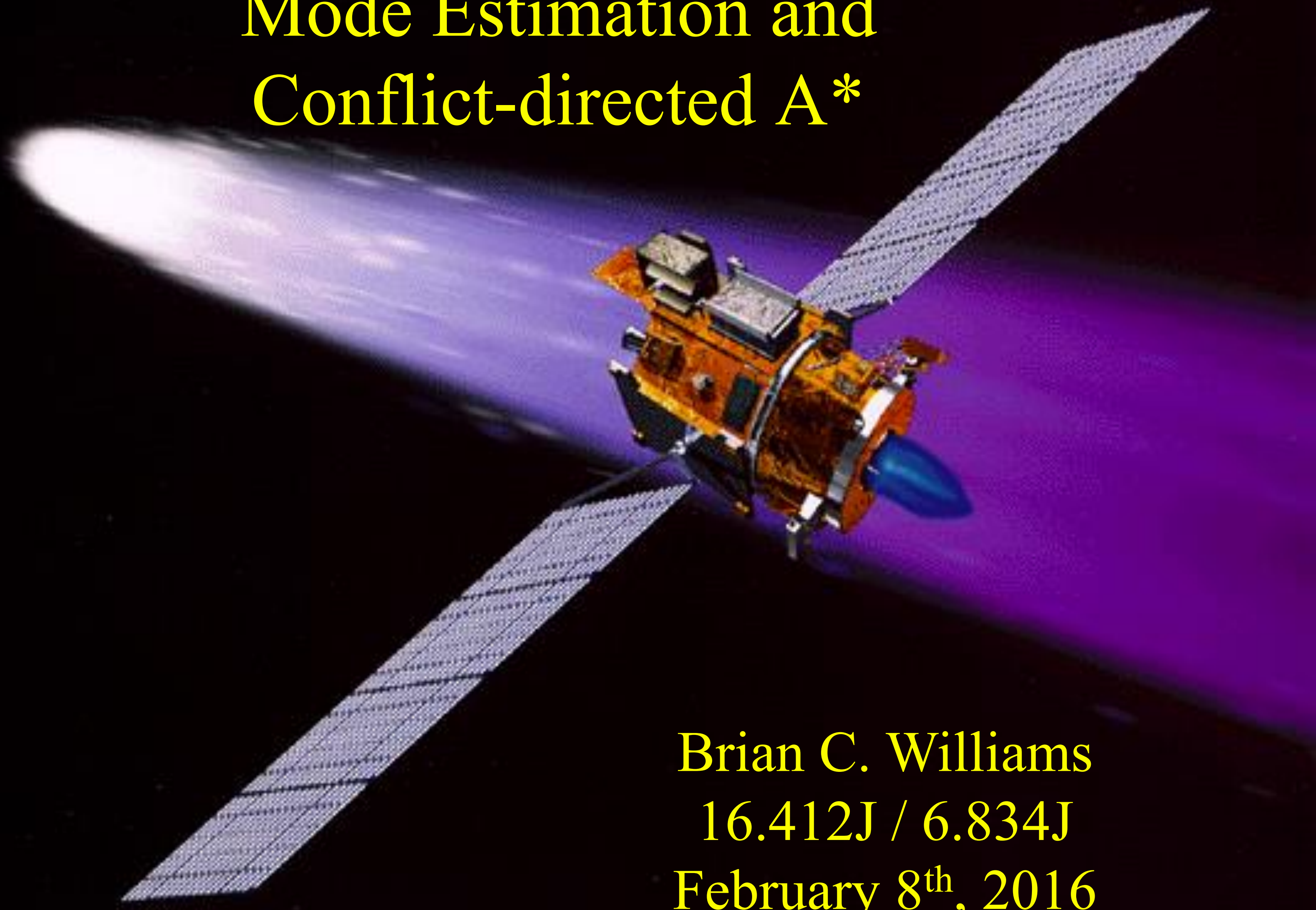
Programs that Monitor Hidden State: Mode Estimation and Conflict-directed A*



Brian C. Williams

16.412J / 6.834J

February 8th, 2016

courtesy of JPL

Today's Assignments

Problems Sets:

- Problem Set #1, Out today, due Wed, February 17th.

Readings:

- **Today**: 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.*
- **Wednesday:** Same.

Background:

- 16.410/13 Lectures on Informed Search, Constraint Satisfaction, Propositional Satisfiability and Diagnosis.

Outline

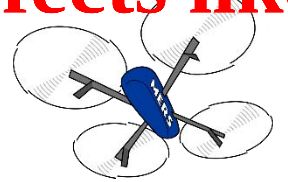
- Programs that monitor
  and control hidden states.
- Consistency-based Diagnosis

Programs that Monitor State

```
method run() {
    sequence {
        uav.launch();
        uav.fly_to_base_station();
        uav.pick_up_med_kit();
        uav.fly_to_hikers();
        uav.drop_off_med_kit();
    }
}
```

Base Station

Actions have preconditions & effects like before



Program sequence of actions in RMPL

A Traditional Reactive Programming Language

Expressions:

1. s — Conditions on sensors
2. u — Assignments to control variables

Control constructs:

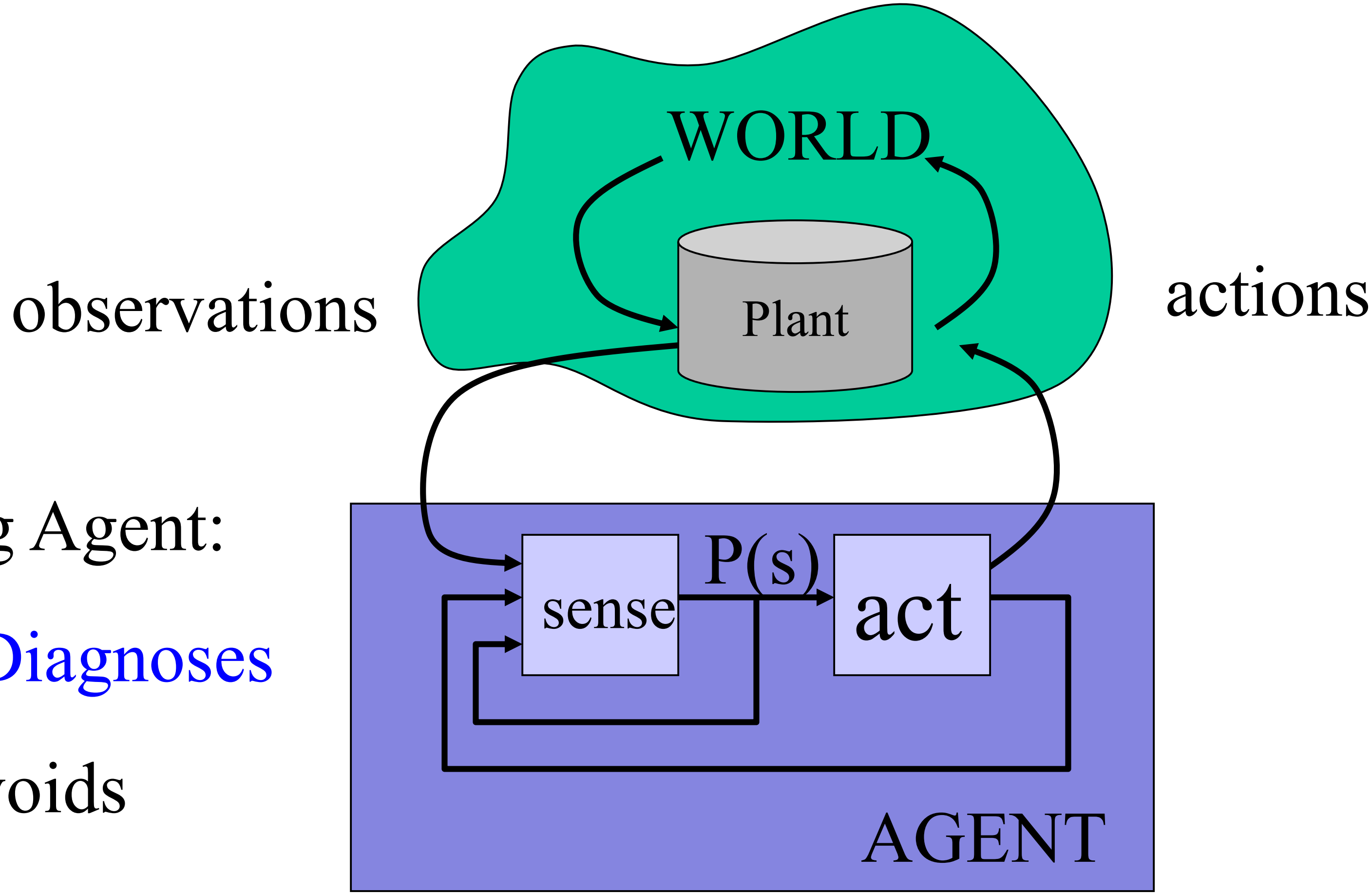
1. u — Control assignments
2. If s next A — Conditional execution
3. Unless s next A — Preemption
4. A, B — Full concurrency
5. Always A — Iteration

where A, B are programs.

