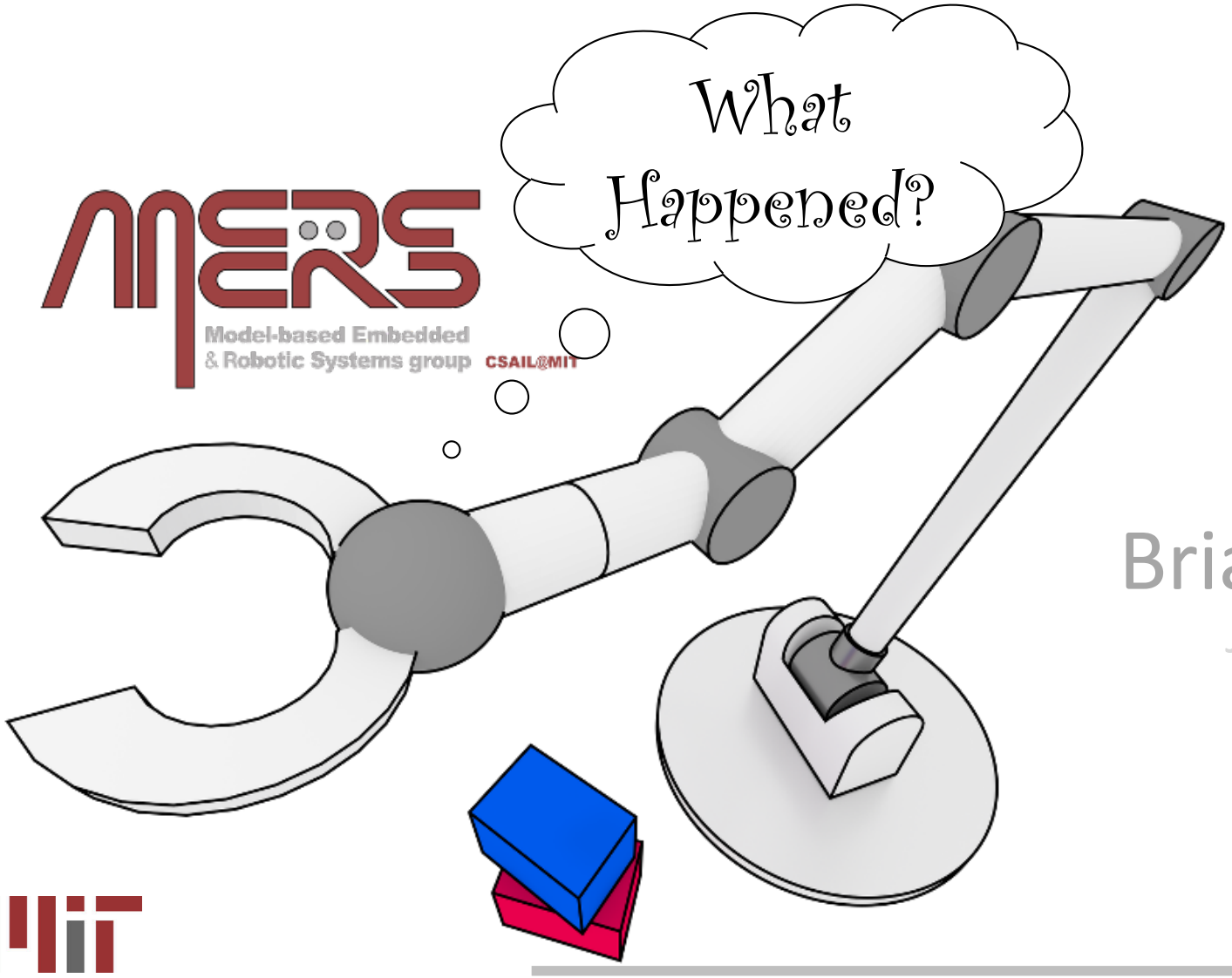
Self-Monitoring, Self-Diagnosing Systems



MERS
Model-based Embedded & Robotic Systems group CSAIL@MIT

Brian Williams

June 12th, 2017

Slide Contributions:
Steve Levine
David Wang
Brian Williams

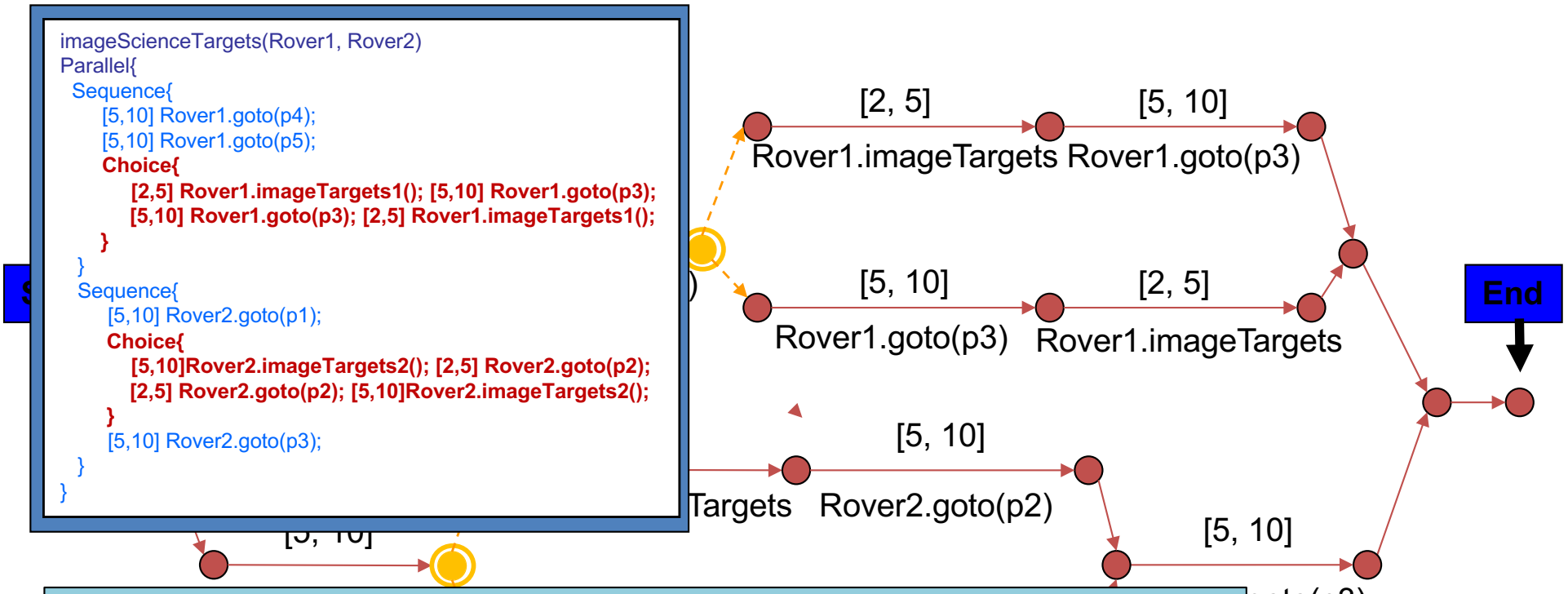
Some Related Papers

- 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.
- B. C. Williams and P. P. Nayak, ”A Model-based Approach to Reactive Self-Configuring Systems,” AAAI, 1996.
- O. Martin, B. Williams and M. Ingham, “Diagnosis as Approximate Belief State Enumeration for Probabilistic Concurrent Constraint Automata,” *AAAI,* 2005.
- M. Hofbaur and B. Williams, “Hybrid Estimation of Complex Systems,” IEEE Trans SMC, v. 34, Oct., 2004.
- L. Blackmore, S. Funiak and B. Williams, “A Combined Stochastic and Greedy Hybrid Estimation Capability for Concurrent Hybrid Models with Autonomous Mode Transitions,” J. Robotics and Autonomous Systems, V. 56, 2007.
- P. Yu and B. Williams, “Continuously Relaxing Over-constrained Conditional Temporal Problems through Generalized Conflict Learning and Resolution,” *IJCAI,* Beijing, China, August 2013.
- P. Yu and C. Fang and B. Williams, “Resolving Uncontrollable Conditional Temporal Problems using Continuous Relaxations,” *ICAPS*, Portsmouth, 2014.
- Kirk: Kim, P., Williams, B., Abramson, M., “Executing Reactive, Model-based Programs through Graph-based Temporal Planning.“ IJCAI, 2001.
- Pike: Levine, S., Williams, B., “Concurrent Plan Recognition and Execution for Human-Robot Teams.” ICAPS, 2014.
- ITC: Shu, I., Effinger, R., Williams, B., ”Enabling Fast Flexible Planning through Incremental Temporal Reasoning with Conflict Extraction.” AAAI, 2005.
- DBT: Effinger, R., Williams, B., ” Extending Dynamic Backtracking to Solve Weighted Conditional CSPs.” AAAI, 2006.

Recall: Dynamic Languages Leave Choices Open

RMPL

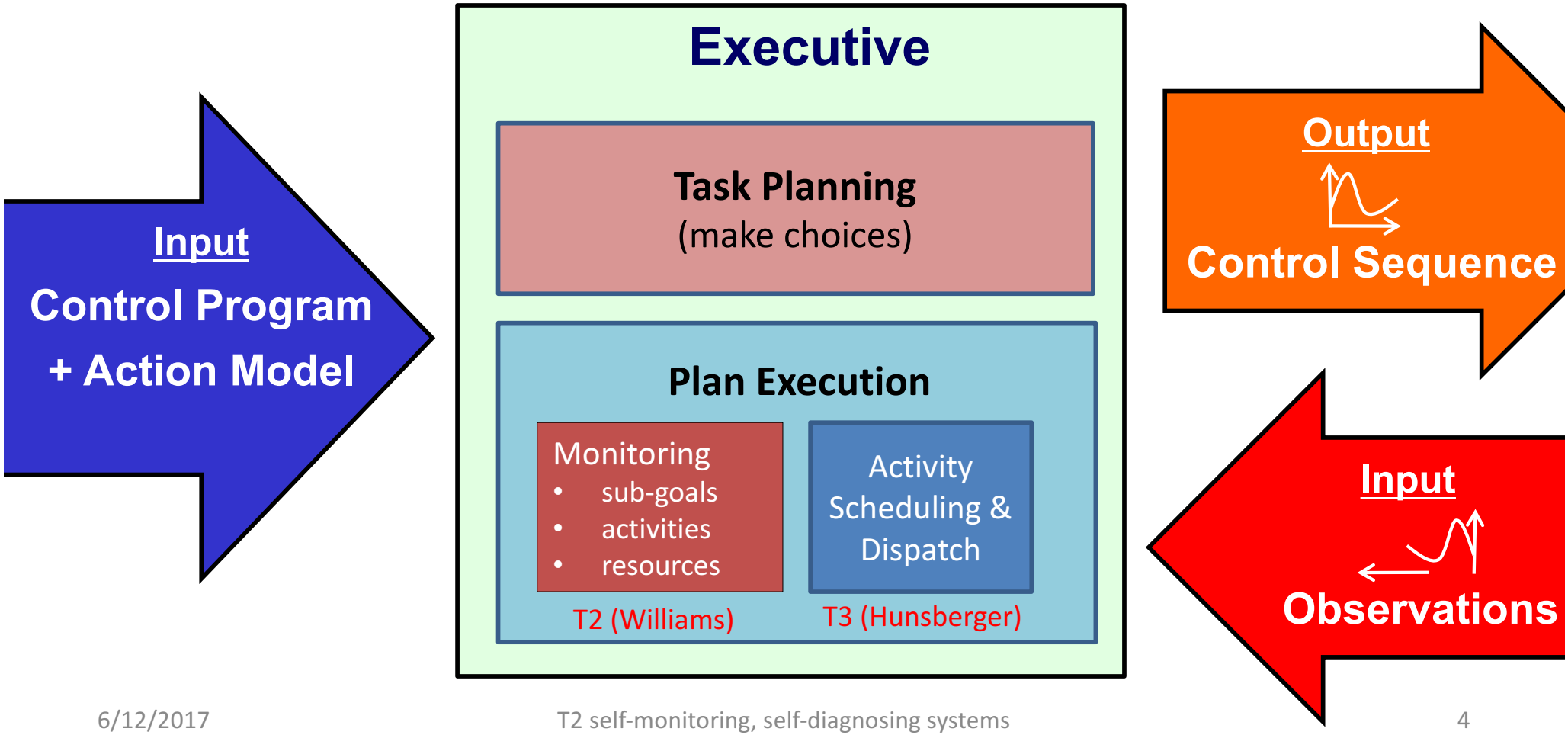
```
imageScienceTargets(Rover1, Rover2)
Parallel{
  Sequence{
     [5,10] Rover1.goto(p4);
     [5,10] Rover1.goto(p5);
     Choice{
        [2,5] Rover1.imageTargets1(); [5,10] Rover1.goto(p3);
        [5,10] Rover1.goto(p3); [2,5] Rover1.imageTargets1();
     }
  }
  Sequence{
     [5,10] Rover2.goto(p1);
     Choice{
        [5,10]Rover2.imageTargets2(); [2,5] Rover2.goto(p2);
        [2,5] Rover2.goto(p2); [5,10]Rover2.imageTargets2();
     }
     [5,10] Rover2.goto(p3);
  }
}
```



