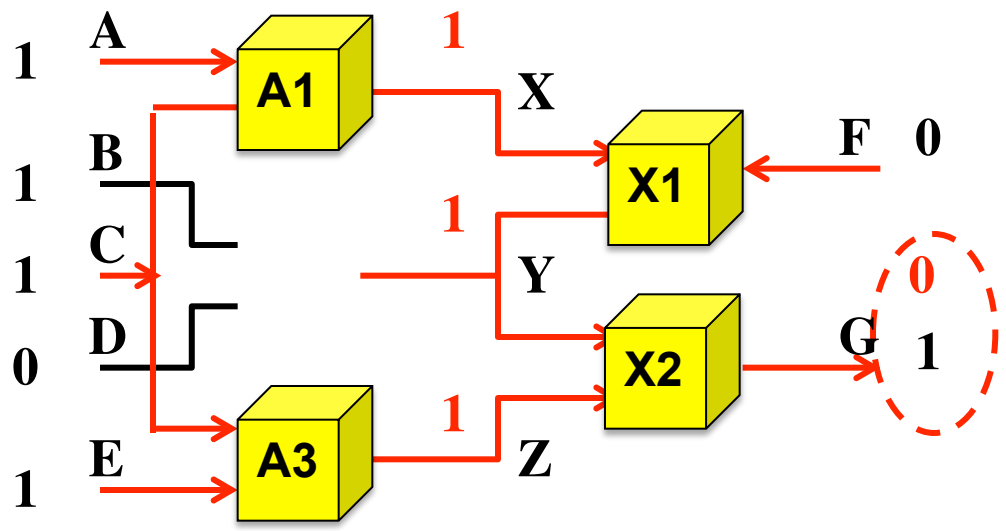
Second Iteration

- $P_{Xi=G} >> P_{Xi=U}$
- $P_{single} >> P_{double}$
- $P_{A2=U} > P_{A1=U} > P_{A3=U} > P_{X1=U} > P_{X2=U}$



- Conflicts ⇨ Constituent Kernels
  - {A1=U, A2=U, X1=U}
- Best Kernel:
  - {A2=U} (why?)
- Best Candidate:
  - A1=G ∧ A2=U ∧ A3=G ∧ X1=G ∧ X2=G

Test: A1=G ∧ A2=U ∧ A3=G ∧ X1=G ∧ X2=G



- Extract Conflict and Constituent Kernels:

¬ [A1=G ∧ A3=G ∧ X1=G ∧ X2=G]