Action Model: PDDL

Self-Repairing Agent:

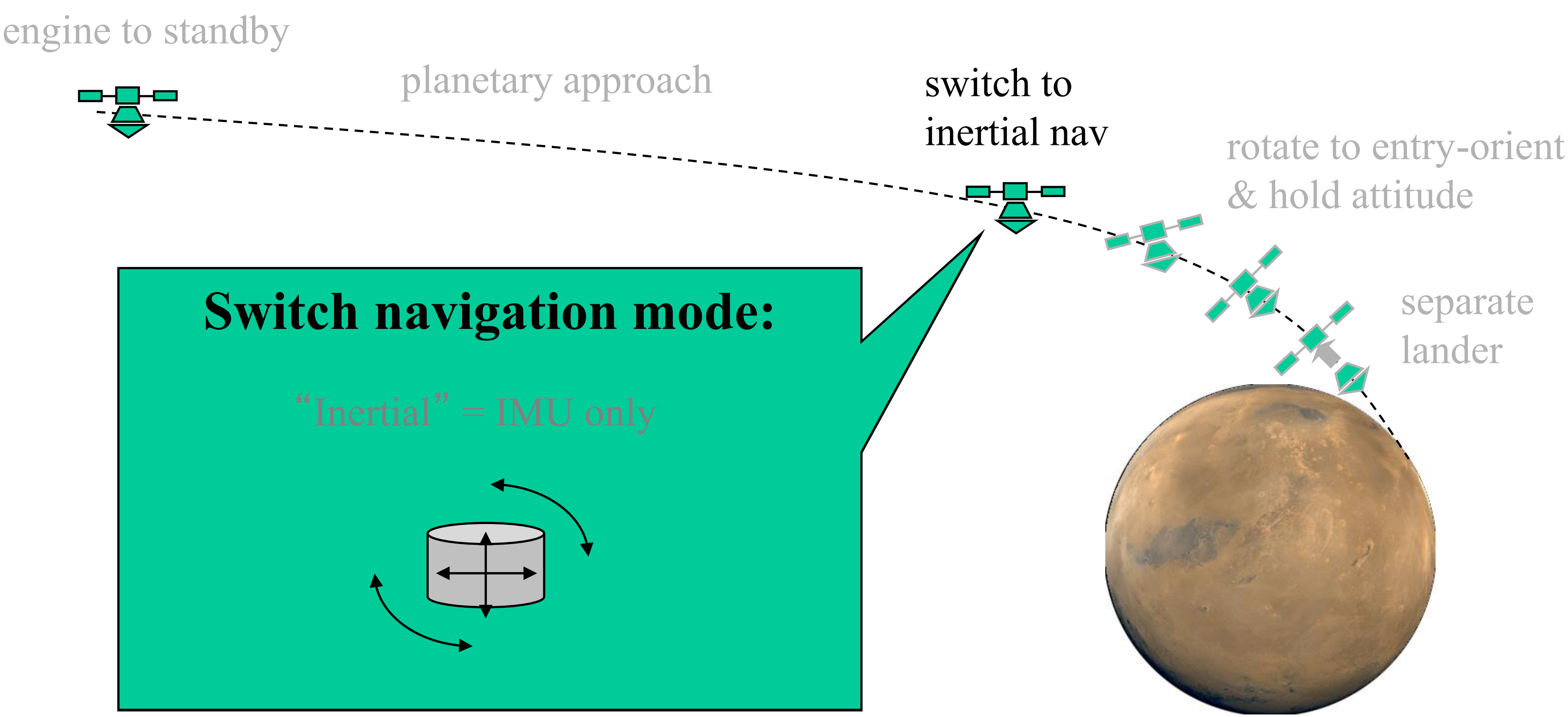
- Monitors & Diagnoses
- Repairs & Avoids
- Probes and Tests