Program A ::=
- prim_action(args) |
- [lb, ub] A |
- Sequence {A1; A2; … } |
- Parallel {A1; A2; … } |
- Choose { [with reward R1] A1; [with reward] R2 A2; . . .}

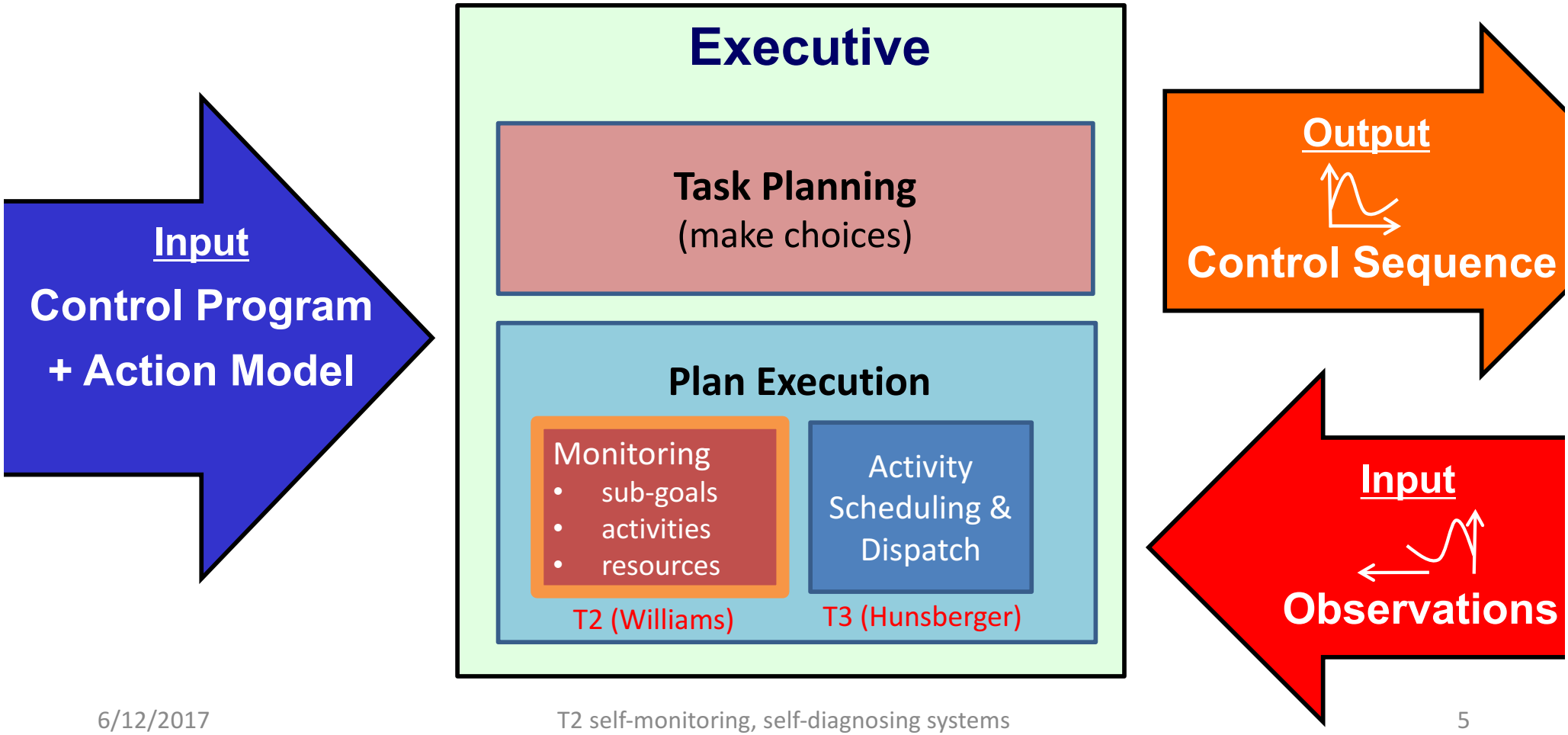
Recall: Program Executives Monitor Goals and Make Online Choices

- **Plan**: Make optimal, consistent program choices.
- **Execute** : Dispatch and monitor resulting plan.

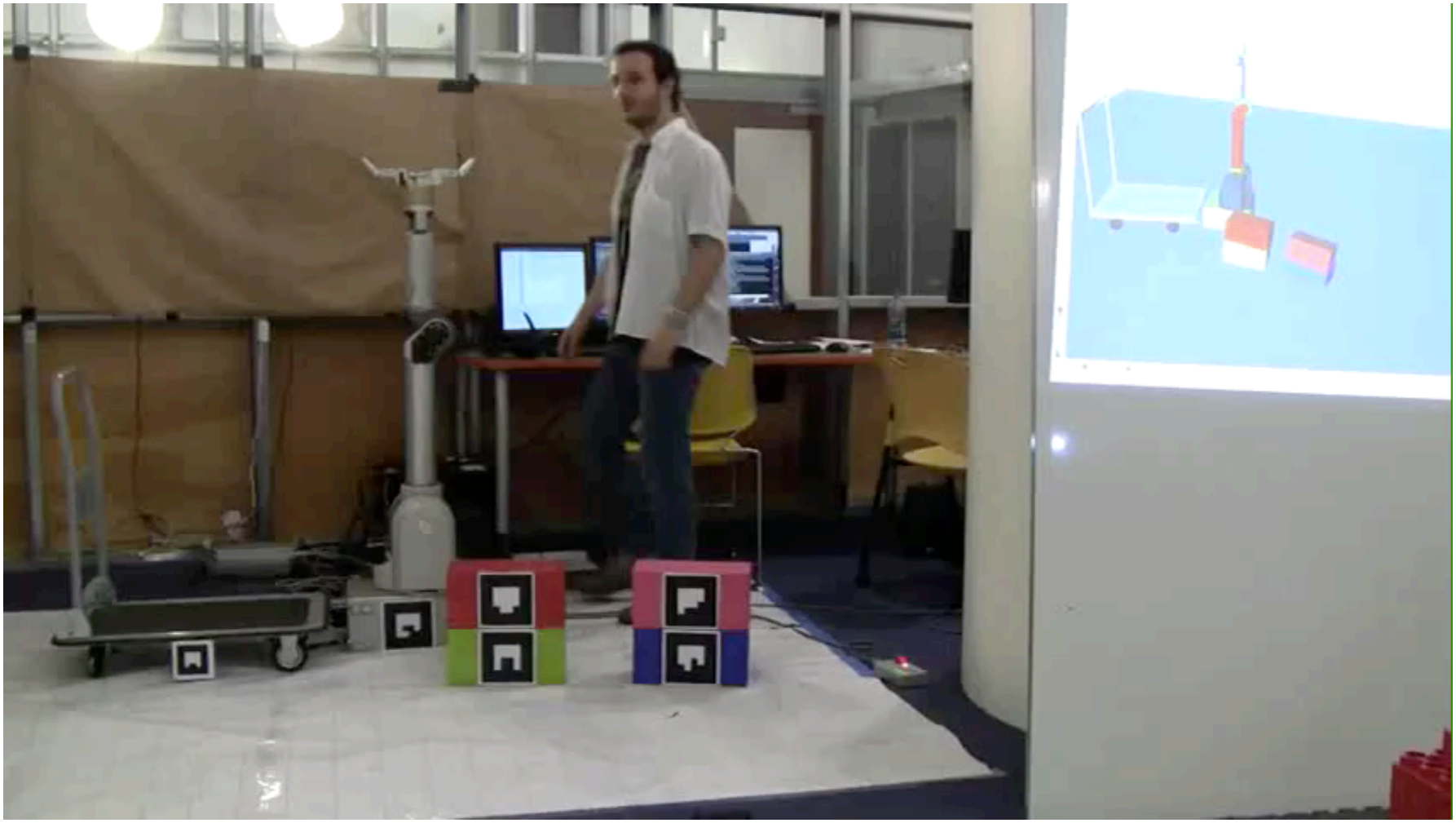


Recall: A "Suspicious" Executive Closes the Loop on Goals

- **Monitors**: detects as soon as sub-goals fail, actions break.
- **Adapts**: fixes the problem.



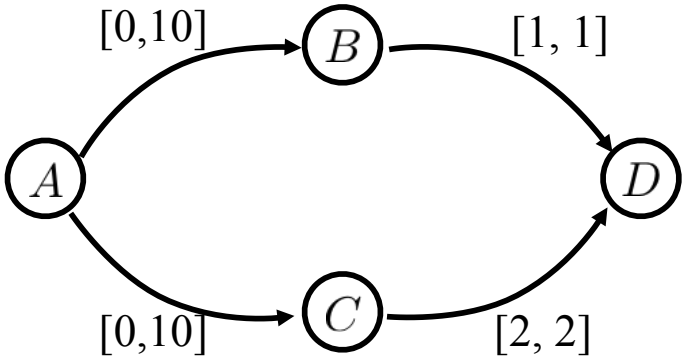
Recall: Self-Monitoring Systems



Detects failing actions and plans, due to:

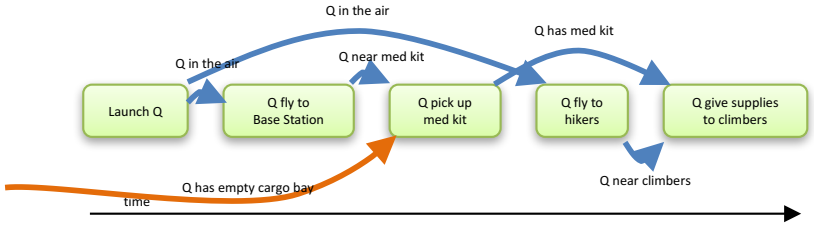
- Anomalies in relevant state (sub-goals).
- Faulty hardware.

We can create different programming models by combining different methods



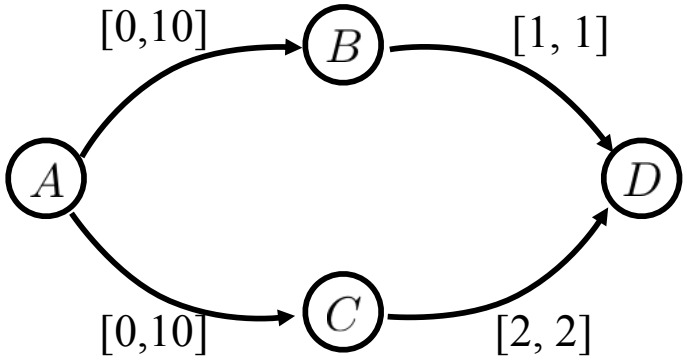
Flexible Time

+



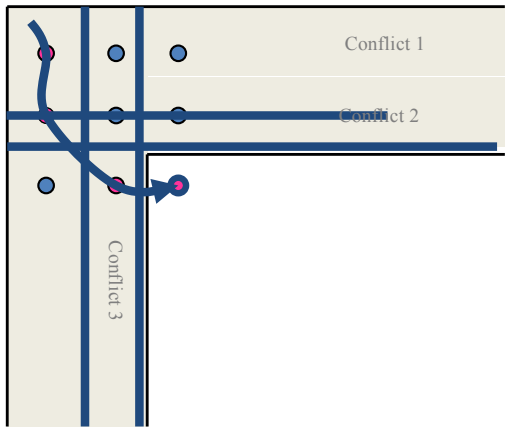
Execution monitoring

and



Flexible Time (w Uncertainty)

+



Diagnosis and Reconfiguration

Discrete Choice and Search

Outline

- Programs that Monitor State
- Programs that Self-Diagnose (opt)

Outline

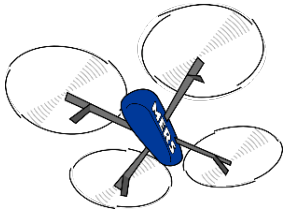
- Programs that Monitor State
  - Sub-goal monitoring
  - Model-based diagnosis and mode estimation
  - Programs that Self-Diagnose (opt)

Volcano eruption!

```
method run() {
    sequence {
        uav.launch();
        uav.fly(base_station);
        uav.pick_up(med_kit);
        uav.fly(hikers);
        uav.drop_off(med_kit);
    }
}
```

Actions have specified preconditions & effects

Base Station



Program (Plan) is a sequence of **actions** in **RMPL**

Volcano Rescue in RMPL

```
class Main{
  QuadCopter uav;
  Location base-station;
  People climbers;
  MovableObject med-kit;
  ...

 method run() {
    sequence{
     uav.launch();
     uav.fly(base_station);
     uav.pick_up(med_kit);
     uav.fly(climbers);
     uav.drop_off(med_kit)}}
```

Program A ::=
prim_action(args) |
[lb, ub] A |
Sequence {A1; A2; ... } |
Parallel {A1; A2; ... }

QuadCopter Action Model in RMPL

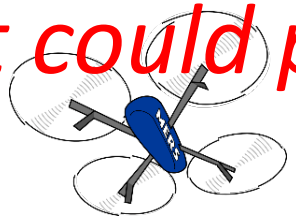
```
class QuadCopter {
  Roadmap location;
  Boolean flying;
  Boolean loaded;

  primitive method Launch()
    flying == no => flying == yes;

  primitive method land()
    flying == yes => flying == no;

  primitive method pickup(MoveableObject object)
    ((flying == yes) && (loaded == no) && (object.location == location))
     => loaded == yes && (object.on == self);

  primitive method drop_off(MoveableObject object)
    ((flying == yes) && (loaded == yes) && object.on == self)
     => ((loaded == no) &&(object.location == location) && object.on == none);

  #MOTION_PRIMITIVES(location, fly, flying==yes)
}
```

(PDDL overs a rich action model)

Volcano

Base Station

What could possibly go wrong??