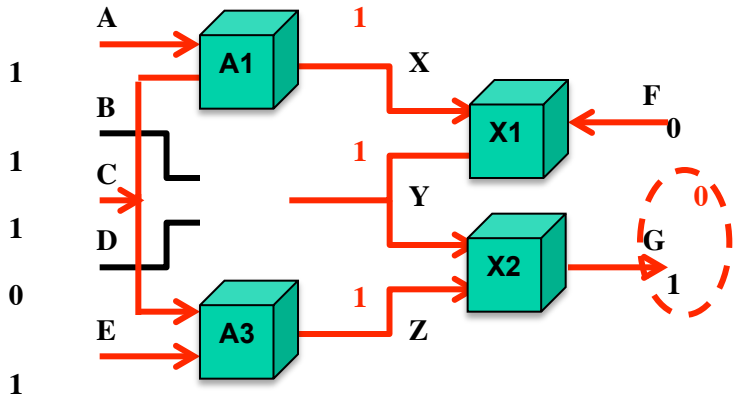
A1=U ∨ A3=U ∨ X1=U ∨ X2=U

Third Iteration

$P_{xi=G} >> P_{Xi=U}$

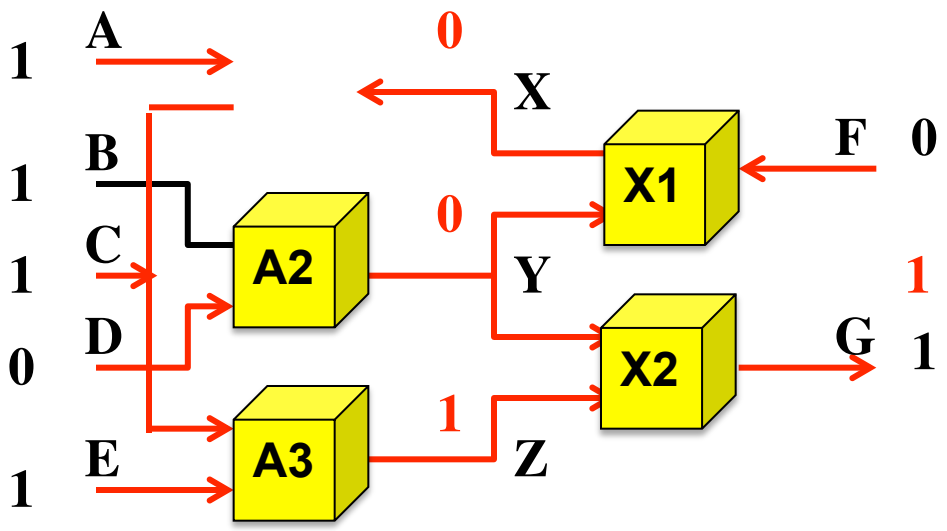
$P_{single} >> P_{double}$

$P_{A2=U} > P_{A1=U} > P_{A3=U} > P_{X1=U} > P_{X2=U}$



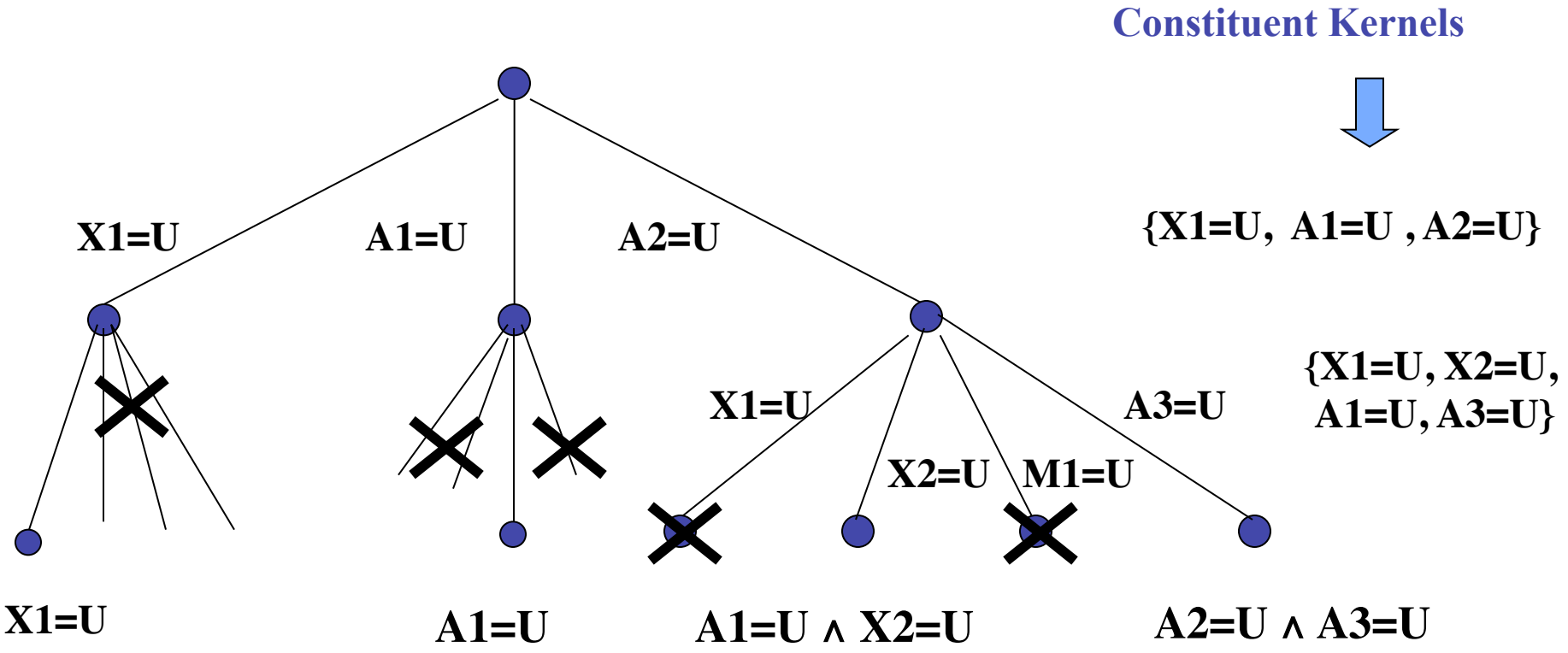
- Conflicts ⇨ Constituent Kernels
  - {A1=U, A2=U, X1=U}
  - {A1=U, A3=U, X1=U, X2=U}
- Best Kernel:
  - {A1=U}
- Best Candidate:
  - A1=U ∧ A2=G ∧ A3=G ∧ X1=G ∧ X2=G

Test: A1=U ∧ A2=G ∧ A3=G ∧ X1=G ∧ X2=G



- **Consistent!**

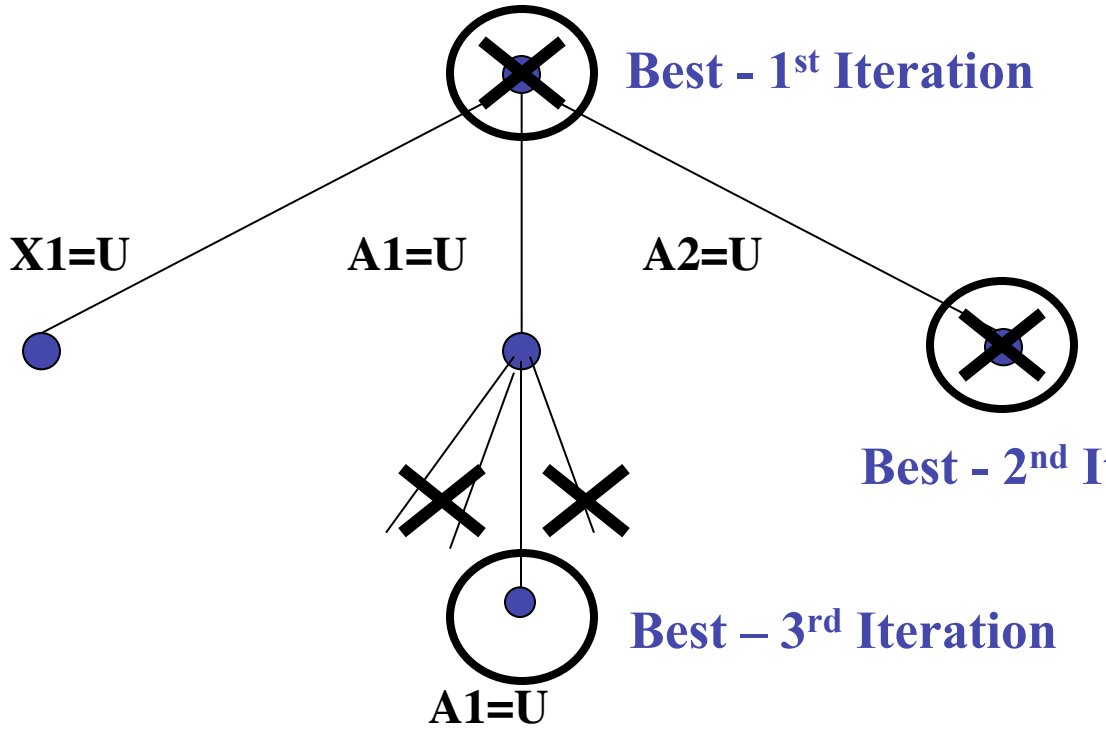
Generating The Best Kernel of The Known Conflicts



Insights:

- Kernels found by minimal set covering.
- Minimal set covering is an instance of breadth first search.

Generating The Best Kernel of The Known Conflicts



Constituent Kernels

{X1=U, A1=U, A2=U}

{X1=U, X2=U, A1=U, A3=U}

Insights:

- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.

➔ To find the best kernel, expand tree in best first order.