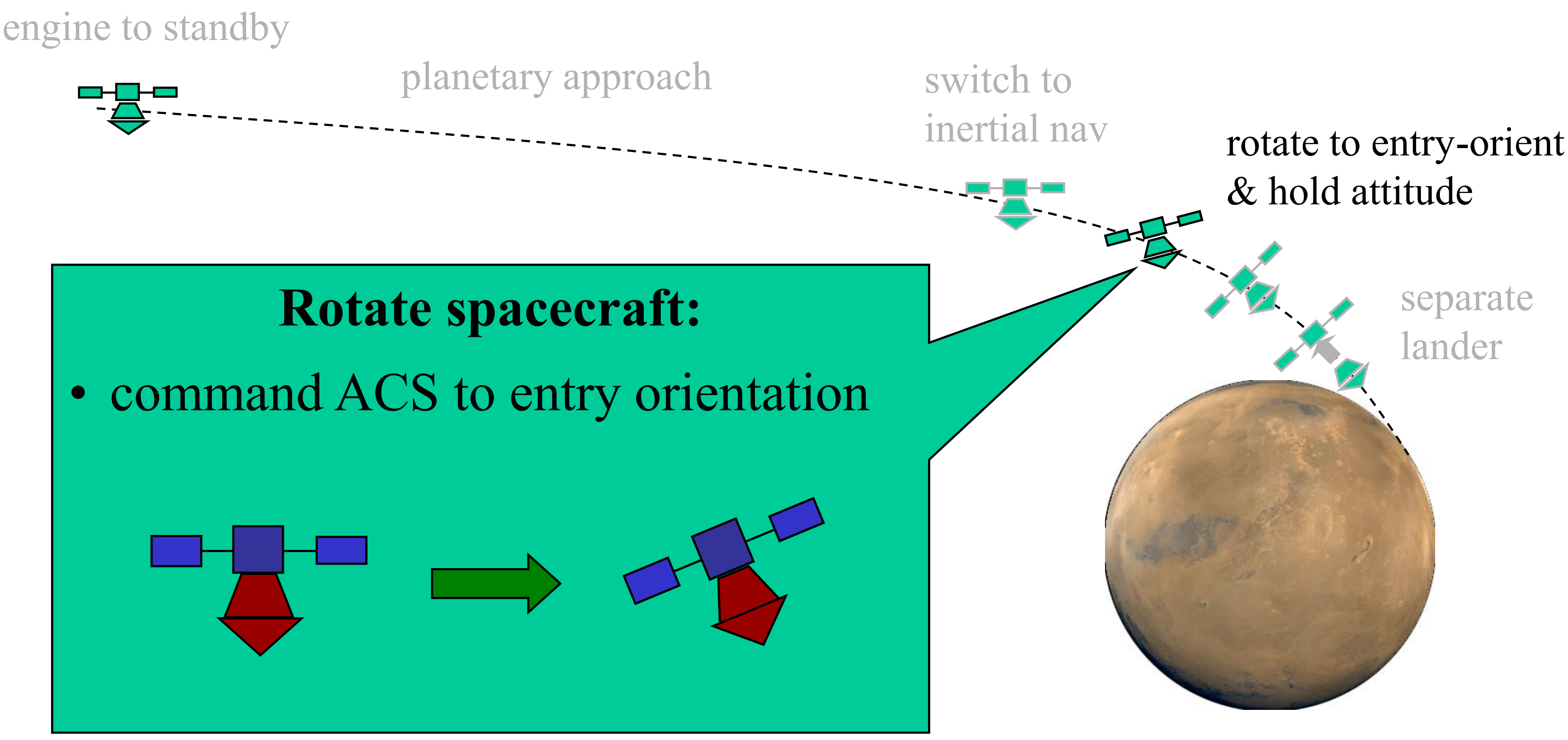
Symptom-directed

Programs that Monitor and Control Hidden (Failure) States

Mission Storyboards
Specify Evolving States



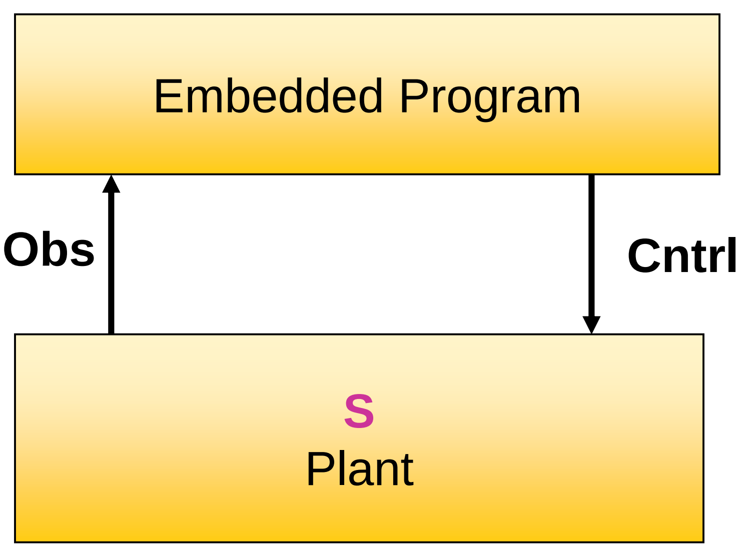
Mission Storyboards Specify Evolving States



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

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

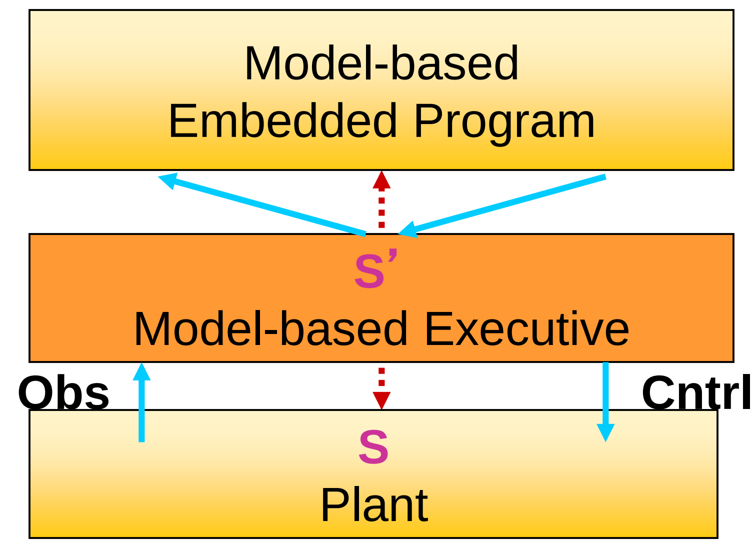
- Read sensors
- Set actuators



Programmer maps between state and sensors/actuators.

Model-based programs evolve abstract states through direct interaction:

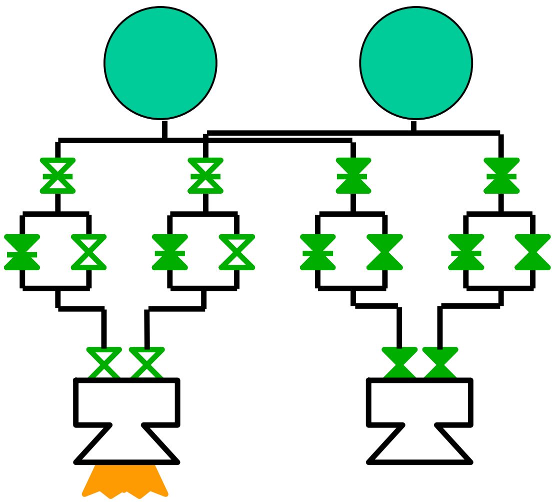
- Read abstract state
- Write abstract state



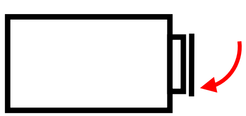
Model-based executive maps between state and sensors/actuators.