Launch Q

Q fly to Base Station

Q pick up med kit

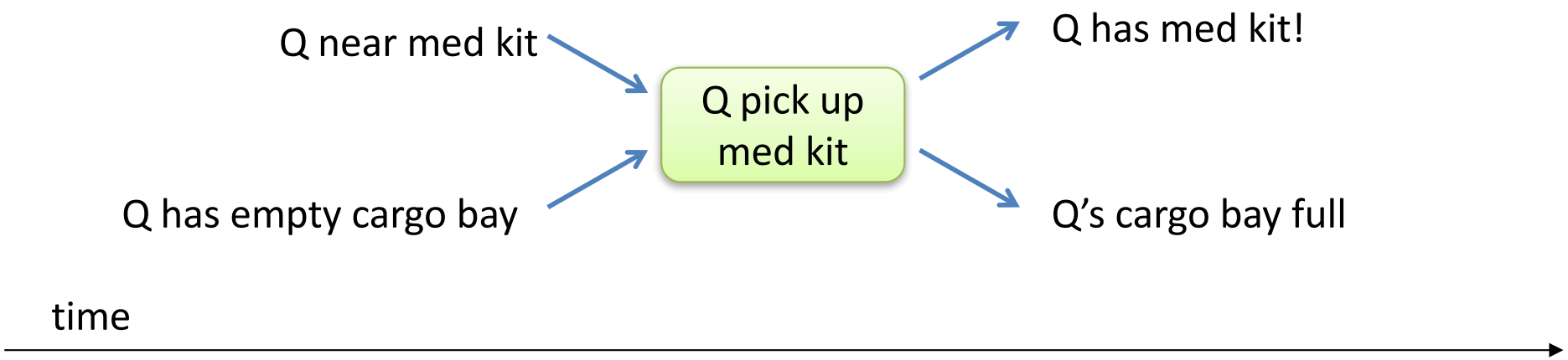
Q fly to hikers

Q give med kit to climbers

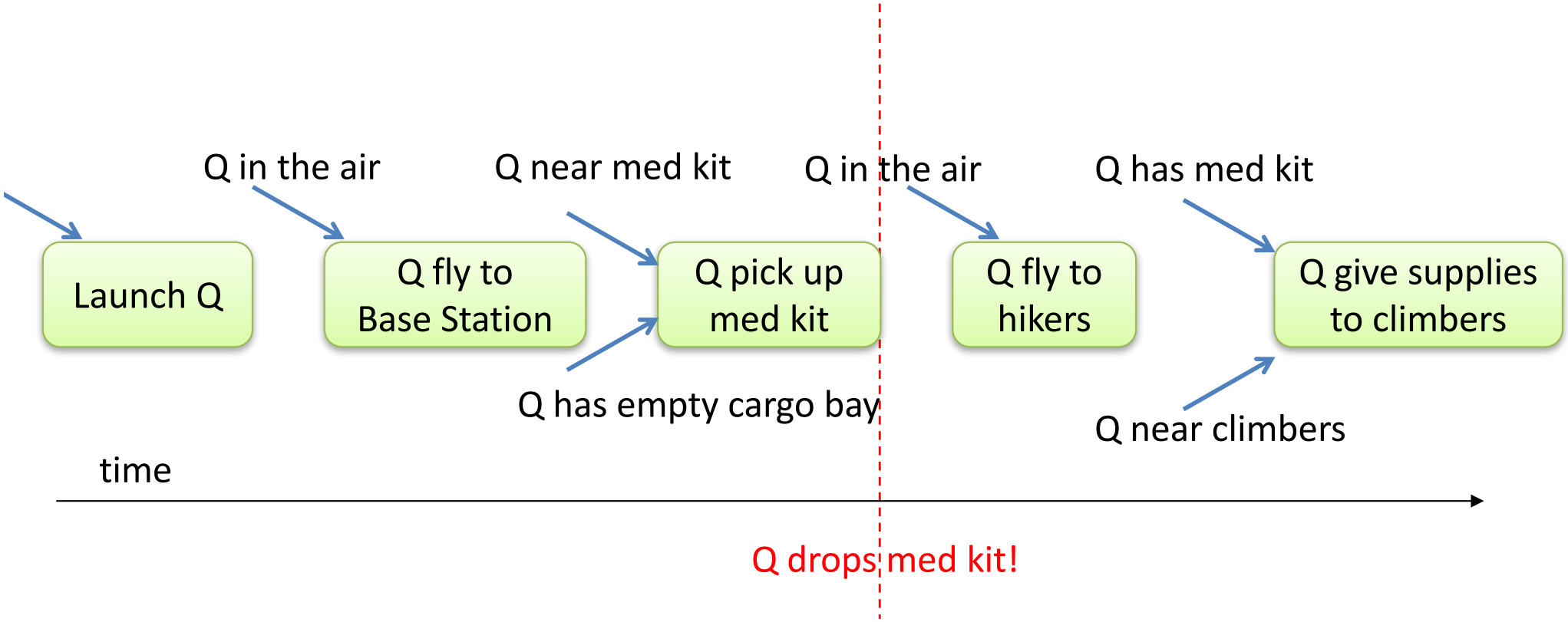
Can set expectations from action preconditions & effects

Can set expectations from action preconditions & effects

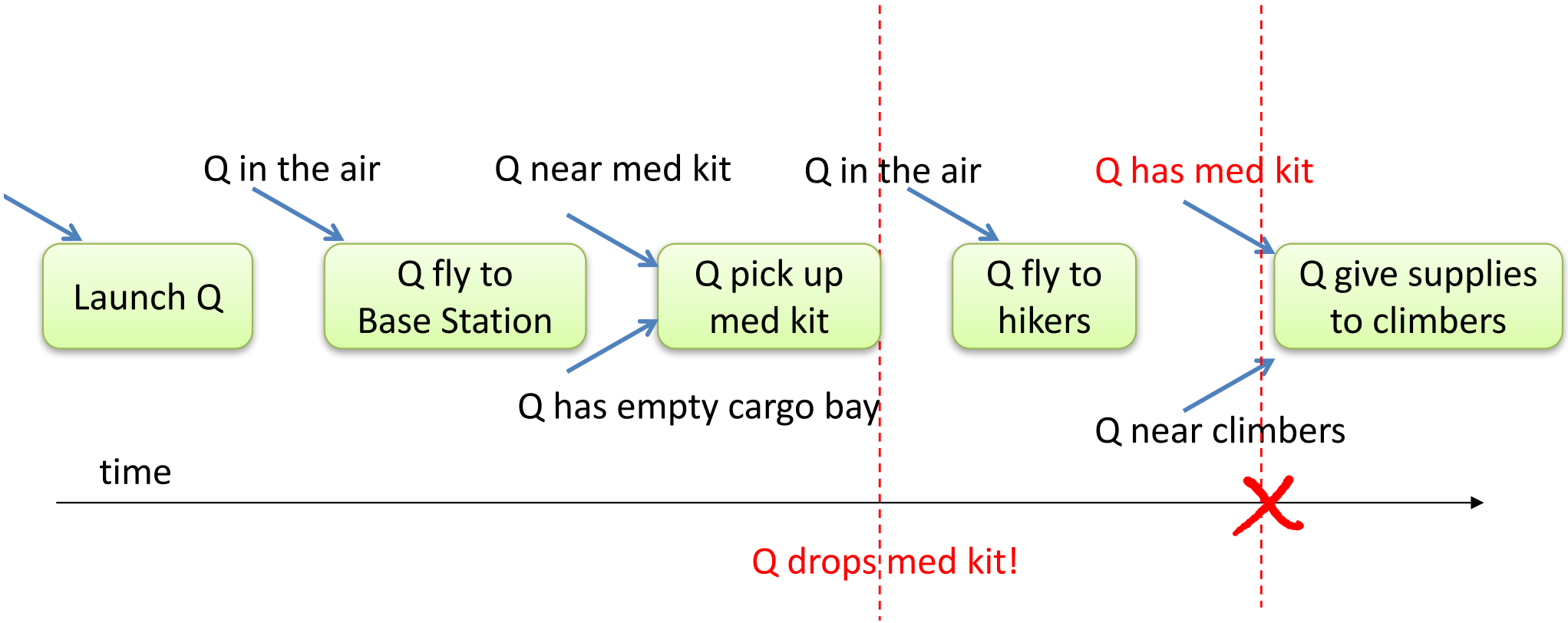
Can set expectations from action preconditions & effects



Preconditions tell executive what's *relevant*



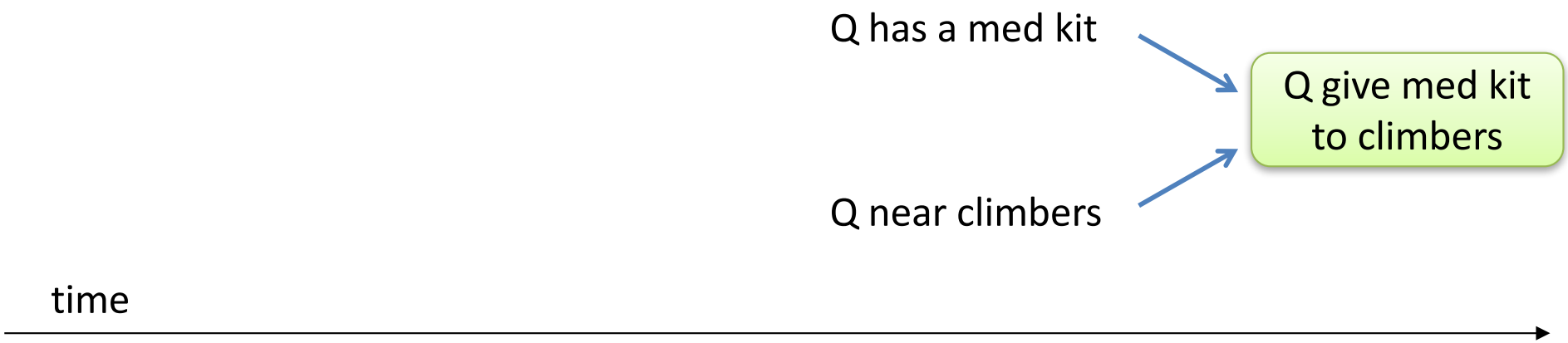
Preconditions tell executive what's *relevant*



Action monitoring checks preconditions just before action executed.

(e.g., RosPlan)

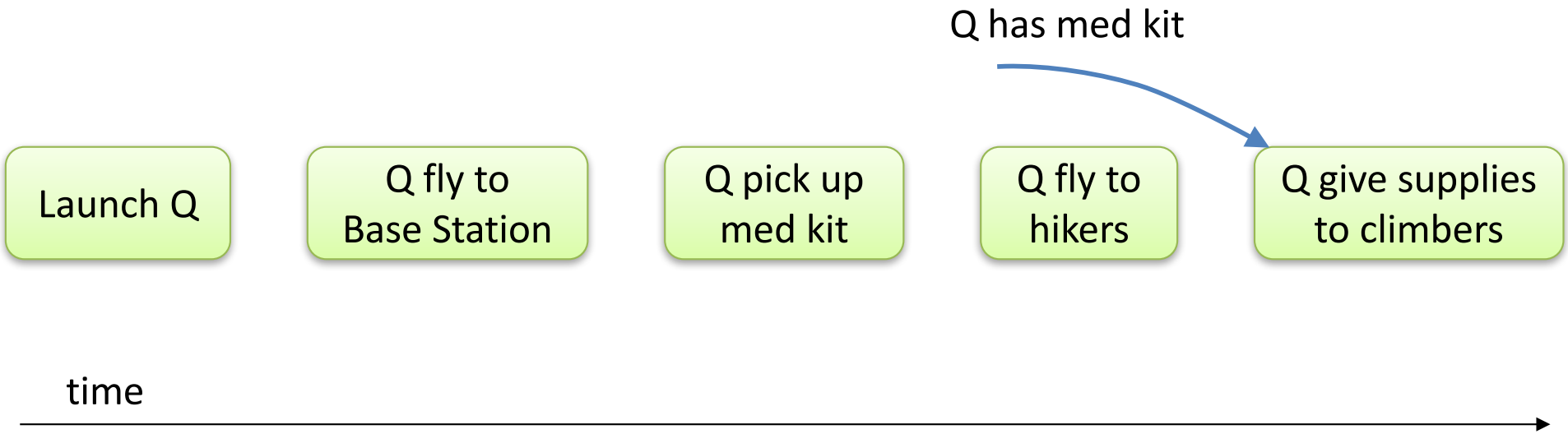
How do we anticipate failures?



Causal-link monitoring checks preconditions as soon as desired effect is first established.