$P_{Xi=G} >> P_{Xi=U}$

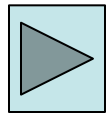
$P_{single} >> P_{double}$

$P_{A2=U} > P_{A1=U} > P_{A3=U} > P_{X1=U} > P_{X2=U}$

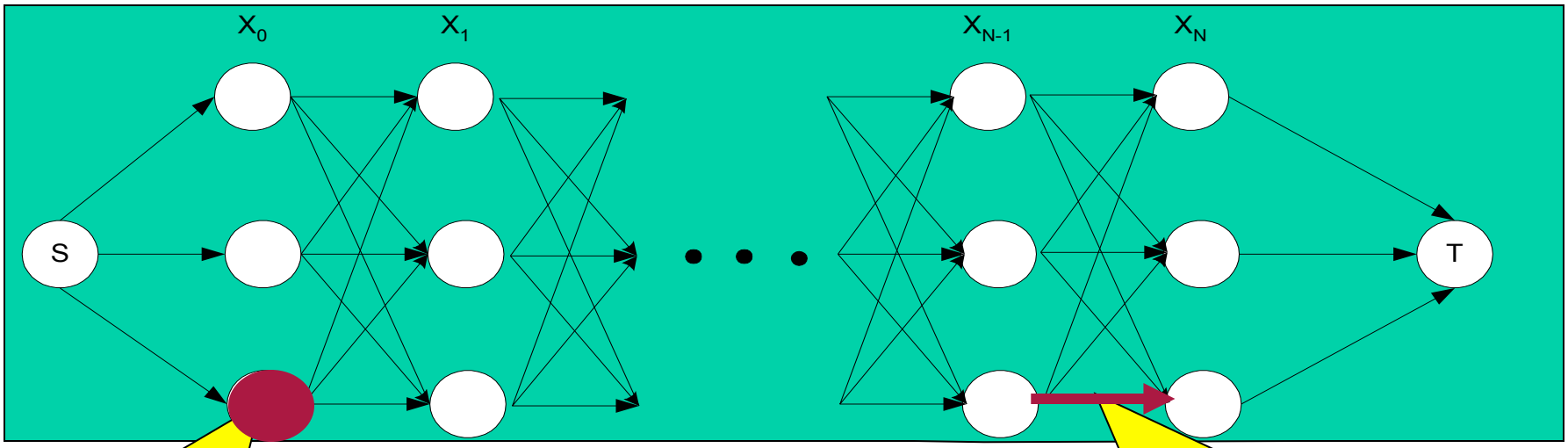
Performance: With and Without Conflicts

Problem Parameters				Constraint-based A* (no conflicts)		Conflict-directed A*			Mean CD-CB Ratio	
Dom Size	Dec Vars	Clau-ses	Clau-se lngth	Nodes Expanded	Queue Size	Nodes Expand	Queue Size	Conflicts used	Nodes Expanded	Queue Size
5	10	10	5	683	1,230	3.3	6.3	1.2	4.5%	5.6%
5	10	30	5	2,360	3,490	8.1	17.9	3.2	2.4%	3.5%
5	10	50	5	4,270	6,260	12.0	41.3	2.6	**0.83%**	1.1%
10	10	10	6	3,790	13,400	5.7	16.0	1.6	2.0%	**1.0%**
10	10	30	6	1,430	5,130	9.7	94.4	4.2	4.6%	5.8%
10	10	50	6	929	4,060	6.0	27.3	2.3	3.5%	3.9%
5	20	10	5	109	149	4.2	7.2	1.6	**13.0%**	**13.0%**
5	20	30	5	333	434	6.4	9.2	2.2	6.0%	5.4%
5	20	50	5	149	197	5.4	7.2	2.0	12.0%	11.0%

Robust, Model-based Execution (2): Model-based Programming w Hidden States

- Model-based programming and execution.
- Control sequencing.
- Mode estimation.
  - Estimates and kernels.
  - By divide and conquer (GDE).
  - Likely estimates (Conflict-directed A*).
  - Estimating probabilistic constraint automata 
- Mode reconfiguration and reactive planning.

Mode Estimation as Belief State Update for Concurrent PCA



- **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**

1. Infer most likely mode trajectories.
2. Infer distribution on likely mode assignments.

Hidden Markov Model

- S, M, Ω : Finite States, Actions & Observations
- T(s,μ,s'): State transition function

$$T : S \times \mathrm{M} \rightarrow \Pi(S)$$

- O(s',μ,o): Observation function

$$O : S \times \mathrm{M} \rightarrow \Pi(\Omega)$$

- $B^{t+1}(S)$: Belief state at time t.

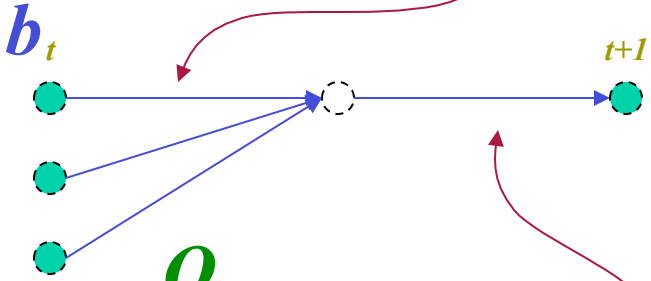
$$P(s^{t+1} \mid o^{<0,t>}, \mu^{<0,t>})$$

HMM Belief State Update

Propagate Dynamics:

$$\mathbf{P}(s_j^{t+1}|o^{<0,t>},\mu^{<0,t>}) = \sum_{s_i^t \in S^t} \left(\mathbf{P}(s_j^{t+1}|s_i^t,\mu^t)\mathbf{P}(s_i^t|o^{<0,t>},\mu^{<0,t-1>})\right)$$

T

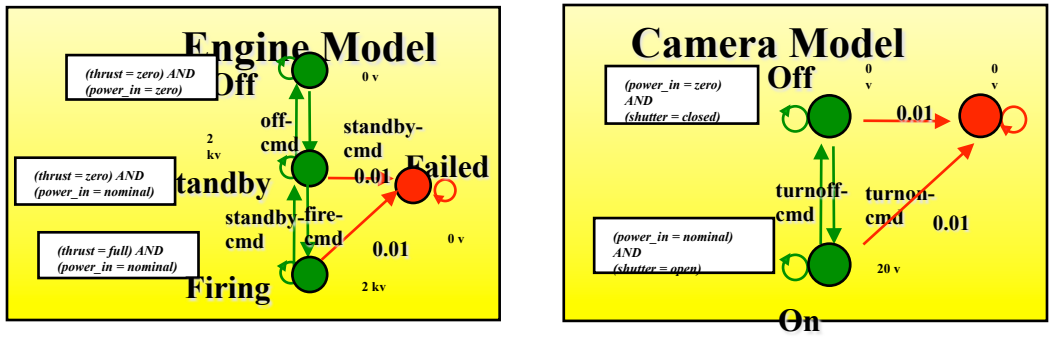


Update Based on Observations:

$$\mathbf{P}(s_j^{t+1}|o^{<0,t+1>},\mu^{<0,t>}) = \frac{\mathbf{P}(s_j^{t+1}|o^{<0,t>},\mu^{<0,t>}) \cdot \mathbf{P}(o^{t+1}|s_j^{t+1})}{\sum_{s_i^{t+1} \in S^{t+1}} \mathbf{P}(s_i^{t+1}|o^{<0,t>},\mu^{<0,t>})\mathbf{P}(o^{t+1}|s_i^{t+1})}$$

O

PCCA as HMMs



- PCCA encodes HMM compactly using concurrency and constraints.
- State abstracted to modes.

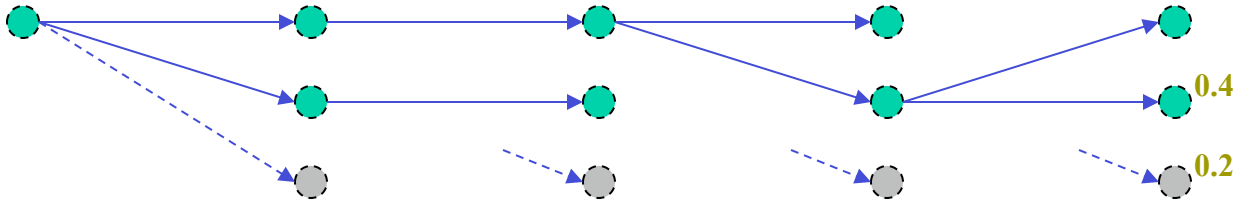
Assume:

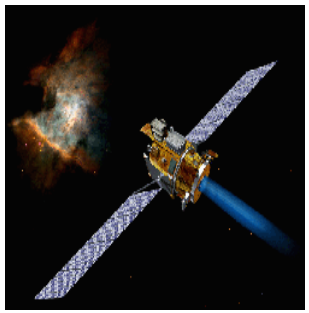
- Transitions only permitted on modes.
- Transitions are conditionally independent.
- For each time t, all consistent assignments are equally likely.