Model-based Programming of a Saturn Orbiter

Turn camera off and engine on



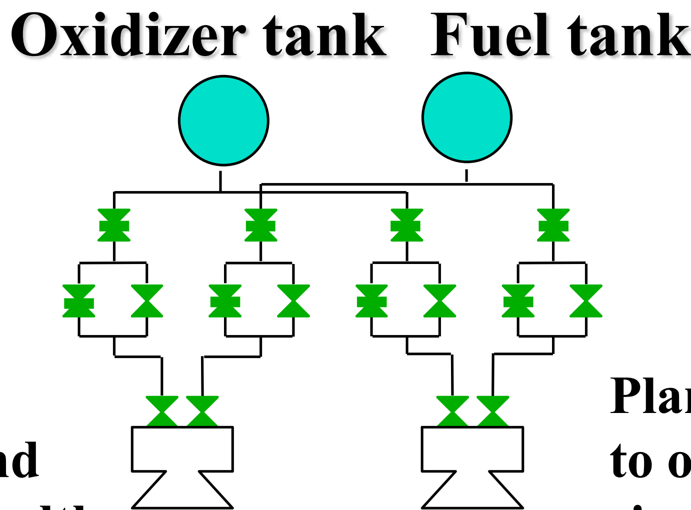
EngineA EngineB



Science Camera

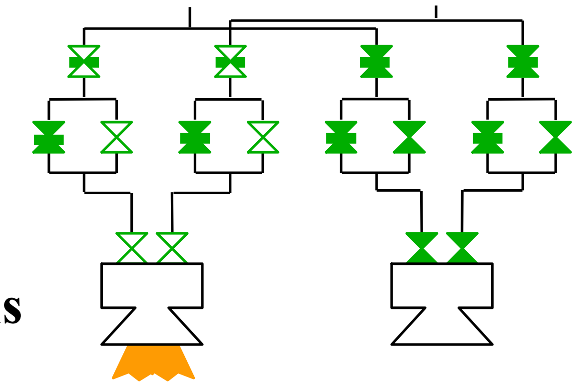
```
OrbitInsert()::

 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 . . . .

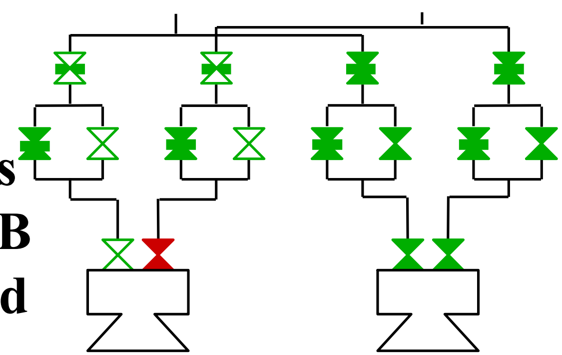


Deduces that thrust is off, and the engine is healthy

Plans actions to open six valves

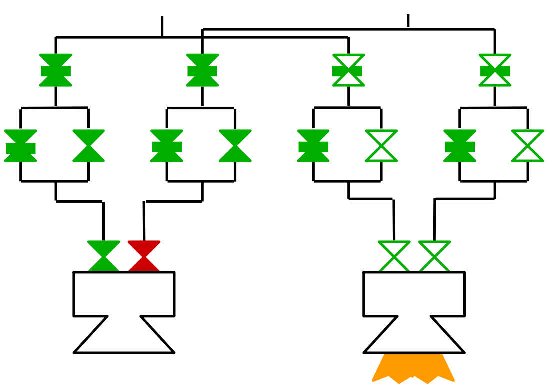


Deduces that a valve failed - stuck closed



Prog: EngineB = Thrusting

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



Plant Model:
Probabilistic Constraint Automata (PCA)

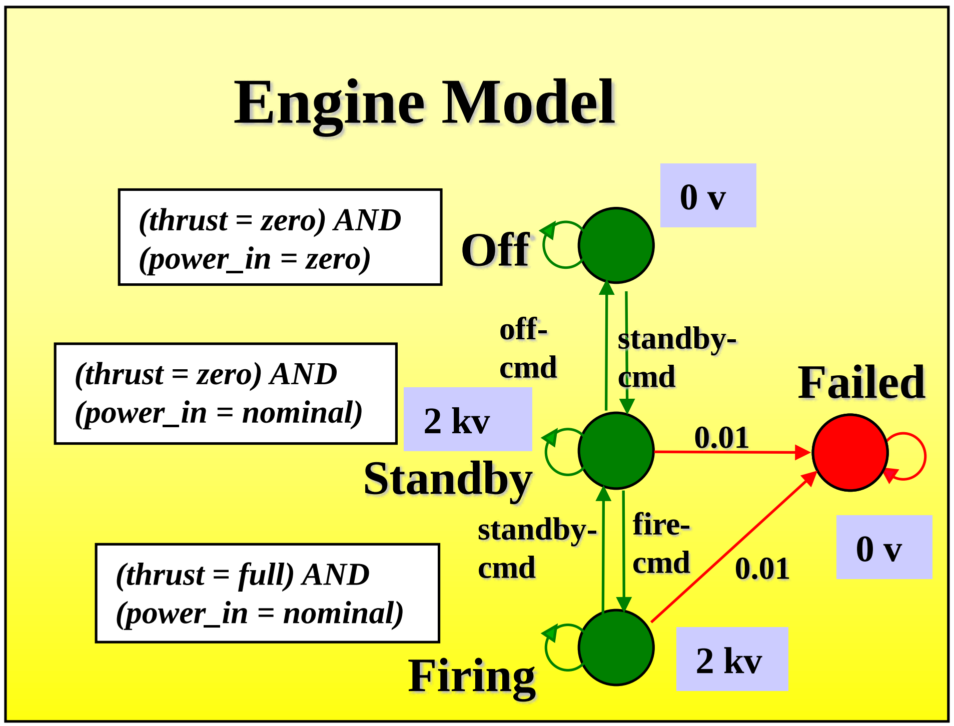
component modes…

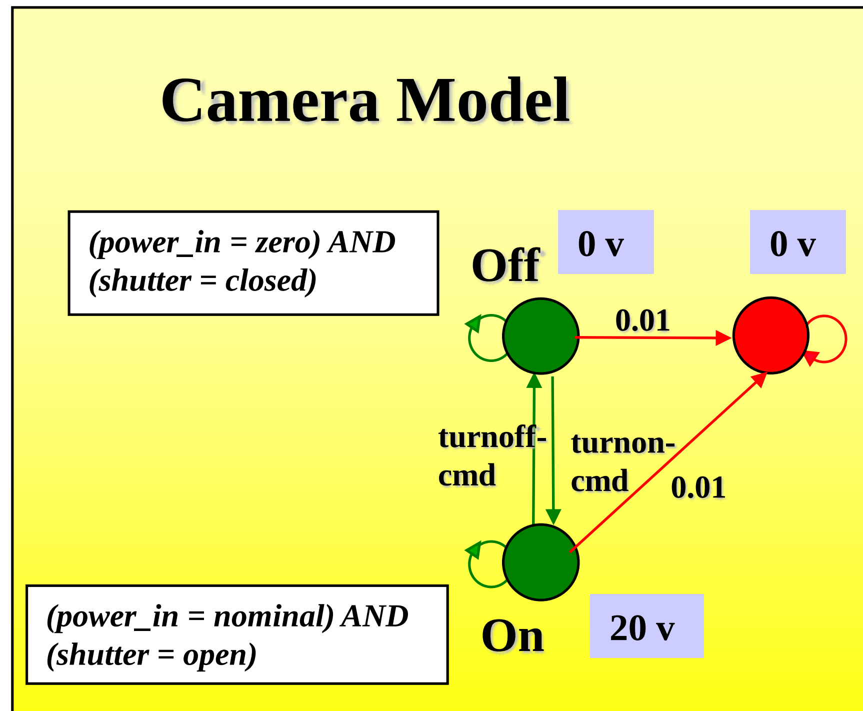
described by finite domain constraints on variables…

guarded deterministic and probabilistic transitions

cost / reward & prior distribution

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





one per component … operating concurrently

A Reactive Model-based Programming Language (RMPL)

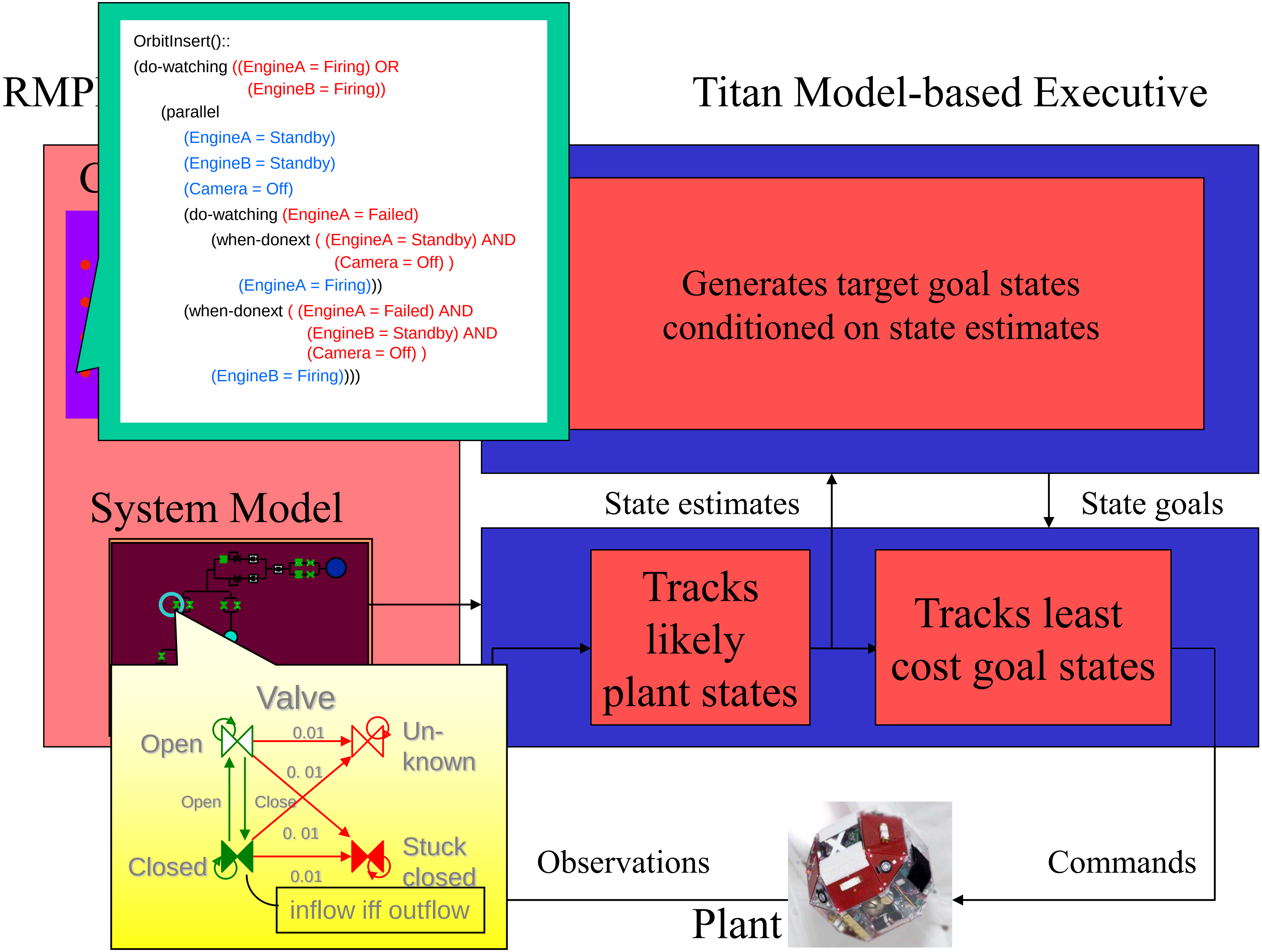
Idea: A concurrent constraint language (e.g. TCC/HCC [Saraswat et al.])

- whose constraints c operate on the state of the plant s, and
- replaces the constraint store with a model-based controller:

1. c[s]	Primitive constraint on state
2. If c[s] next A	Conditional execution
3. Unless c[s] next A	Preemption
4. A, B	Full concurrency