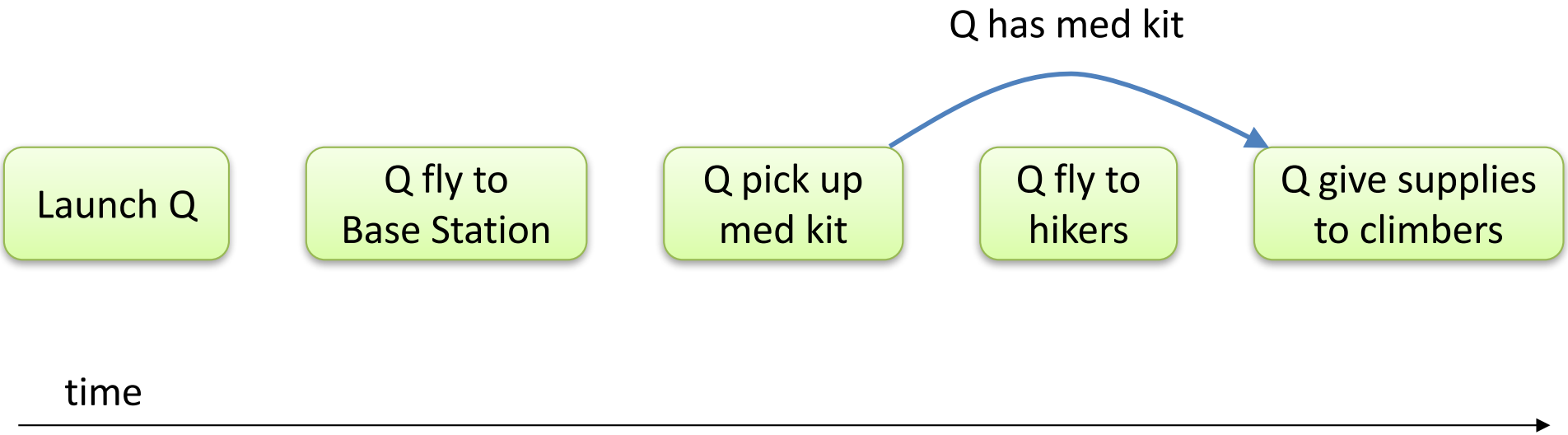
- But for how long should each precondition hold?

When are preconditions established?



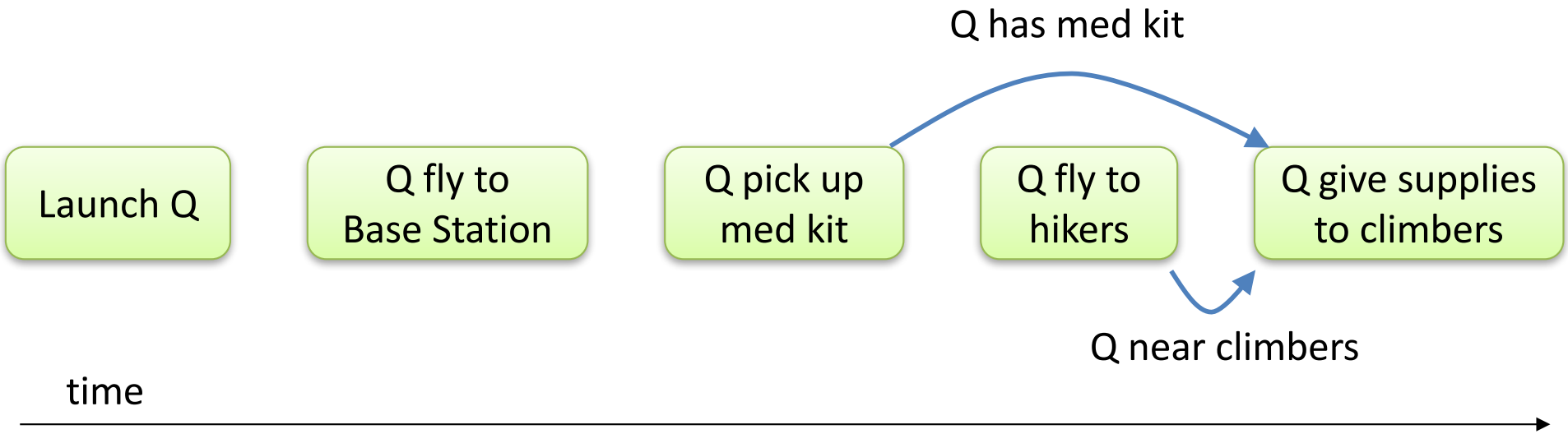
Causal-link monitoring checks preconditions as soon as desired effect is first established.

When are preconditions established?



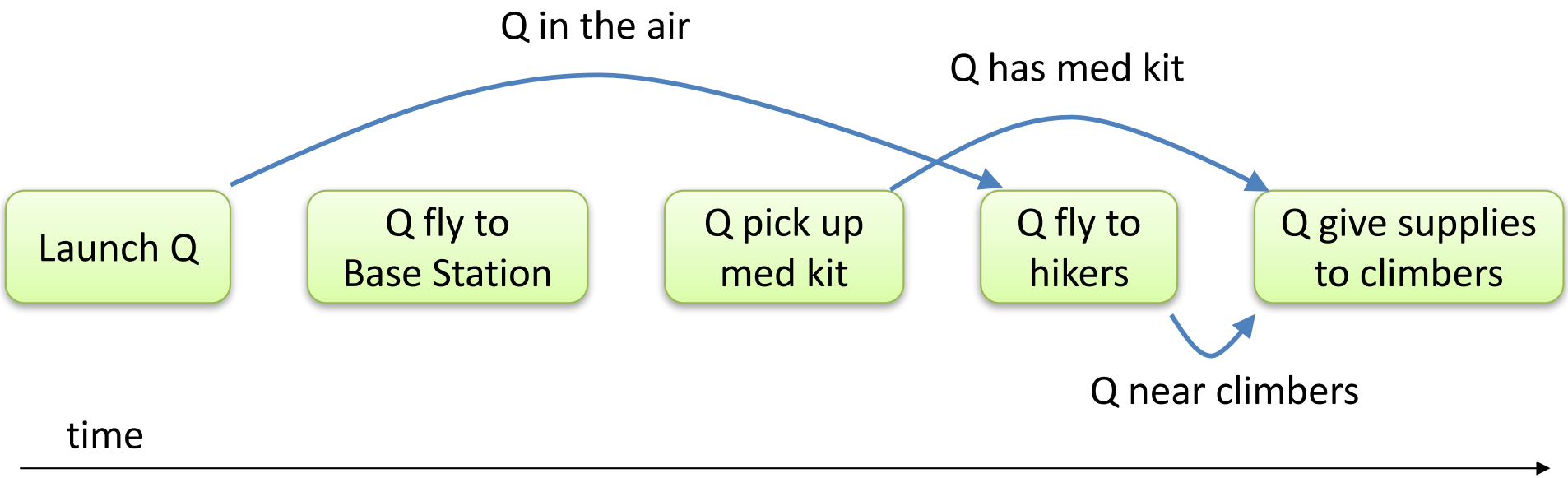
Causal-link monitoring checks preconditions as soon as desired effect is first established.

When are preconditions established?



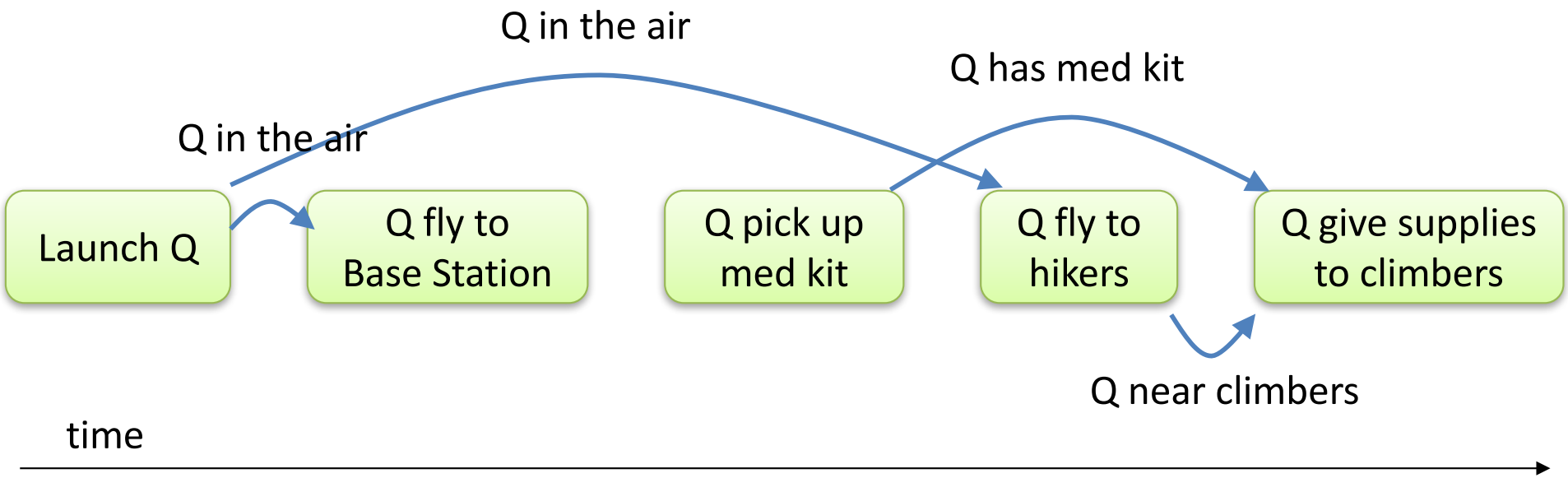
Causal-link monitoring checks preconditions as soon as desired effect is first established.

When are preconditions established?



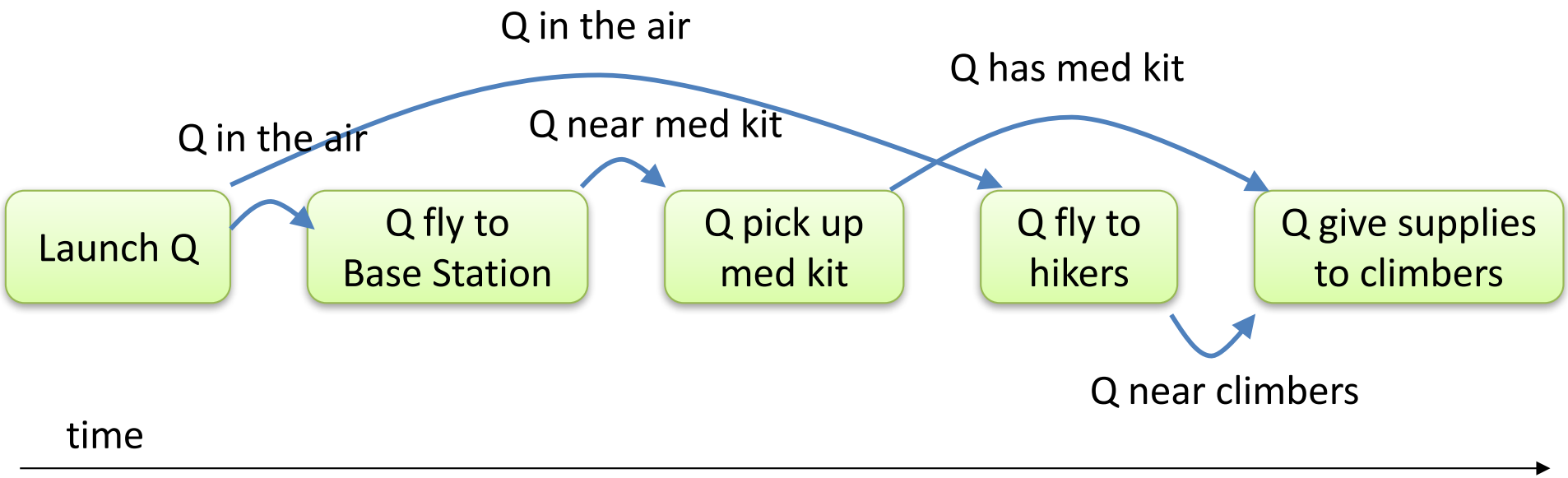
Causal-link monitoring checks preconditions as soon as desired effect is first established.

When are preconditions established?



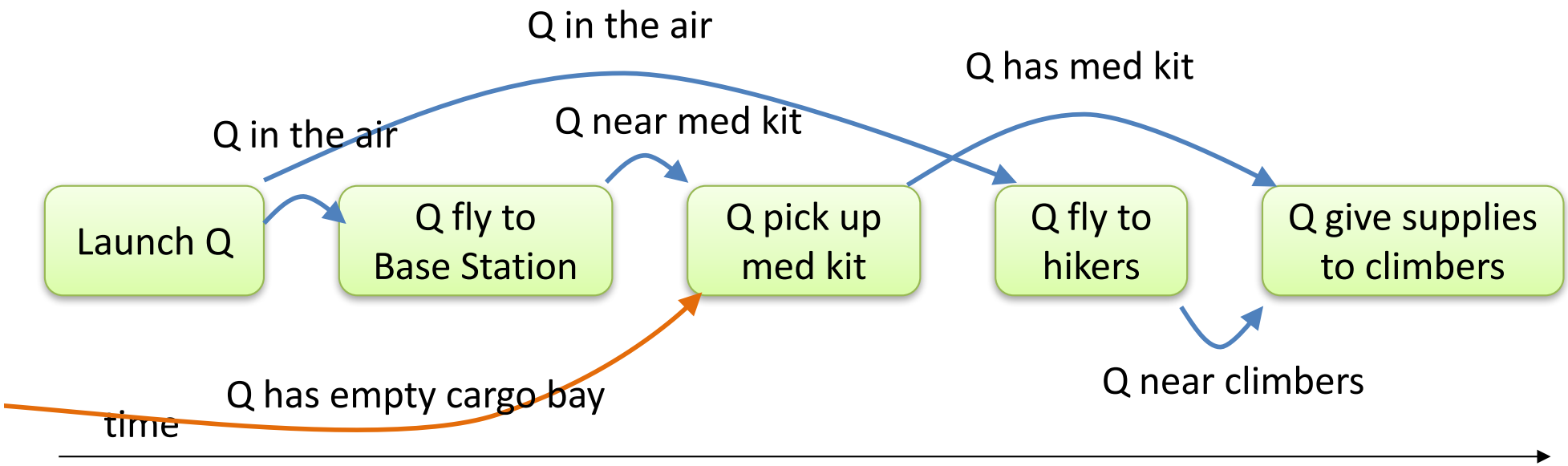
Causal-link monitoring checks preconditions as soon as desired effect is first established.