Approximating The Belief State

Best-first Trajectory Enumeration (BFTE):

[Williams and Nayak, AAAI-96][Kurien and Nayak, AAAI-00]
[Williams et al., IEEE '03]

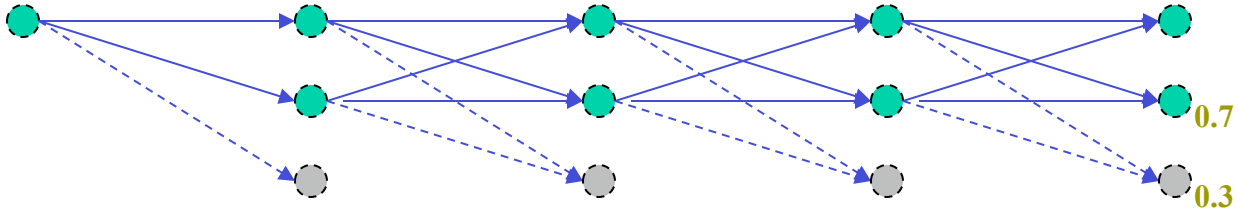


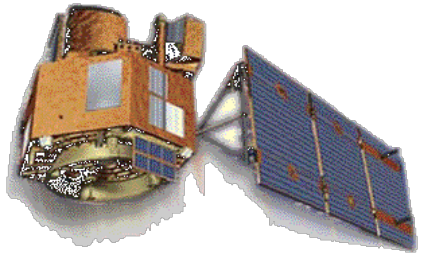


Deep Space One

- Best-first State Enumeration (BFSE):

[Martin, Williams and Ingham, AAAI-05]





Earth Observing One

– Improv accuracy through compact encoding.
– Accuracy improves runtime!

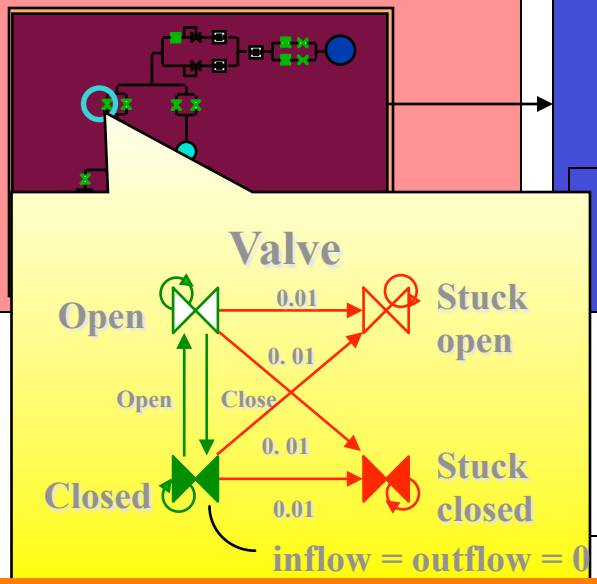
Robust, Model-based Execution (2): Model-based Programming w Hidden States

- Model-based programming and execution.
- Control sequencing.
- Mode estimation.
- Mode reconfiguration and reactive planning.

Control Program

- Executes concurrently
- Preempts
- Queries (hidden) states
- Asserts (hidden) state