5. Always A	Iteration

Action Model:

Probabilistic
Constraint
Automata


RMPL
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
Valve
Open
Un-known
Open
Close
Closed
Stuck closed
0.01
0. 01
0. 01
0.01
inflow iff outflow
Observations
Commands
Plant

Mode Estimation:

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

Mode Reconfiguration:

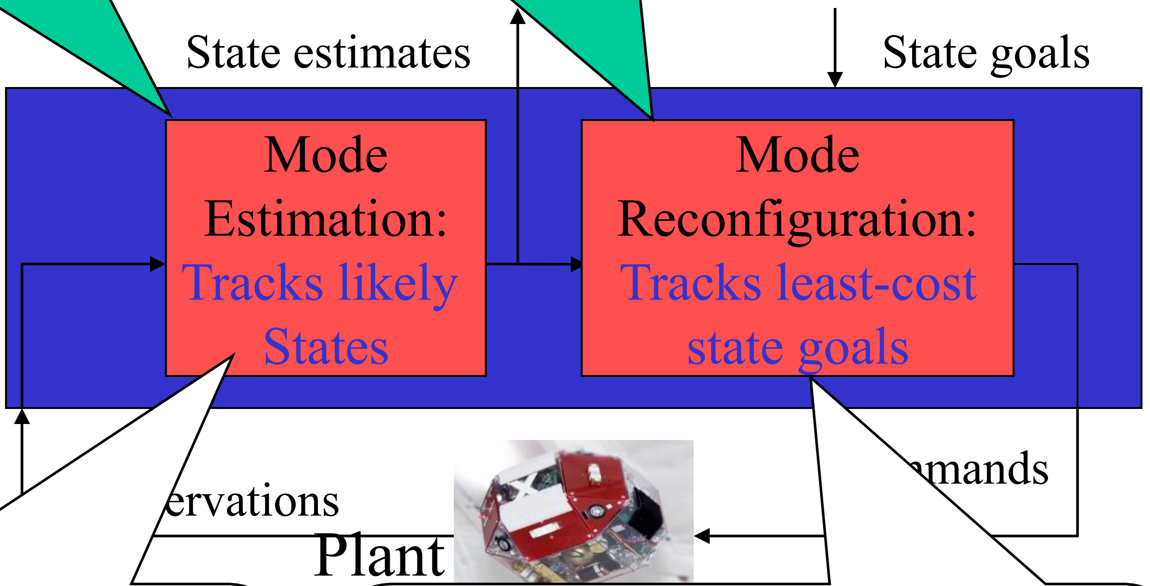
Select a least cost set of commandable component modes that entail the current goal, and are consistent.

Optimal CSP:

arg min f(x)

s.t. C(x) is satisfiable

D(x) is unsatisfiable



arg min $P_t(Y \mid Obs)$

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

arg max $R_t(Y)$

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

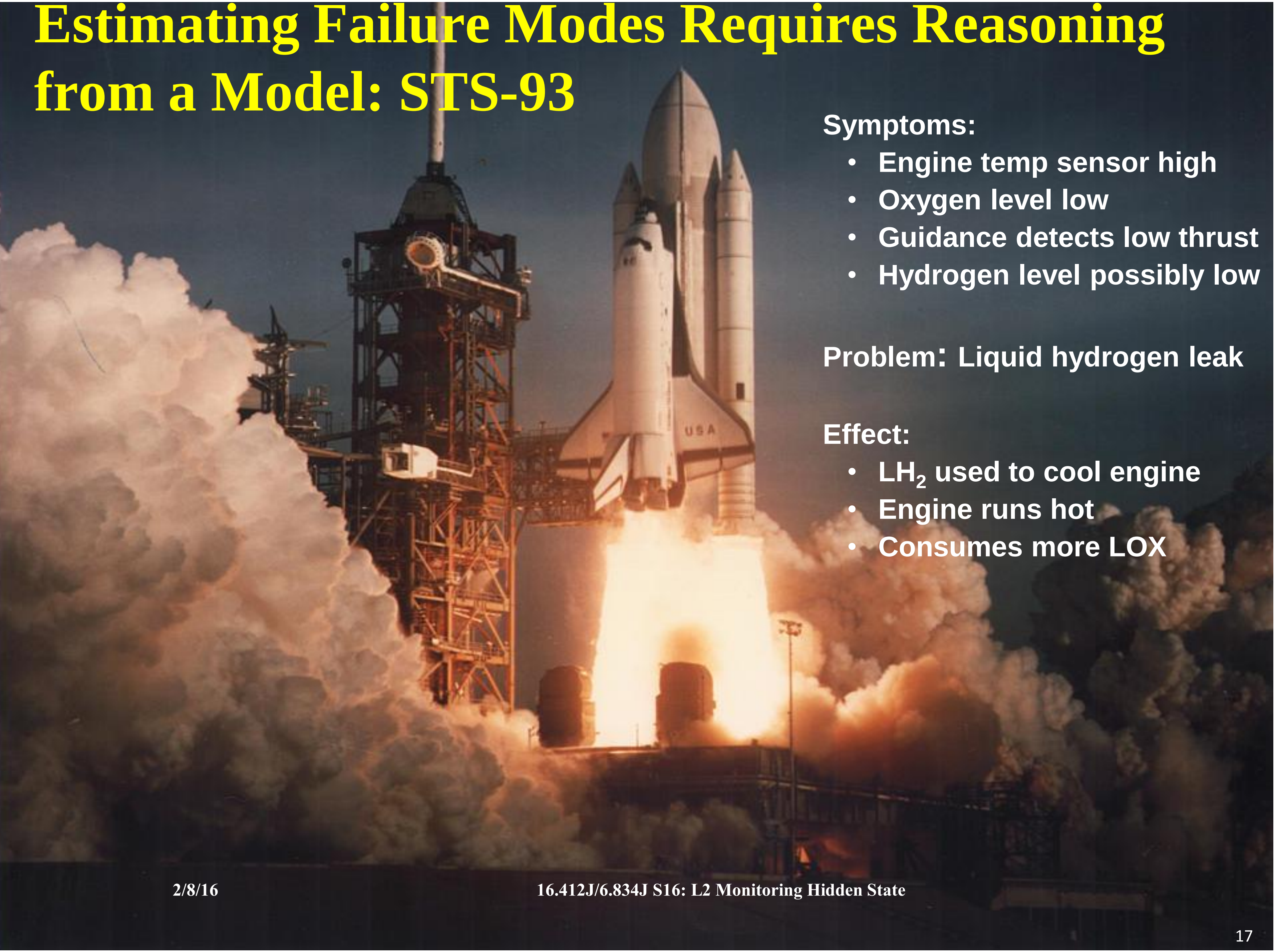
s.t. $\Psi(X,Y)$ is consistent

s.t. Y is reachable

Outline

- Programs that monitor and control hidden states.
- Consistency-based Diagnosis

Estimating Failure Modes Requires Reasoning from a Model: STS-93



Symptoms:

- Engine temp sensor high
- Oxygen level low
- Guidance detects low thrust
- Hydrogen level possibly low

Problem: Liquid hydrogen leak

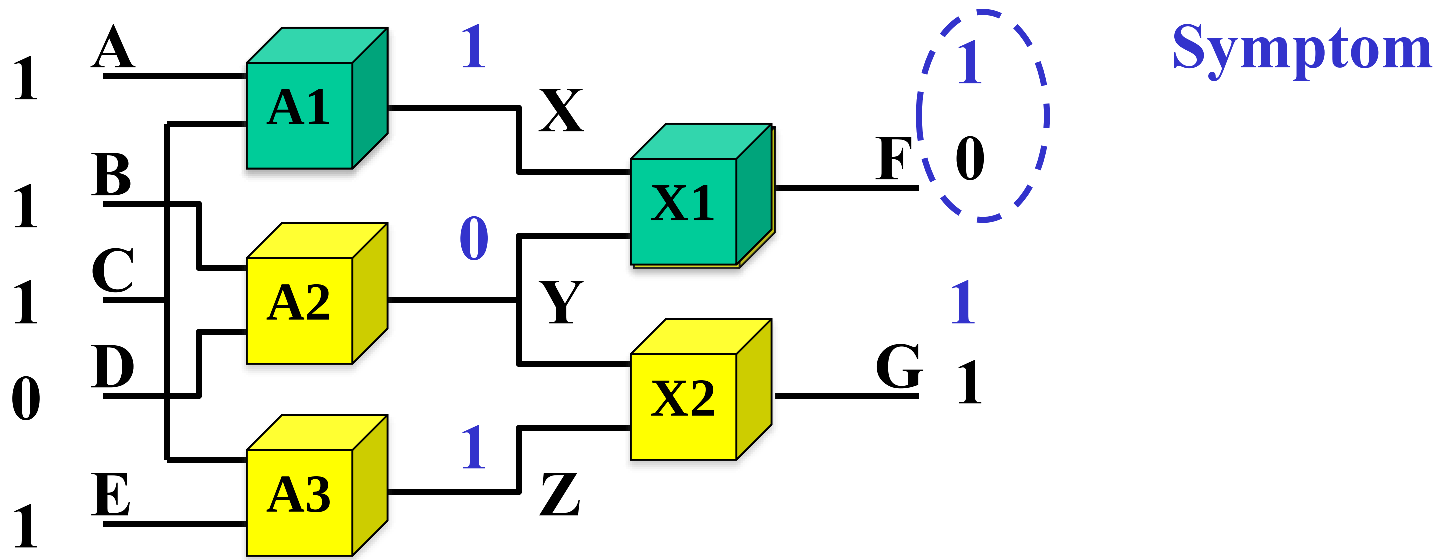
Effect:

- LH_2 used to cool engine
- Engine runs hot
- Consumes more LOX

Model-based Diagnosis

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

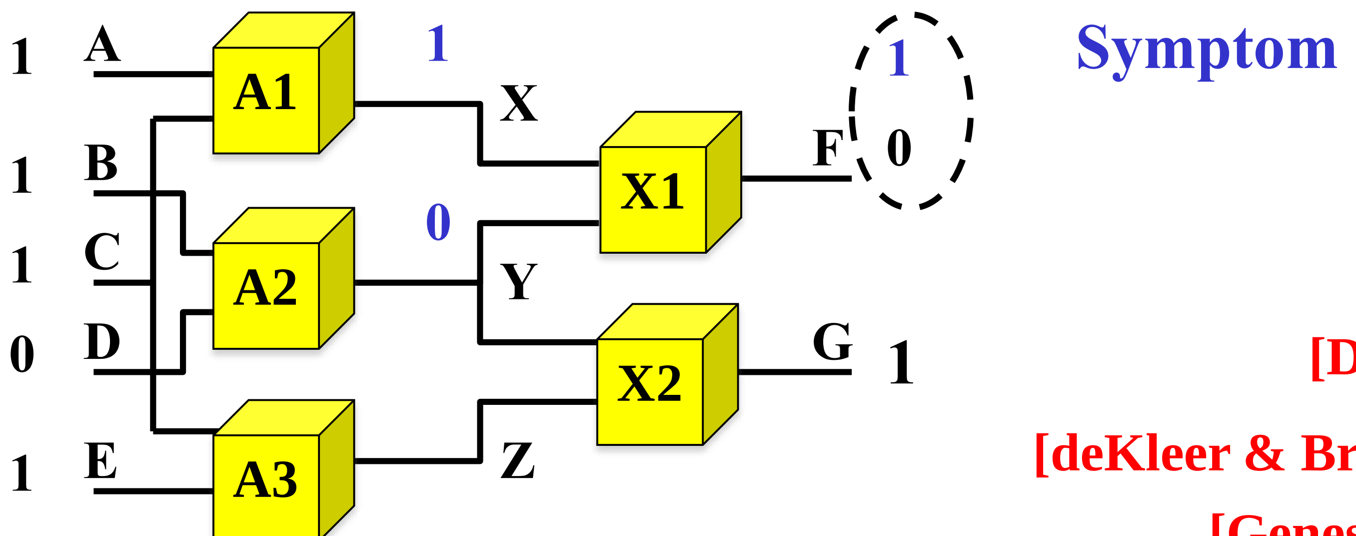
Output: Diagnoses that account for the symptoms.