When are preconditions established?



Causal-link monitoring checks preconditions as soon as desired effect is first established.

When are preconditions established?



Causal-link monitoring checks preconditions as soon as desired effect is first established.

These are called “causal links”

Causal link: identifies how an action’s effect is consumed by a later action.

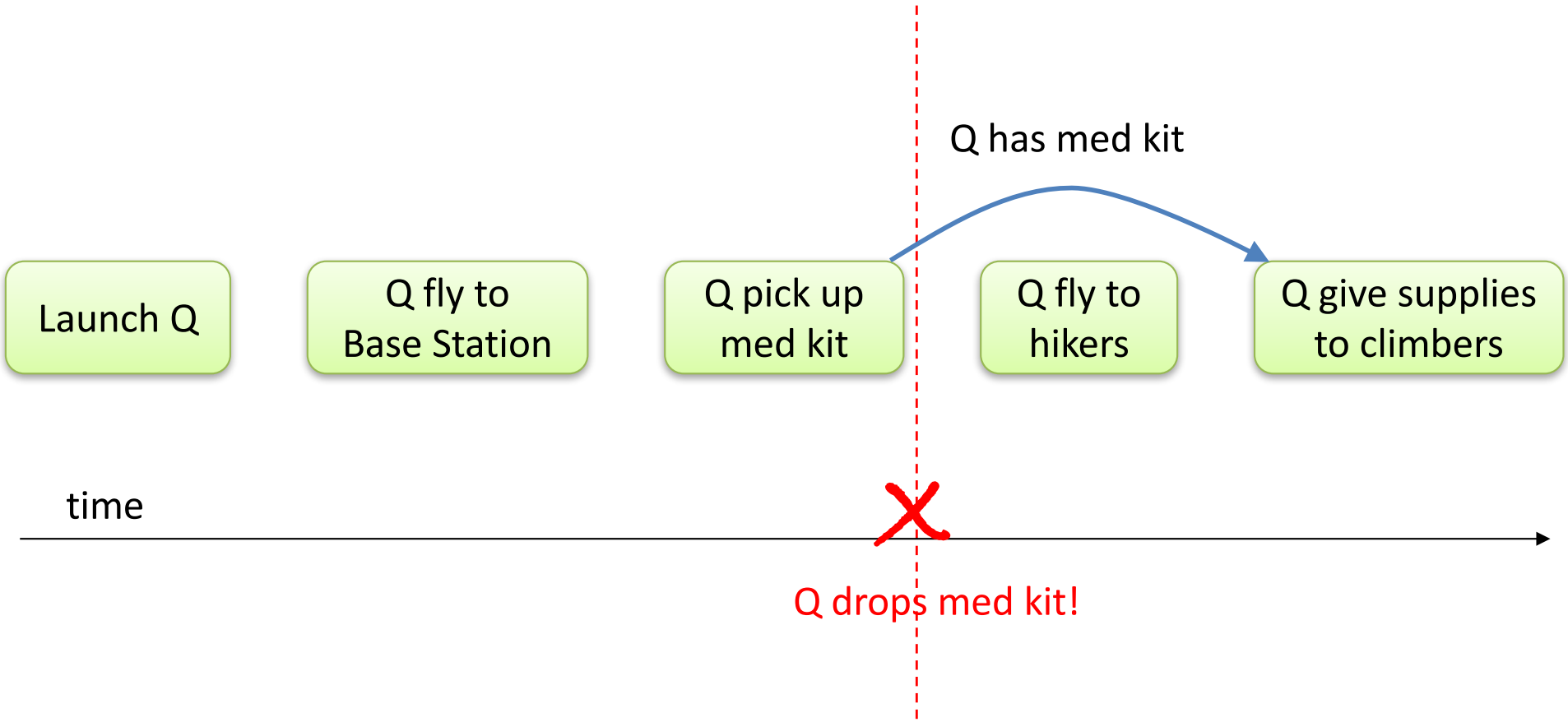
Causal Link

$$< a_{producer},\ p,\ a_{consumer} >$$

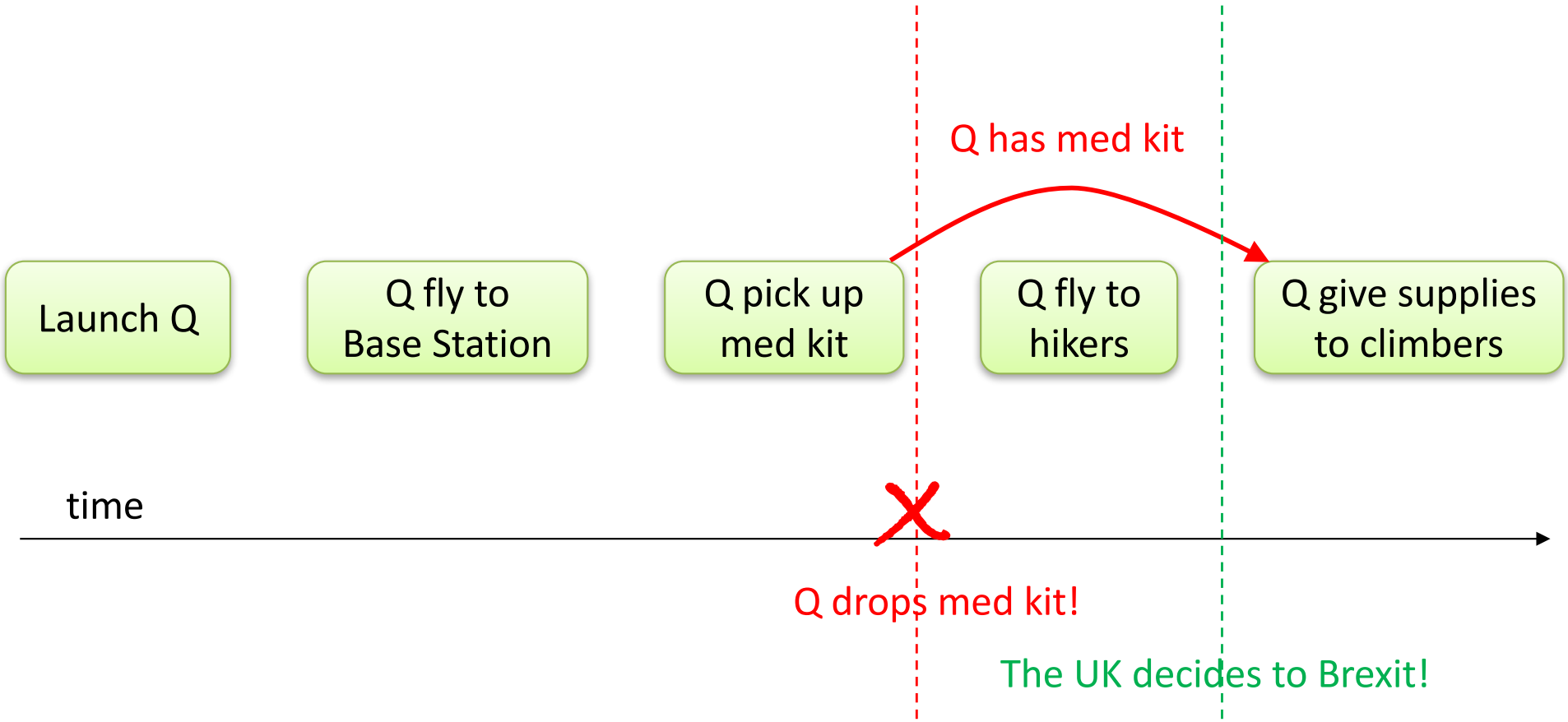
- Proposition p
- Producer: action with *p* as effect
- Consumer: action with *p* as precondition

- Producer must precede consumer
- No action interferes in between

Links tell executive what's *relevant* to plan success at any time.



Links tell executive what's *relevant* to plan success at any time.



How do we do perform causal link execution monitoring?

Offline:

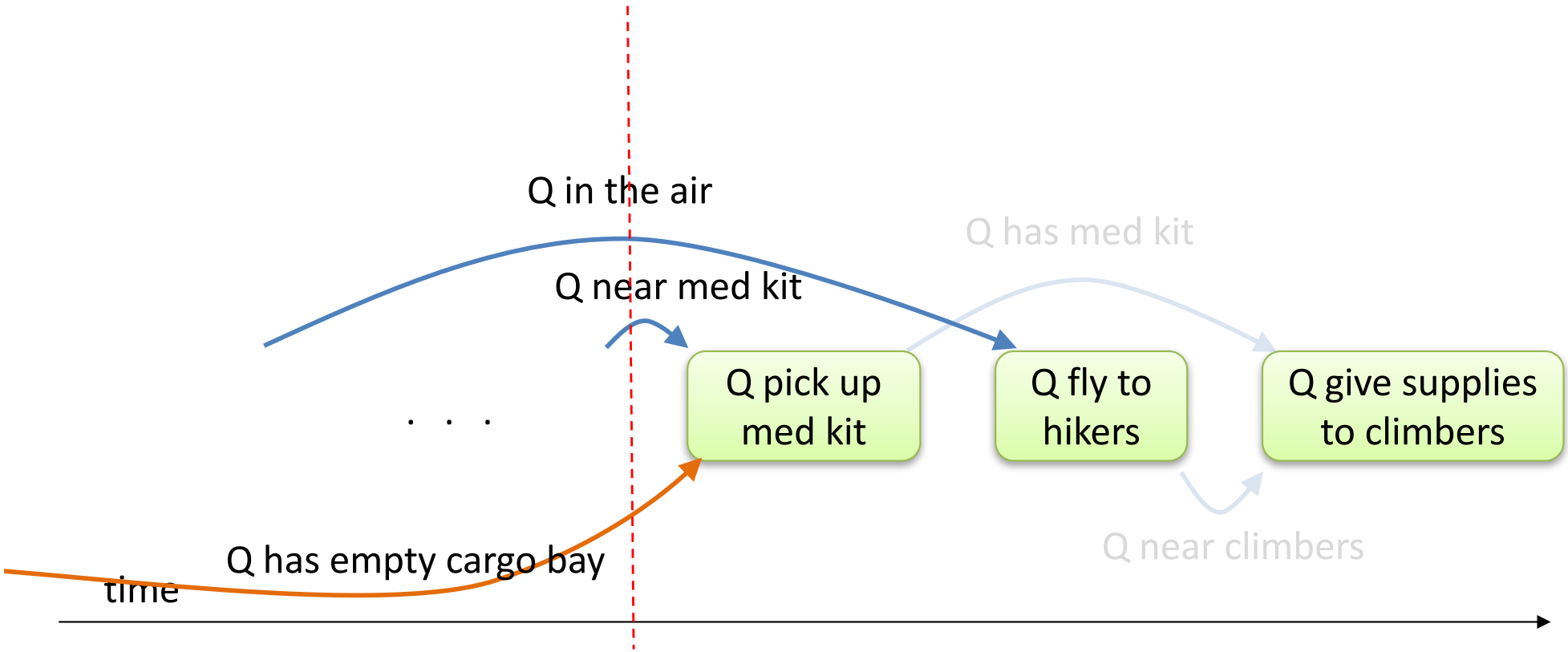
- ***Extract*** causal links from plan & action model

Online:

- Continuously ***monitor*** *"active"* causal links against sensor measurements

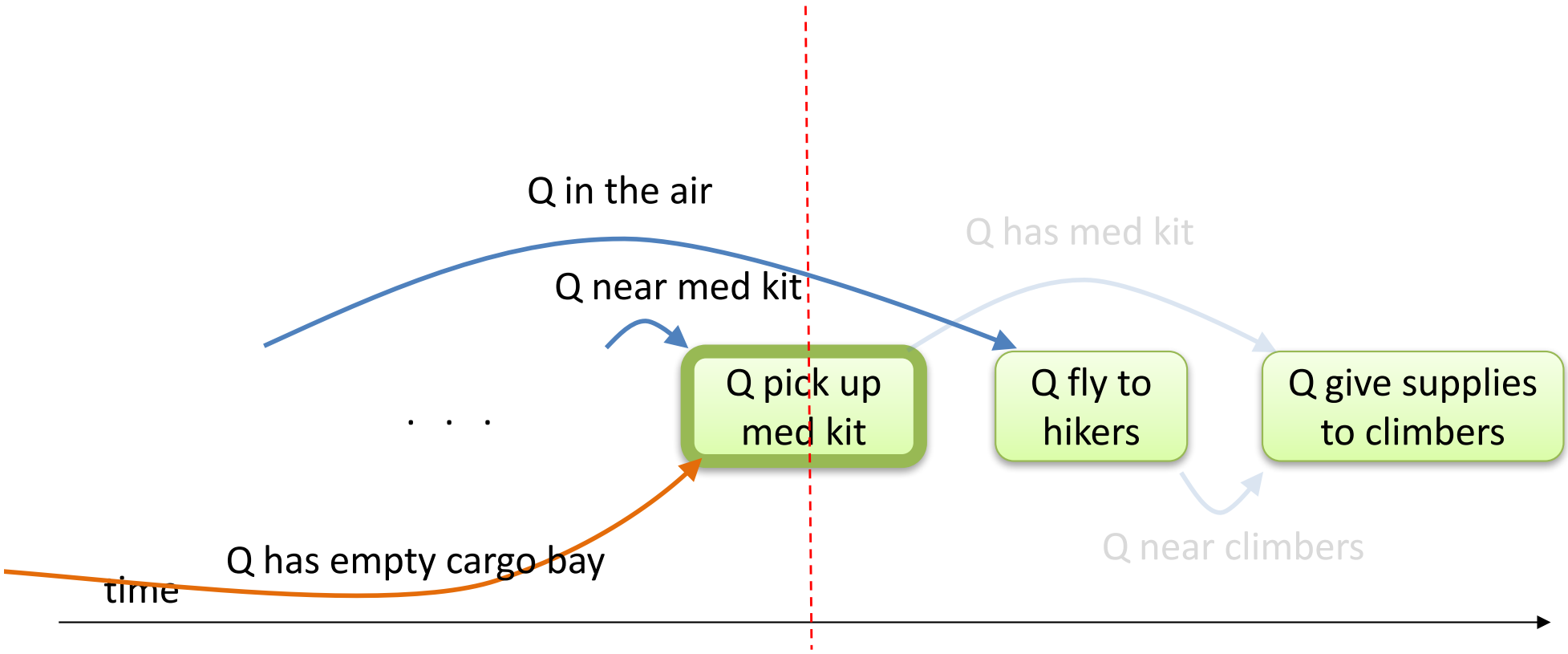
Active links:

< … , Q near med kit, Q pick up med kit>
< … , Q in the air, Q pick up med kit>
< … , Q has empty cargo bay, Q pick up med kit>



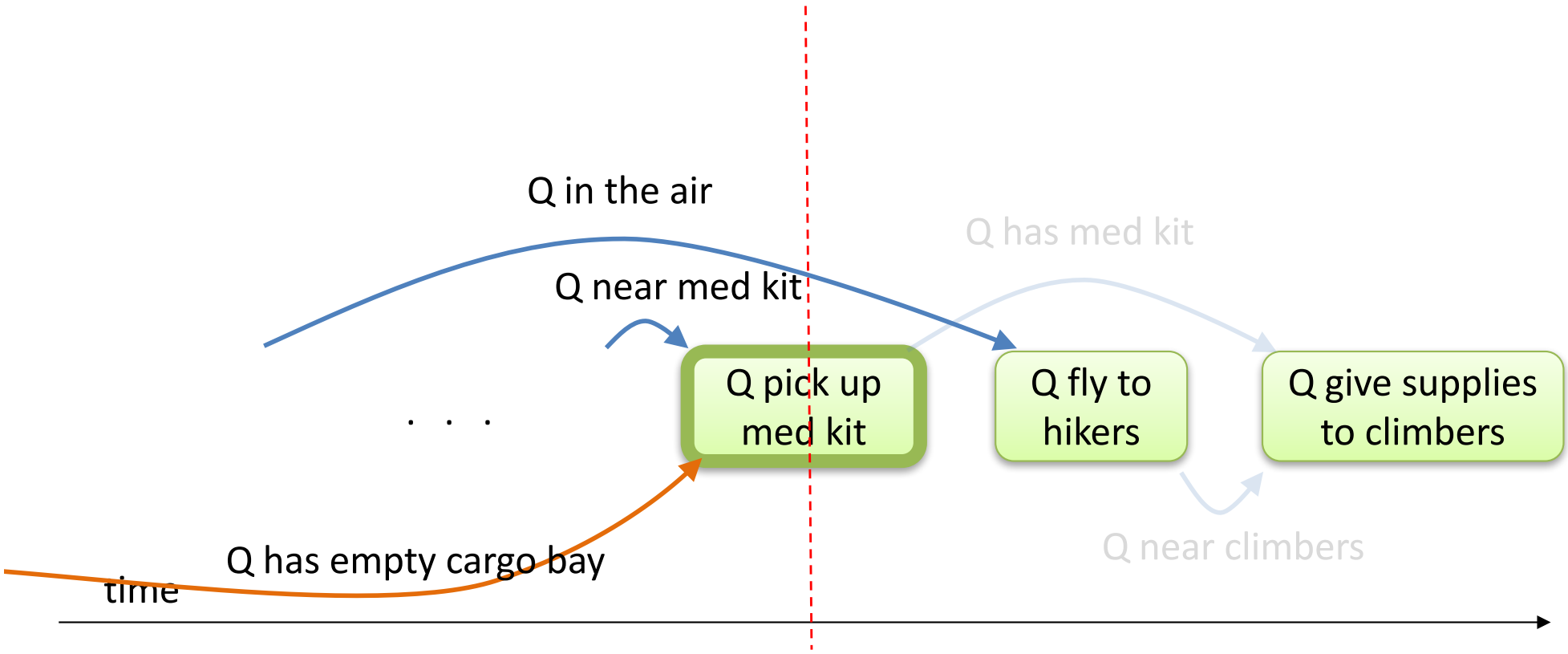
Active links:

< … , Q near med kit, Q pick up med kit>
< … , Q in the air, Q pick up med kit>
< … , Q has empty cargo bay, Q pick up med kit>



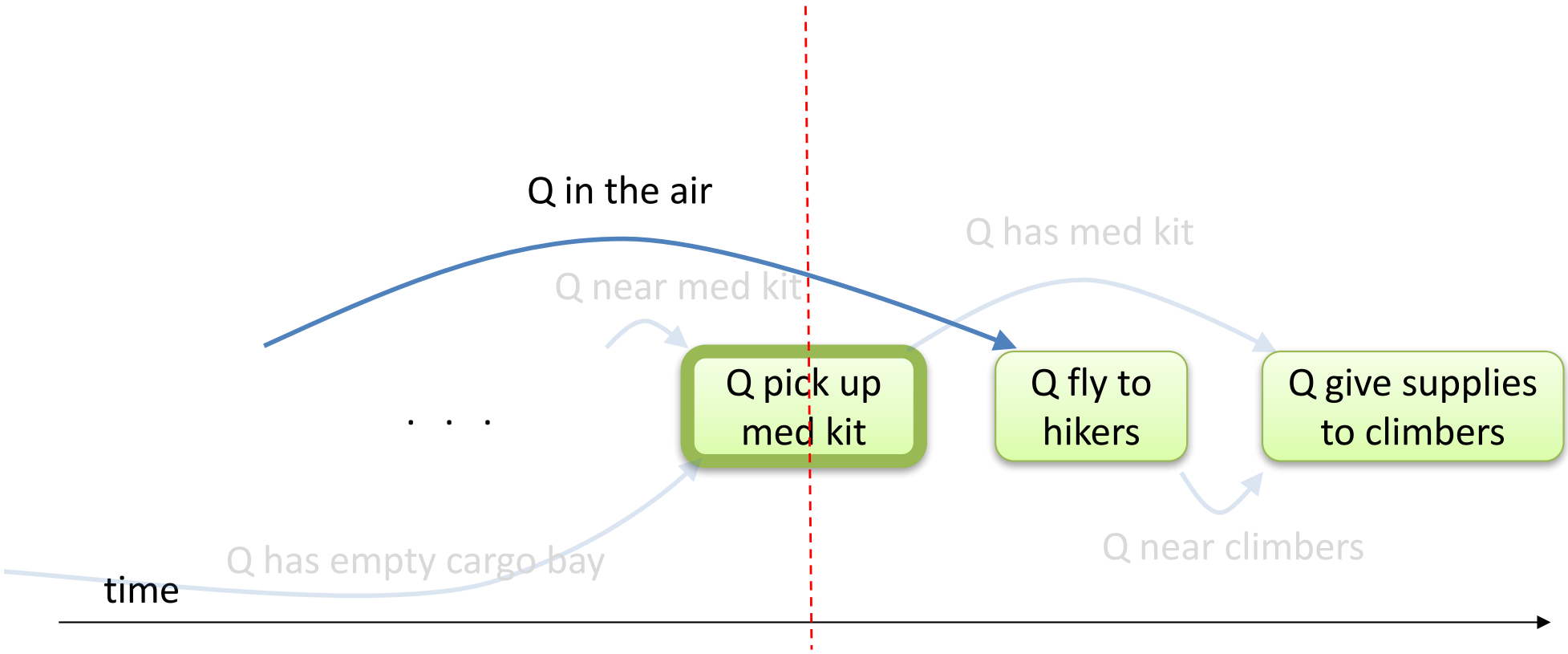
Active links:

< … , Q near med kit,	Q pick up med kit>
< … , Q in the air,	Q pick up med kit>
< … , Q has empty cargo bay,	Q pick up med kit>



Active links:

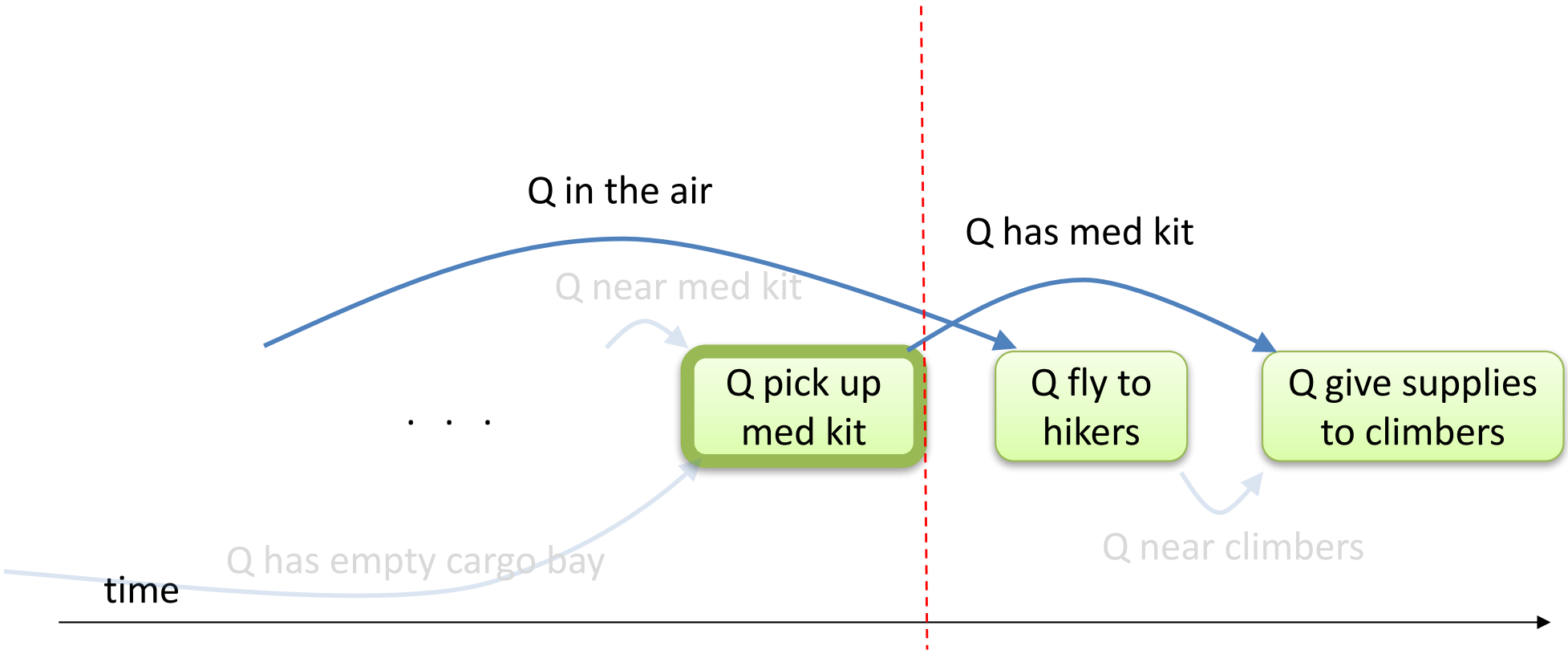
< … , Q in the air, Q pick up med kit>



Active links:

< … , Q in the air, Q pick up med kit>

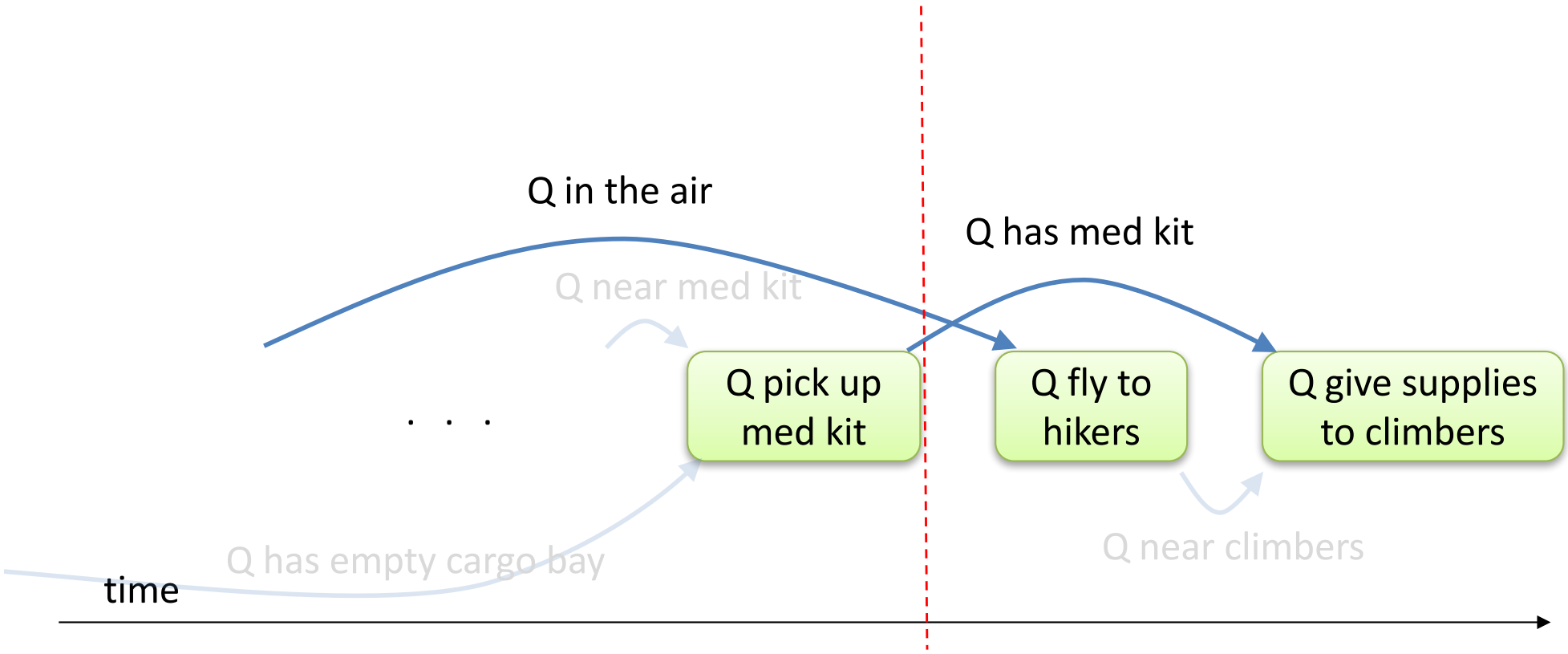
<Q pick up med kit, Q has med kit, Q give supplies>



Active links:

< … , Q in the air, Q pick up med kit>

<Q pick up med kit, Q has med kit, Q give supplies>



Active links:

< … , Q in the air, Q pick up med kit>

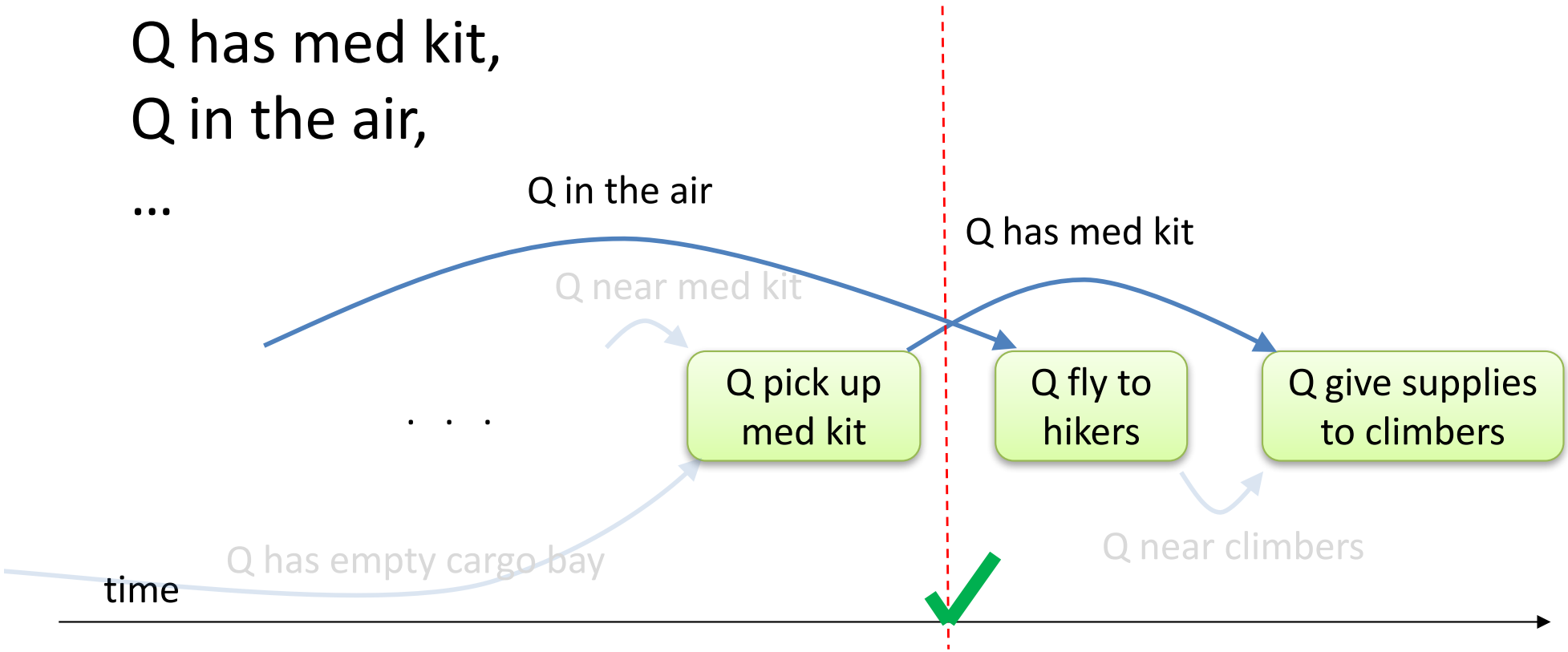
<Q pick up med kit, Q has med kit, Q give supplies>

S:

Q has med kit,

Q in the air,

…



Active links:

< … , Q in the air, Q pick up med kit>

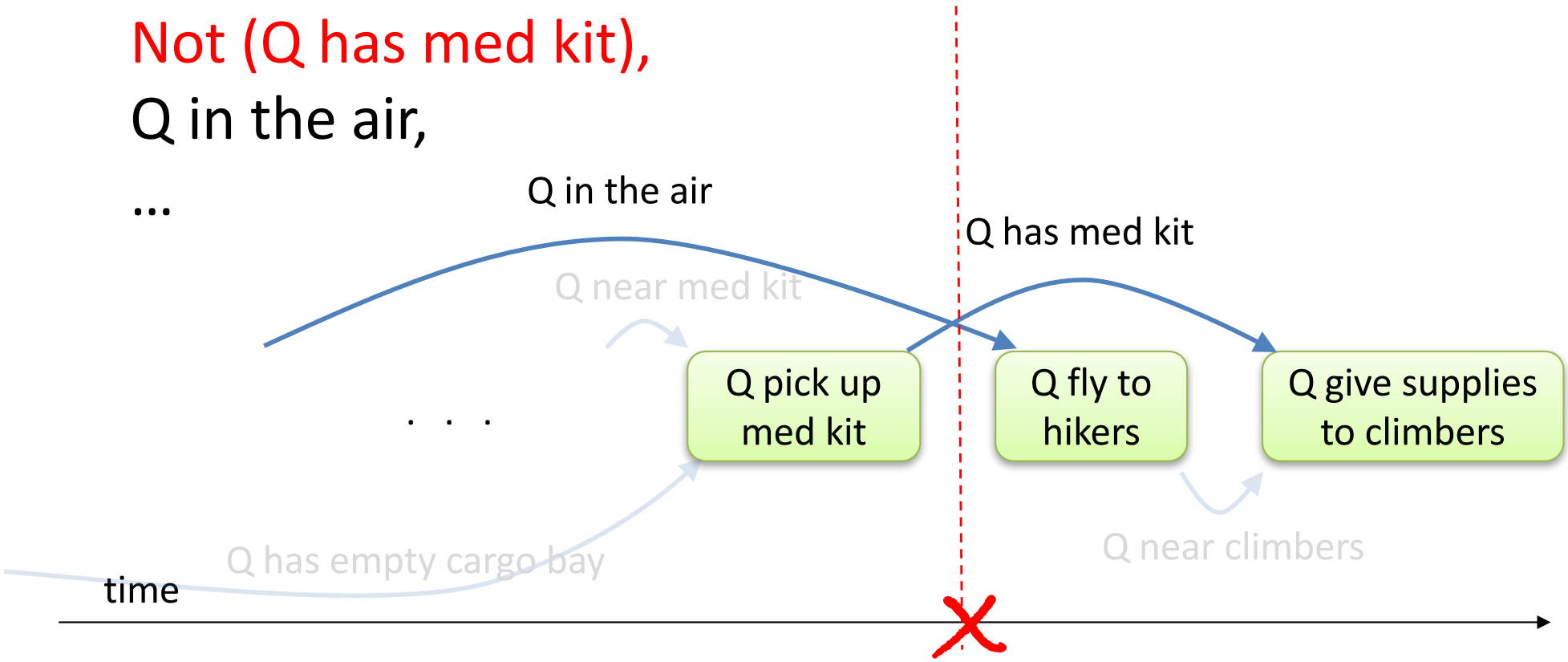
<Q pick up med kit, Q has med kit, Q give supplies>

S:

Not (Q has med kit),

Q in the air,

…



Key takeaways

- Need to monitor plan sub-goals during execution, to anticipate failures.

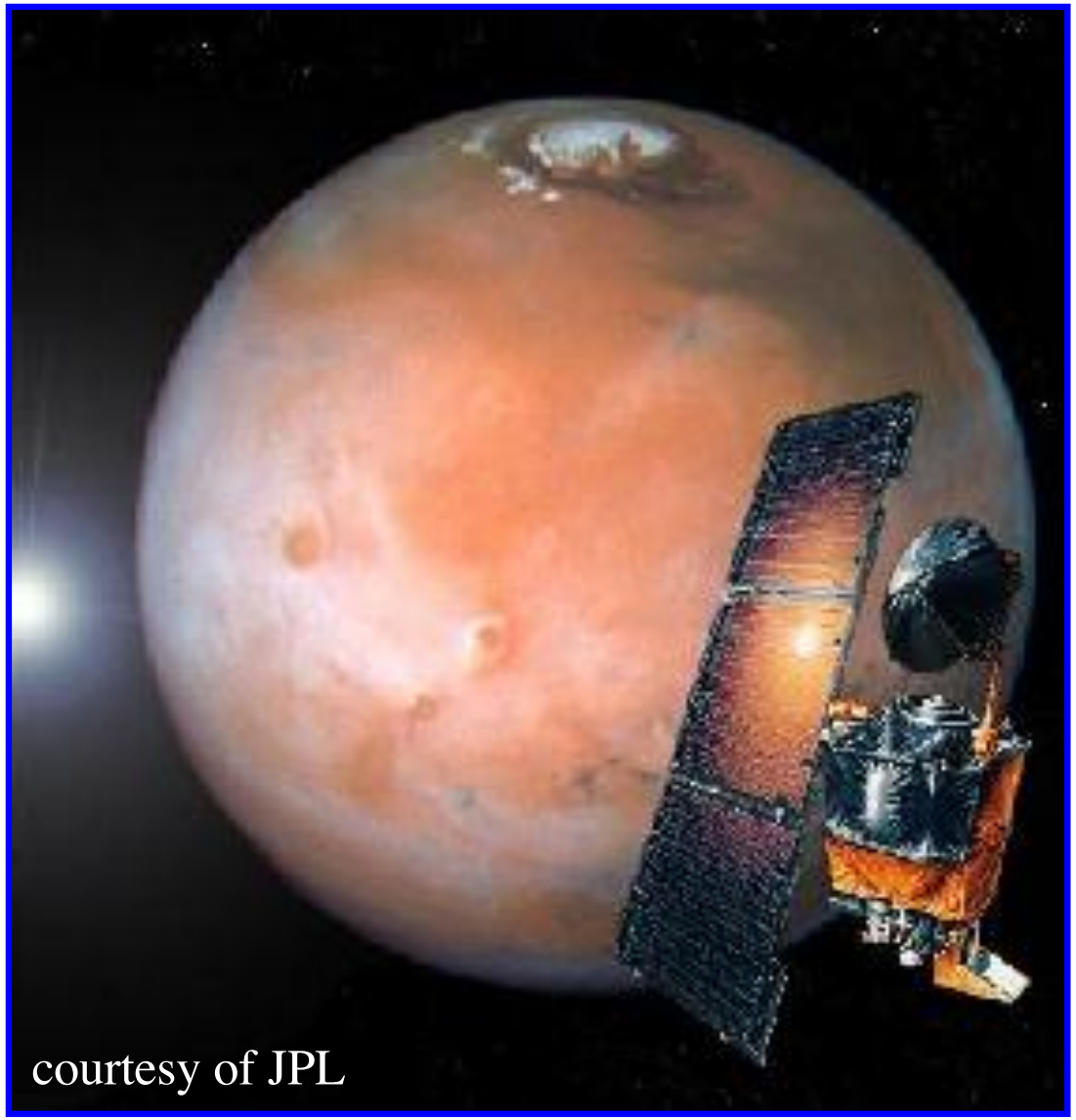
- Causal links:
  - Encode what needs to be true, and when
  - Offline (check completeness) and online (monitor)

- See Appendix for details.