System Model



Probabilistic Constraint Automata

Control Sequencer

State estimates

State goals

Mode Estimation

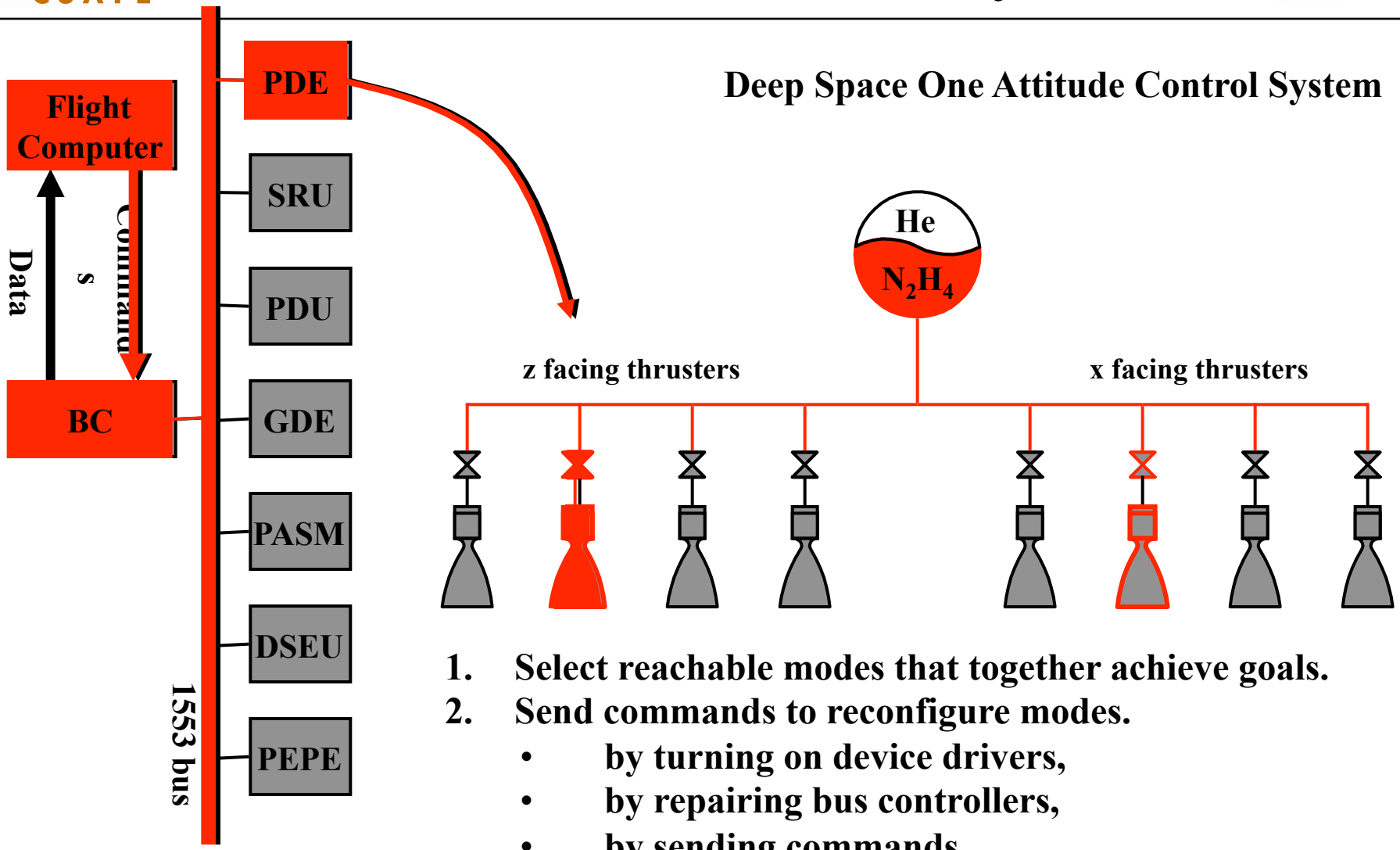
Mode Reconfiguration

Observations

Commands

Plant

DS 1 Attitude Control System

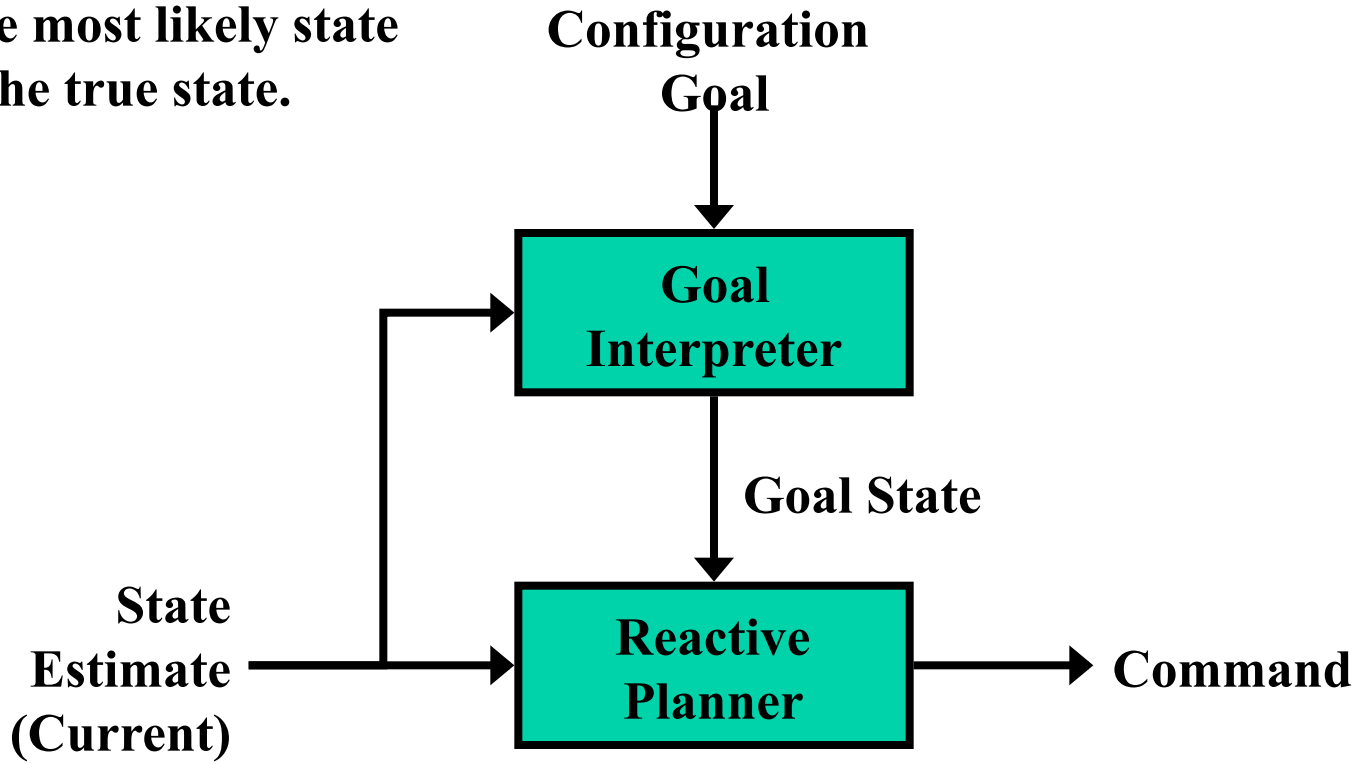


1. Select reachable modes that together achieve goals.
2. Send commands to reconfigure modes.
   - by turning on device drivers,
   - by repairing bus controllers,
   - by sending commands,
   - by powering down devices . . .

Mode Reconfiguration

Max likelihood assumption:

The most likely state is the true state.



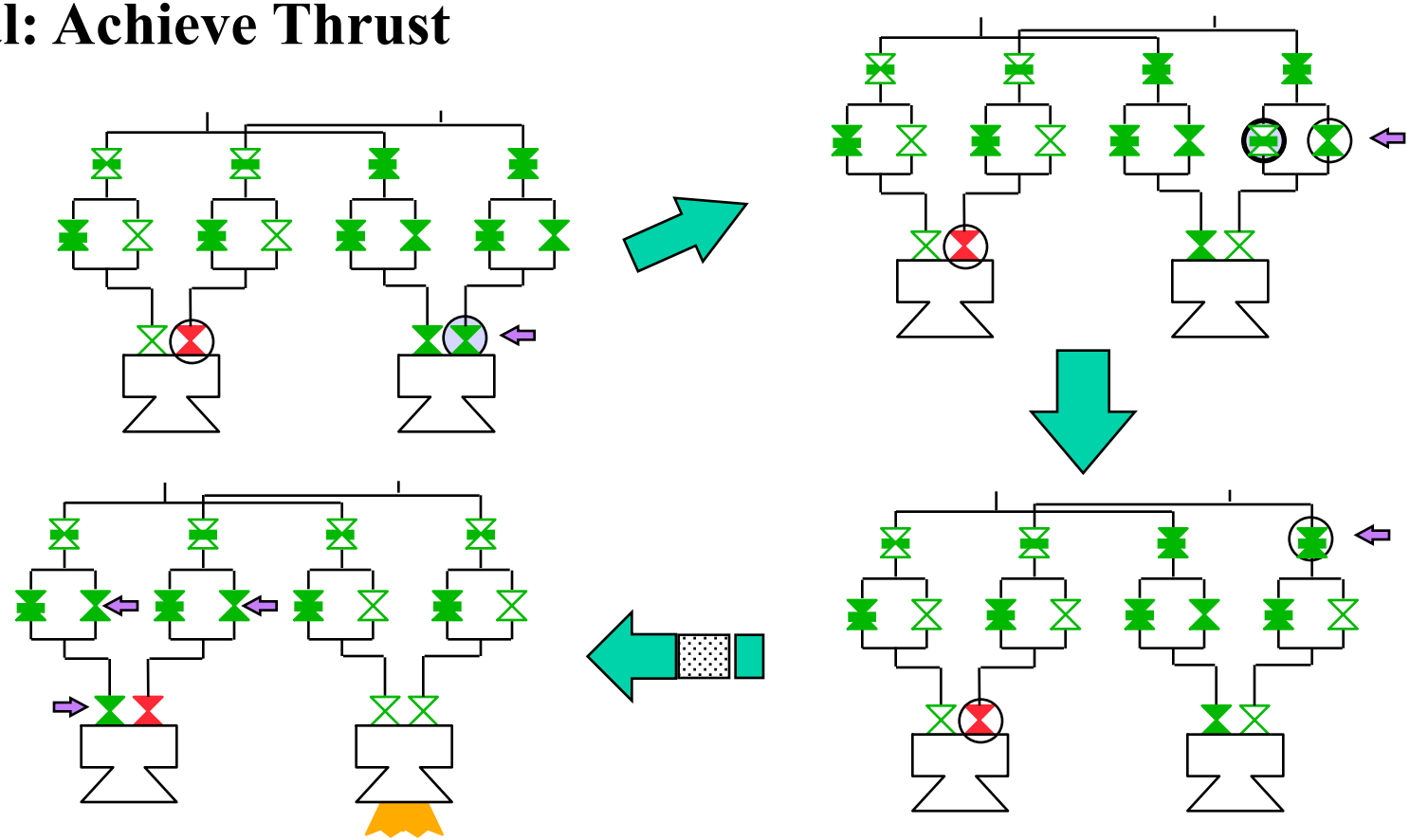
Model-based Programming of Intelligent Embedded Systems and Robotic Explorers
[Williams et al., IEEE' 03]