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.



[Davis, 84]

[deKleer & Brown, 83]

[Geneserth, 84]

Issue 3: Multiple Faults Occur

APOLLO 13

- three shorts, tank-line and pressure jacket burst, panel flies off.

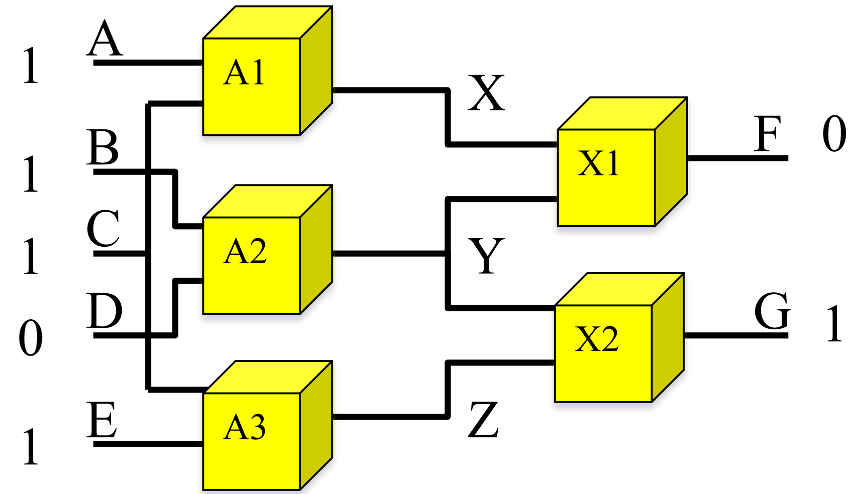
➔ Diagnosis = Mode Assignment

➔ Solution: Divide & Conquer

Solution: Identify all Combinations of Consistent "Unknown" Modes

And(i):

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



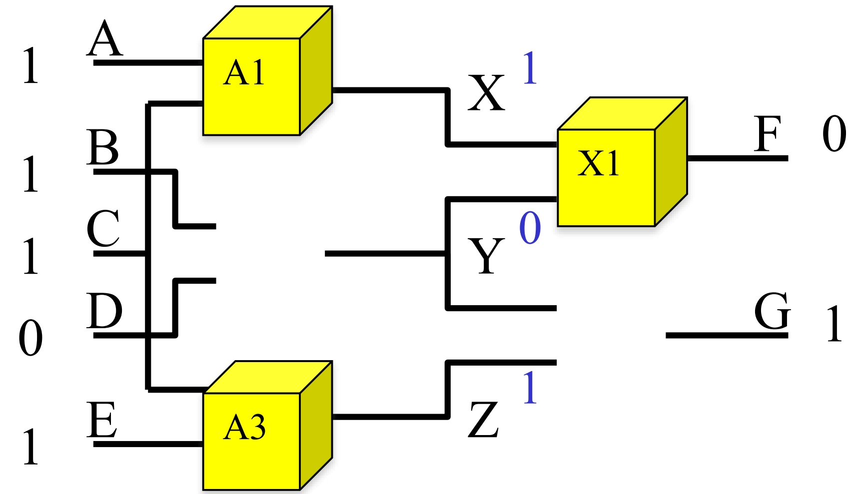
Candidate = {A1=G, A2=G, A3=G, X1=G, X2=G}

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

Solution: 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.

Mode Estimation

Given:

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

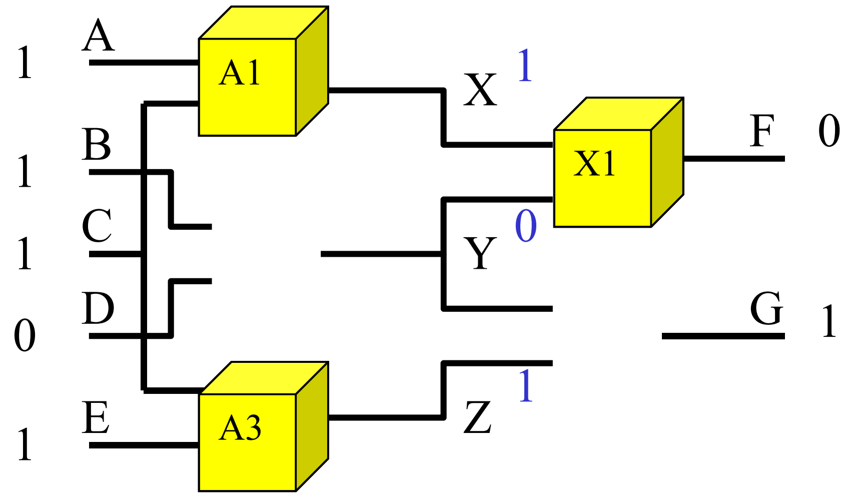
And(i):

G(i):

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

U(i): *No Constraint*

• All behaviors are associated with modes.
• All components have "unknown Mode" U, whose assignment is never mentioned in any constraint.



Return: All mode estimates

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

Models in Propositional State Logic

And(i):

- G(i):
  Out(i) = In1(i) AND In2(i) $\quad$ $i{=}G \Rightarrow \{[In1(i){=}1 \wedge In2(i){=}1] \text{ iff } Out(i){=}1\}$
- U(i): *No Constraint*

Or(i):

- G(i):
  Out(i) = In1(i) OR In2(i) $\quad$ $i{=}G \Rightarrow \{[In1(i){=}1 \vee In2(i){=}1] \text{ iff } Out(i){=}1\}$
- U(i): *No Constraint*

$X \in \{1,0\}$ $\quad$ $X{=}1 \vee X{=}0$

$\neg[X{=}1 \wedge X{=}0]$

$\neg(i{=}G) \vee \neg(In1(i){=}1) \vee Out(i){=}1$

$\neg(i{=}G) \vee \neg(In2(i){=}1) \vee Out(i){=}1$

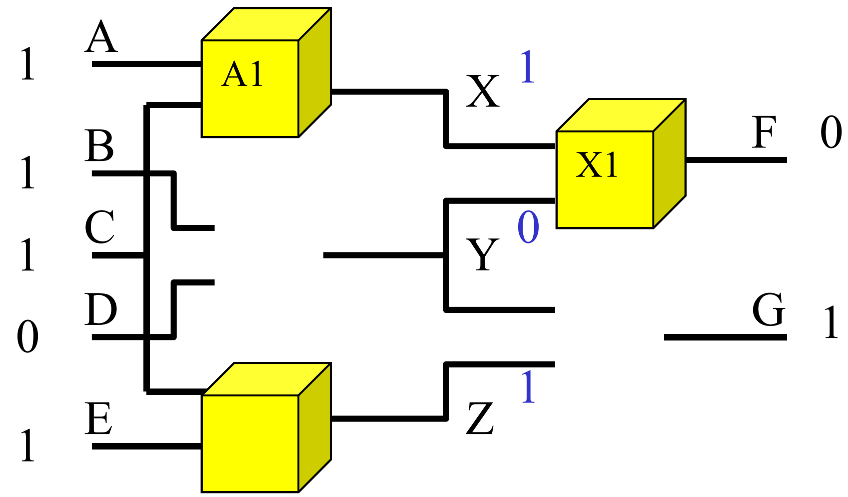
$\neg(i{=}G) \vee \neg(In1(i){=}0) \vee \neg(In2(i){=}0) \vee Out(i){=}0$

Outline

- Programs that monitor and control hidden states.
- Consistency-based Diagnosis
  - Encoding diagnoses compactly using kernels.
  - Using conflicts to divide and conquer.

Need Compact Encoding

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



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

As more constraints are relaxed, candidates are more easily satisfied.

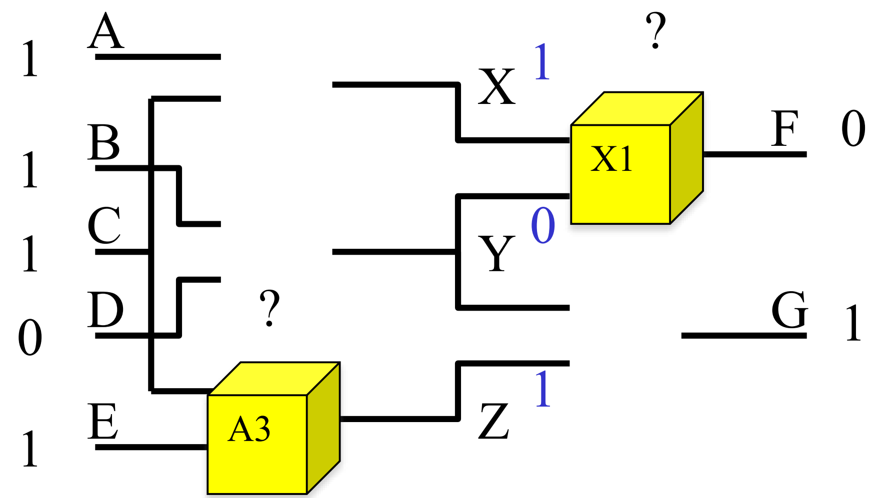
➔Typically an exponential number of diagnoses (mode estimates).

How do we encode solutions compactly?

Partial Diagnoses

Partial Diagnosis

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



Partial Diagnosis:

A partial mode assignment M,
all of whose full extensions are diagnoses.

- M "removes all symptoms."

Diagnoses with common assignments:

{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}

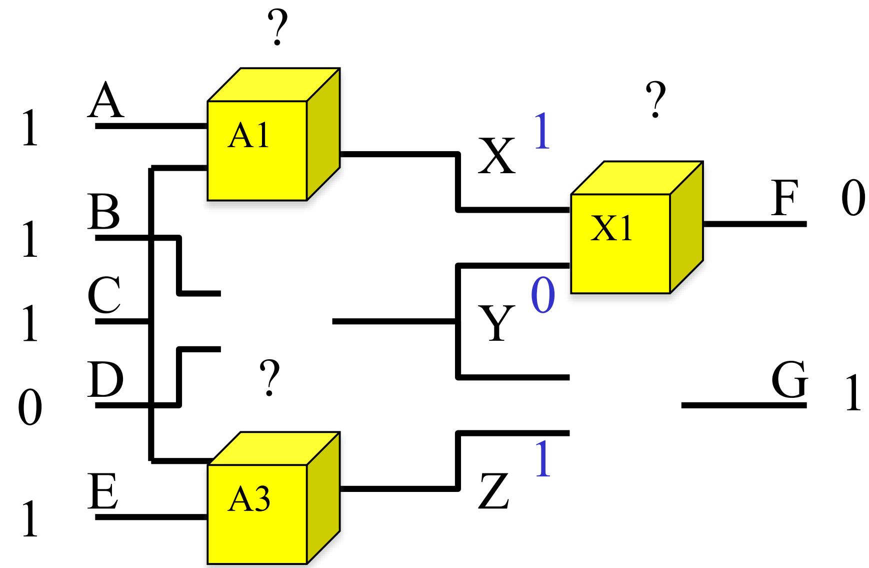
Kernel Diagnoses

Partial Diagnosis

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

Kernel Diagnosis

{A2=U, X2=U}



Partial Diagnosis:

A partial mode assignment M, all of whose full extensions are diagnoses.

Kernel Diagnosis:

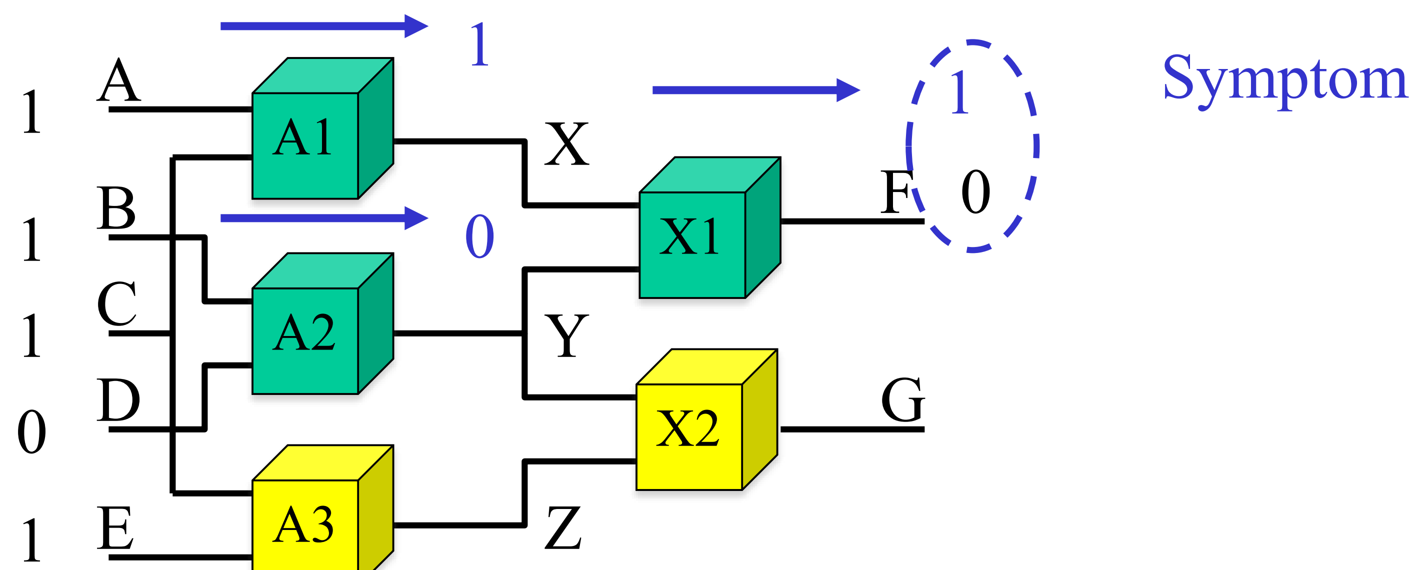
The smallest partial diagnoses.

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

Outline

- Programs that monitor and control hidden states.
- Consistency-based Diagnosis
  - Encoding diagnoses compactly using kernels.
  - Using conflicts to divide and conquer.

Conflicts Explain How to Remove Symptoms



Symptom:

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

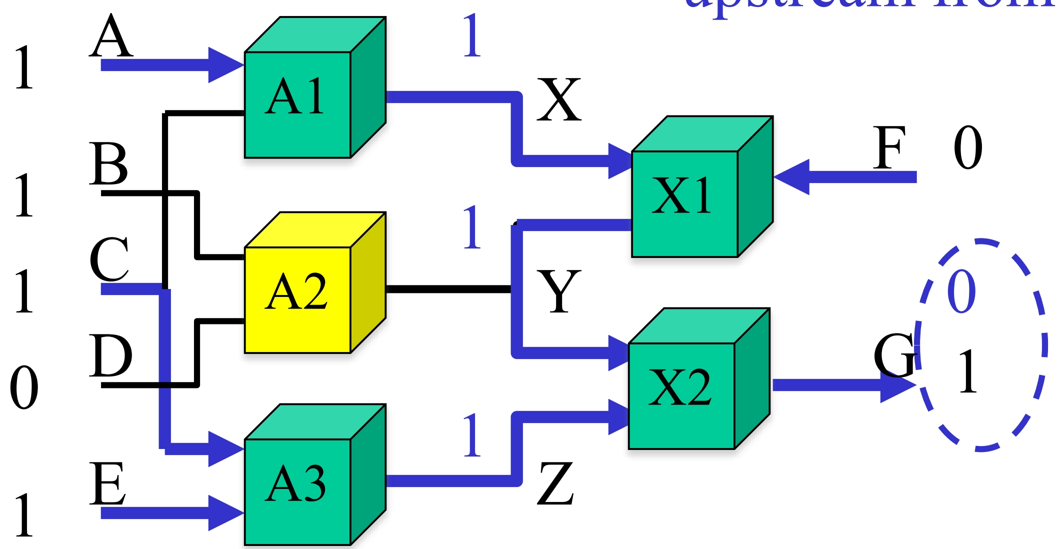
Conflict 1: {A1=G, A2=G, X1=G} is inconsistent.

→ 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.



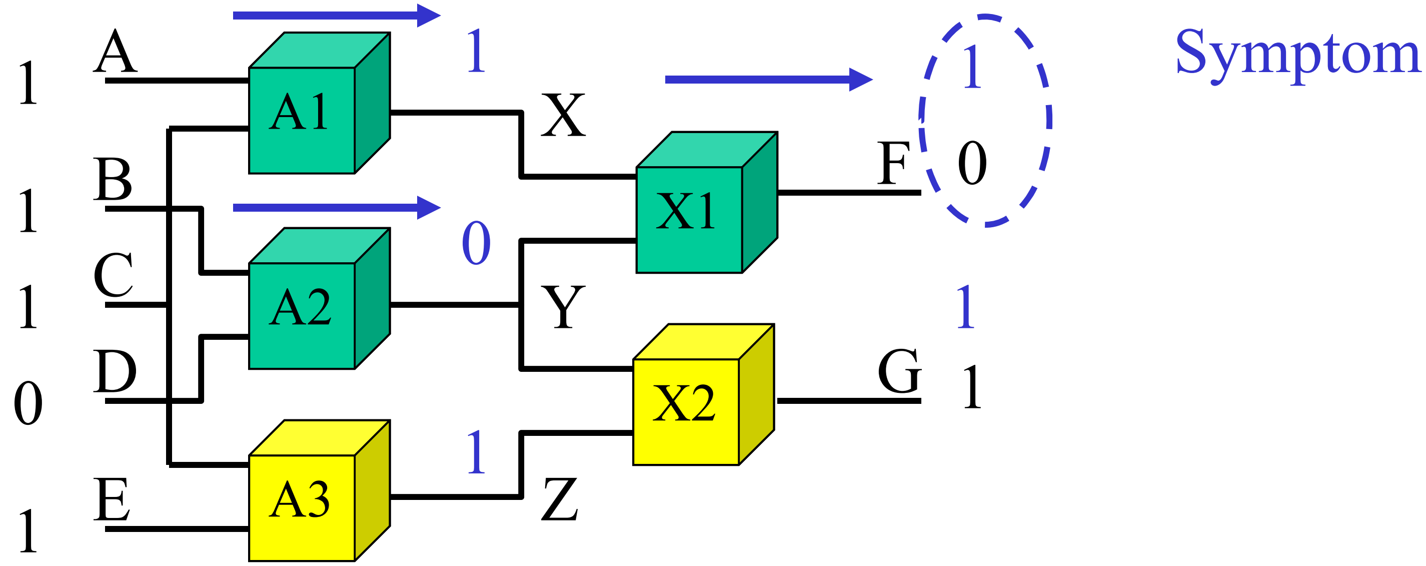
Symptom

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:

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

Diagnosis 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]

Summary: Mode Estimation

Given:

- Mode, State, Observation Variables: X, Y, and $O \subseteq Y$
- Obs = an assignment to O
- Model: $\Phi(X,Y)$ = components + structure

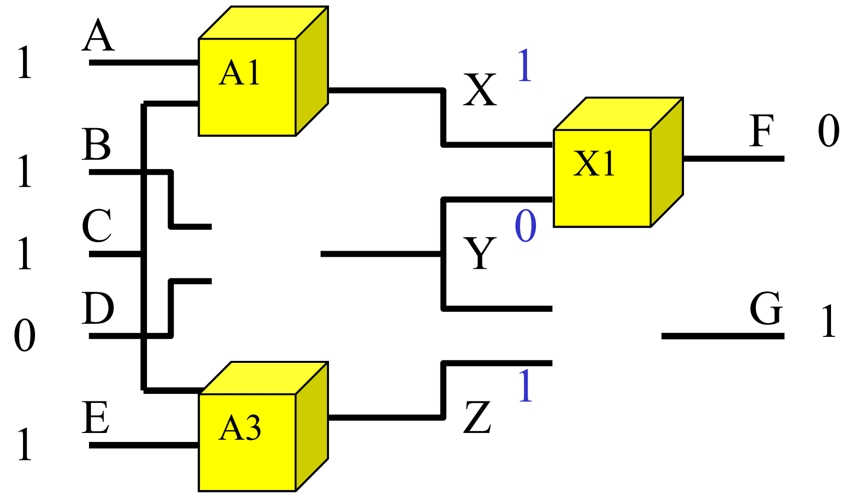
And(i):

G(i):

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

U(i): *No Constraint*

• All behaviors are associated with modes.
• All components have "unknown Mode" U, whose assignment is never mentioned in any constraint.



Return: All mode estimates

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