Reactive Planner for a Model-based Executive
[Williams & Nayak, IJCAI 97]

Goal Interpretation via Conflict Learning

arg max $R_t(Y)$
s.t. $\Psi(X,Y)$ entails $G(X,Y)$
s.t. $\Psi(X,Y)$ is consistent
Y are reachable modes

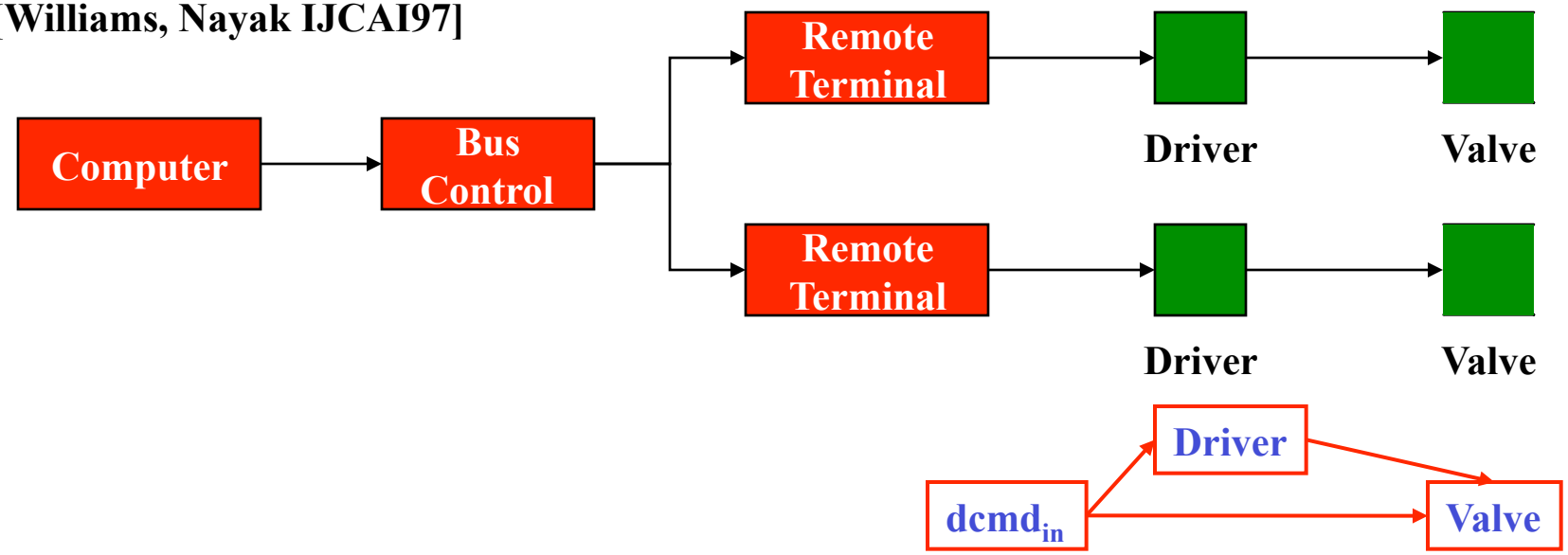
Goal: Achieve Thrust



A *conflict* is a partial assignment to mode variables that prevents the goal (entails the negation of the goal).

Reactive Planning:
Engineered systems tend not to have loops

[Williams, Nayak IJCAI97]



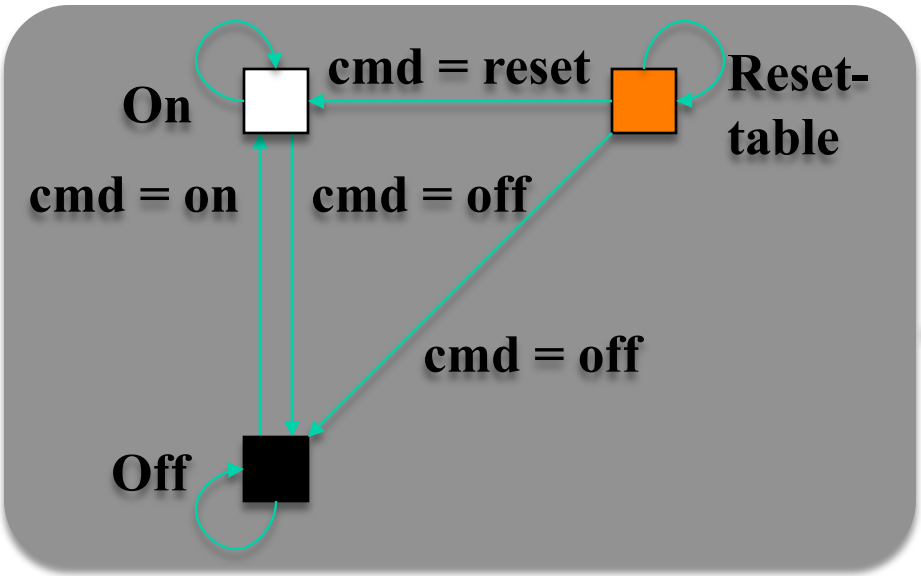
➔ **Work conjunctive goals upstream G from outputs to inputs. Wht?**

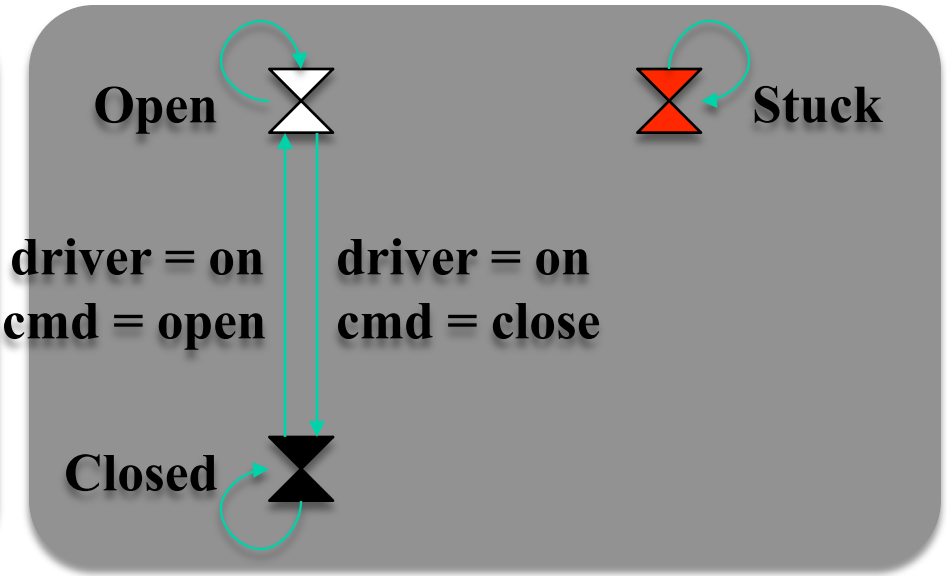
– **Define: Causal Graph *G* of compiled transition system *S***
  - **vertices are state variables.**
  - **edge from v_i to v_j if v_j' s transition is conditioned on v_i.**

– **Requirement: The causal graph is acyclic.**

Compile models to goal-directed policies

[Williams, Nayak, IJCAI 97]





Goal State

Current State: Driver	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Reset-table	cmd = reset	cmd = off

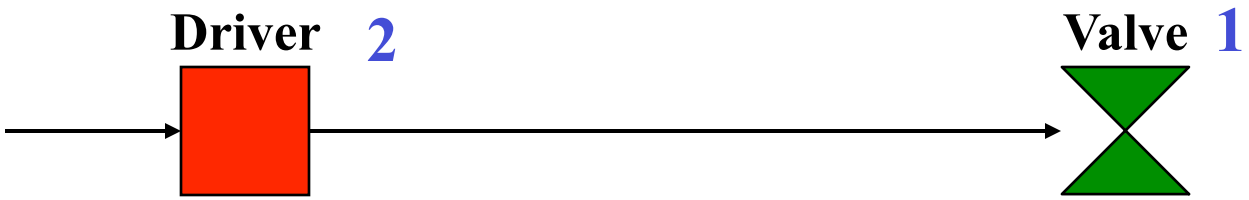
Goal State

Current State: Valve	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	failed	failed

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = off, Valve = open



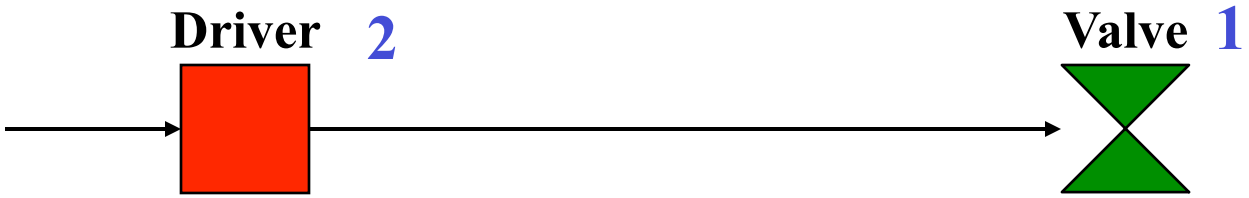
Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = off, Valve = open



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

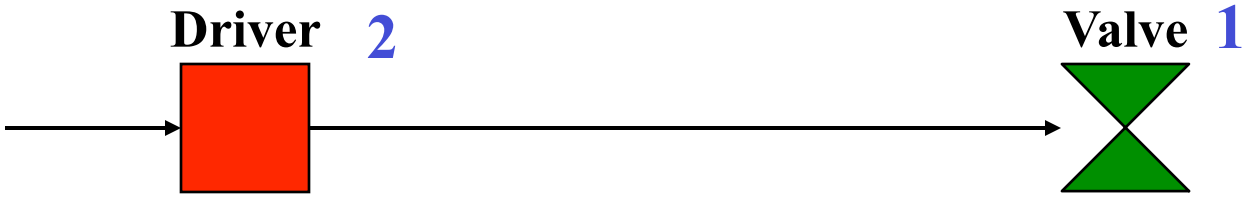
Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = off, Valve = open

Send:
cmd = on



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: ???

Send:
cmd = on

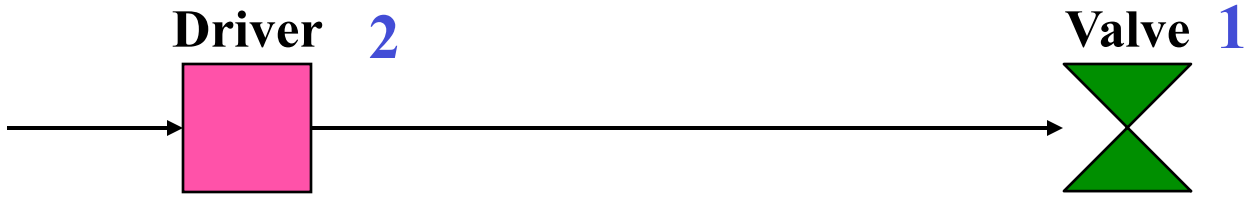
Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = resettable, Valve = open



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

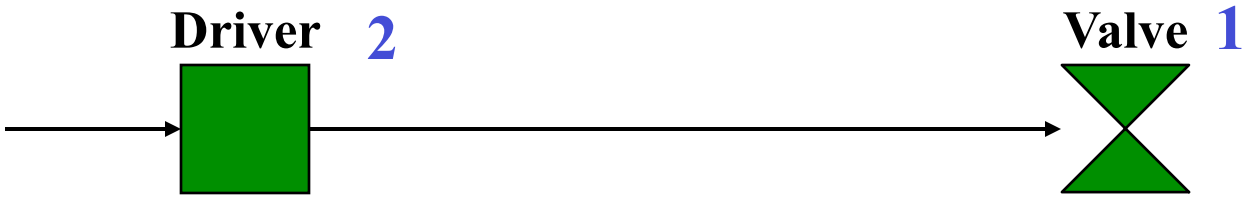
Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = on, Valve = open

Send
cmd = close



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

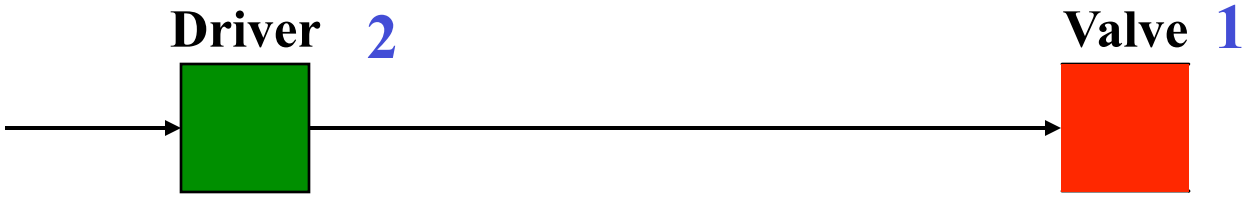
Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = on, Valve = closed

Send cmd = off



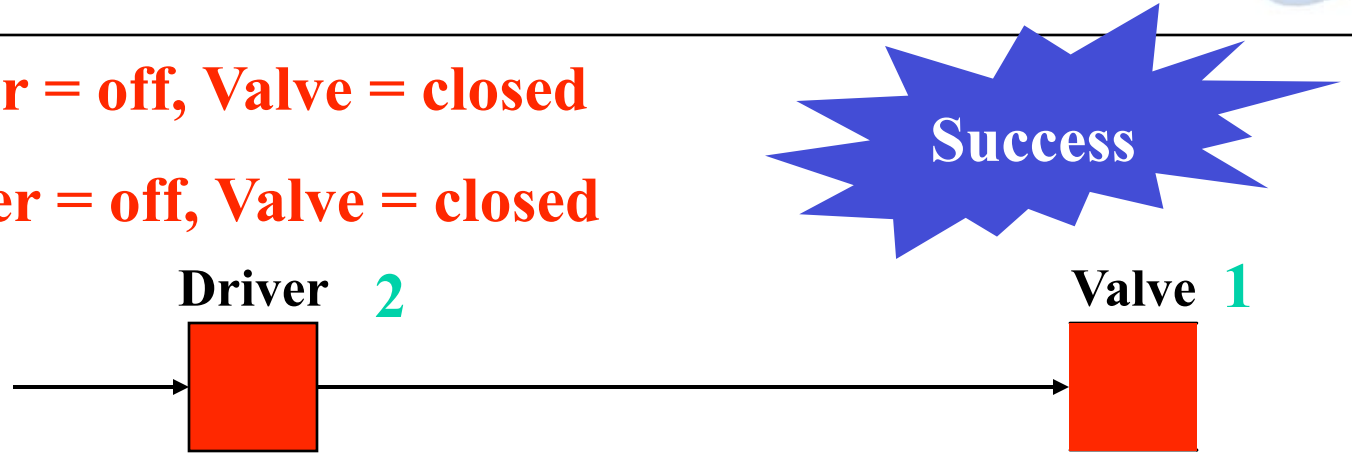
Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = off, Valve = closed



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

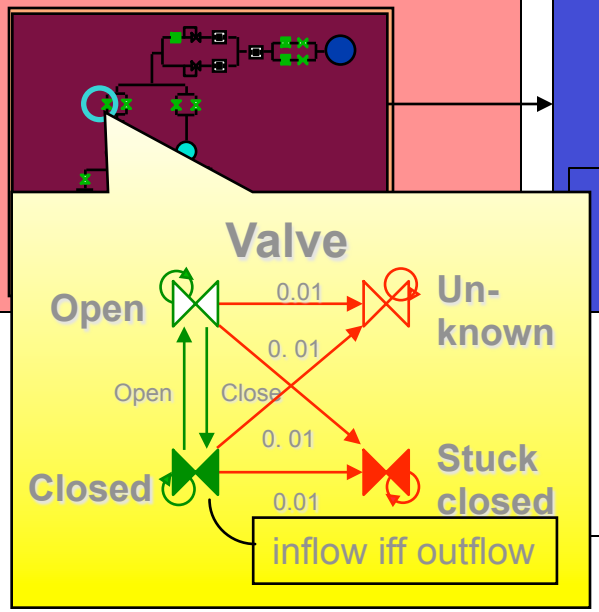
RMPL Model-based Program

Titan Model-based Executive

Control Program

- Executes concurrently
- Preempts
- Queries (hidden) states
- Asserts (hidden) state

System Model



Control Sequencer

State estimates

State goals

Mode Estimation

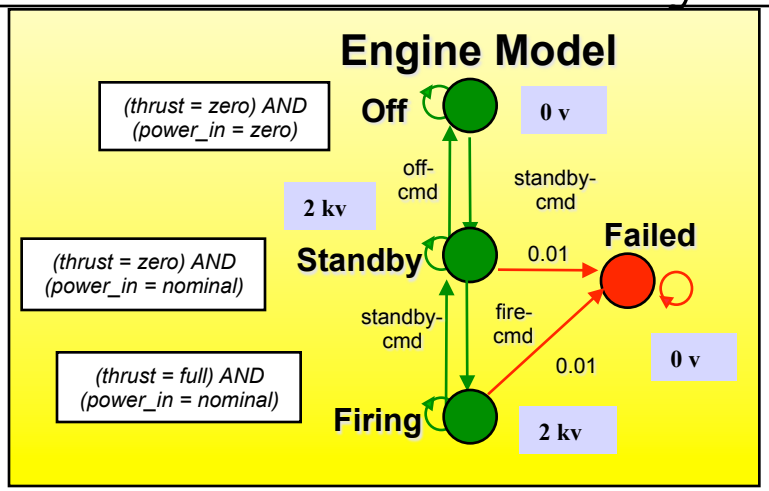
Mode Reconfiguration

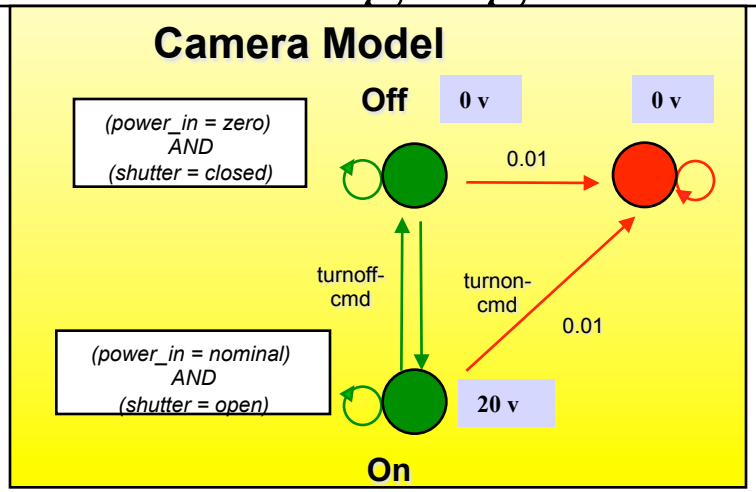
Observations

Commands

Plant

Variants on Probabilistic Constraint Automata define a Family of RMPL Languages





- Complex, discrete behaviors
  - modeled through concurrency, hierarchy and timed transitions.
- Anomalies and uncertainty
  - modeled by probabilistic transitions
- Physical interactions
  - modeled by discrete and continuous constraints

Approach: Model-based Programming and Execution

- An embedded programming language elevated to operations on hidden state and choice
- A language executive that achieves robustness by reasoning over constraint-based models.

Tuesday: Coordination and dynamic scheduling.

Wednesday: Model-based programming with hidden state.