Outline

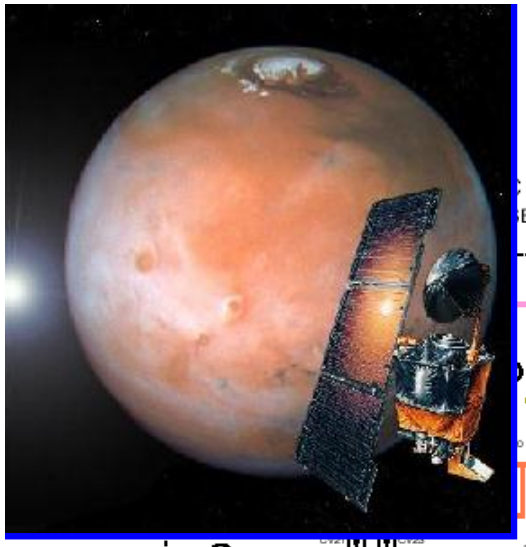
- Programs that Monitor State
  - Sub-goal monitoring
  - Model-based diagnosis and mode estimation
- Programs that Self-Diagnose (opt)

Motivation: Mission Loss Highlights the Challenge of Robustness

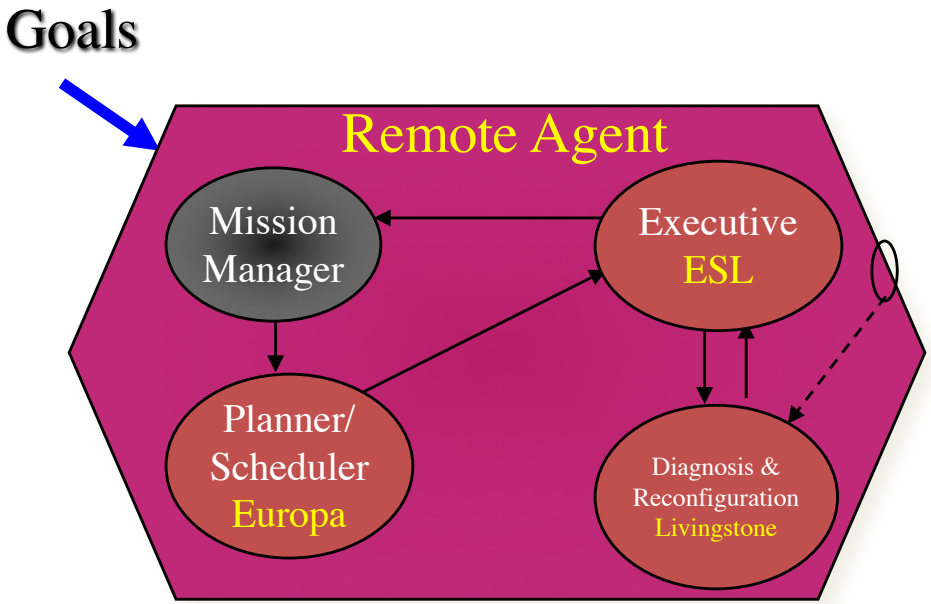


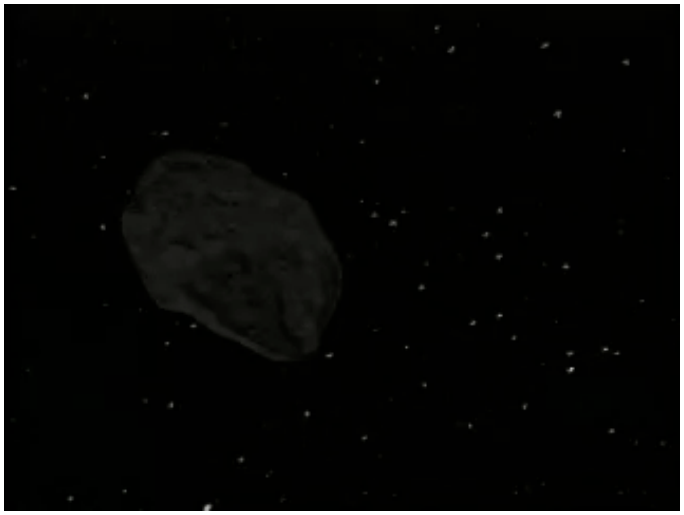
- Mars Observer
- Clementine
- Mars Climate Orbiter
- Mars Polar Lander

Mars Observer Failure



Remote Agent on Deep Space One - May, 1999



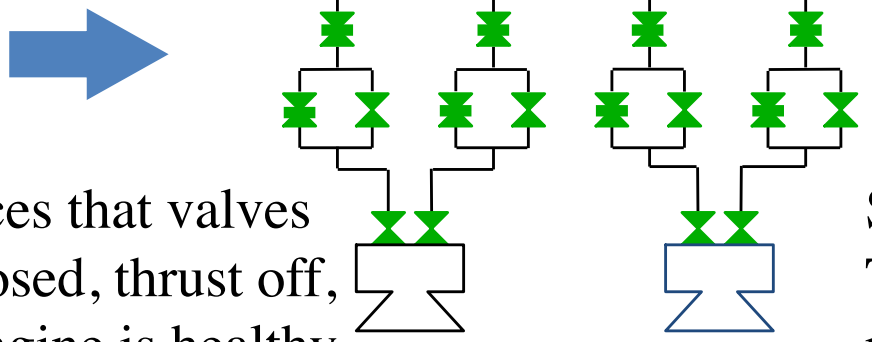


1. Commanded by giving goals
2. Reasons from commonsense models
3. Closes loop on goals
4. Diagnoses, repairs and re-plans.

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

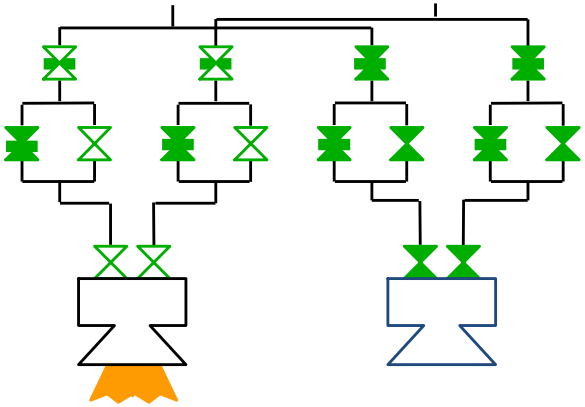
Executive assigns EngineA = Thrusting, and Livingstone. . . .

Estimate Modes



Deduces that valves are closed, thrust off, and engine is healthy

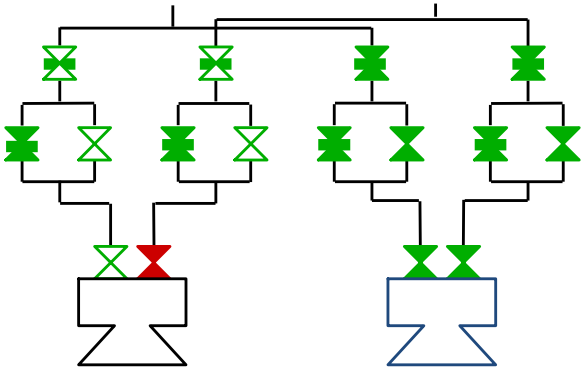
Reconfigure Modes



Selects valves To open, and plans actions to open valves

Deduces that a valve failed - stuck closed

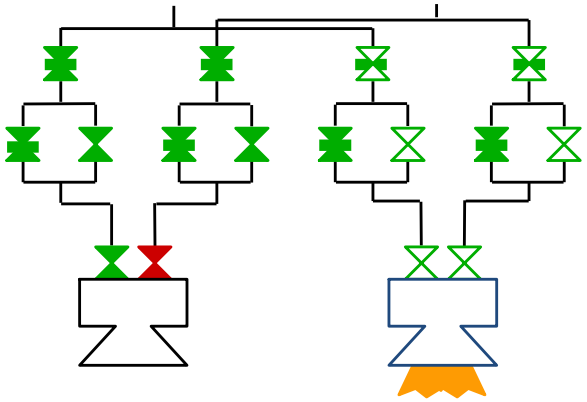
Estimate Modes



Prog: EngineB = Thrusting

Selects valves in the backup engine B that will achieve thrust, and plans needed actions.

Reconfigure Modes



Model: Probabilistic Constraint Automata (PCA)

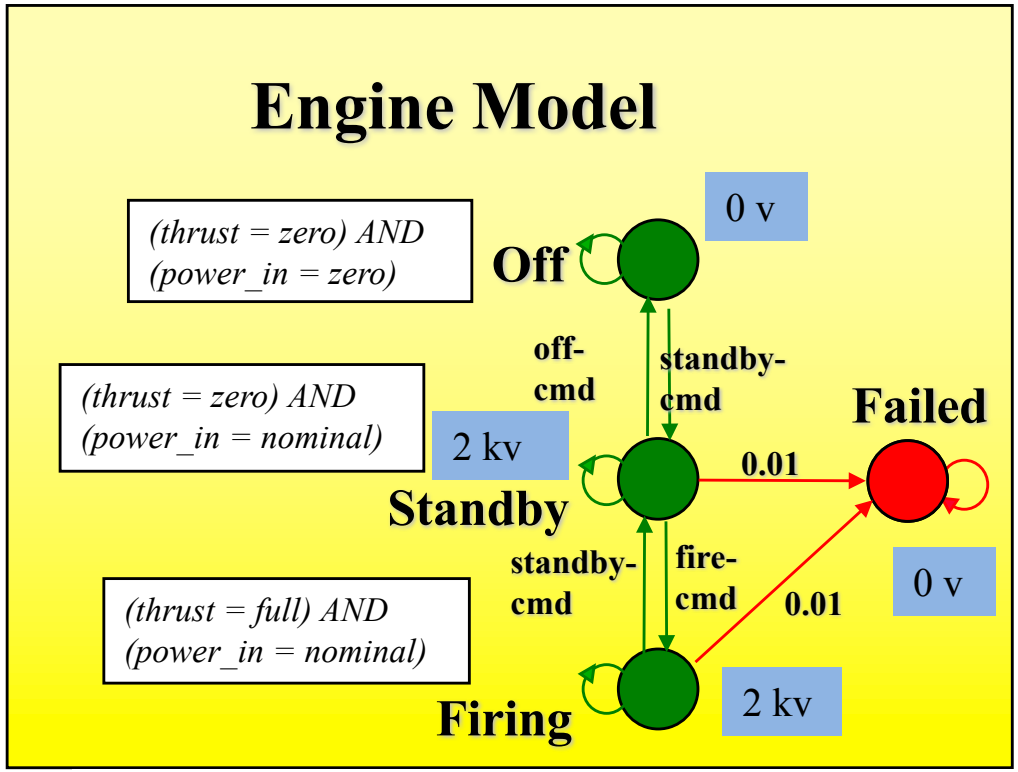
component modes…

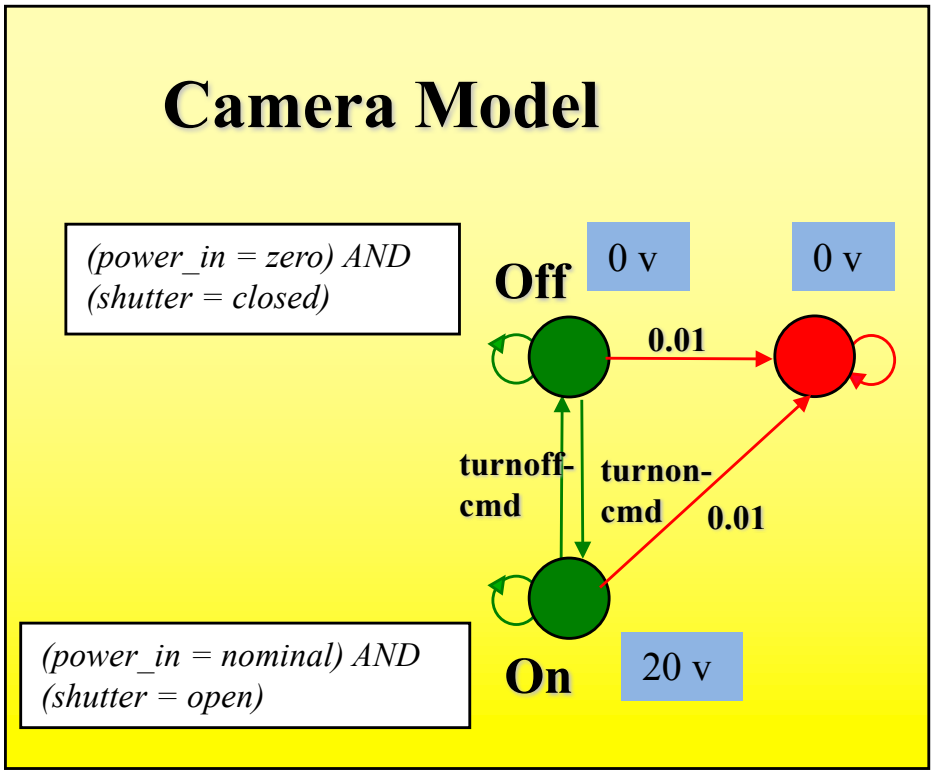
described by finite domain constraints on variables…

guarded deterministic and probabilistic transitions

cost / reward & prior distribution

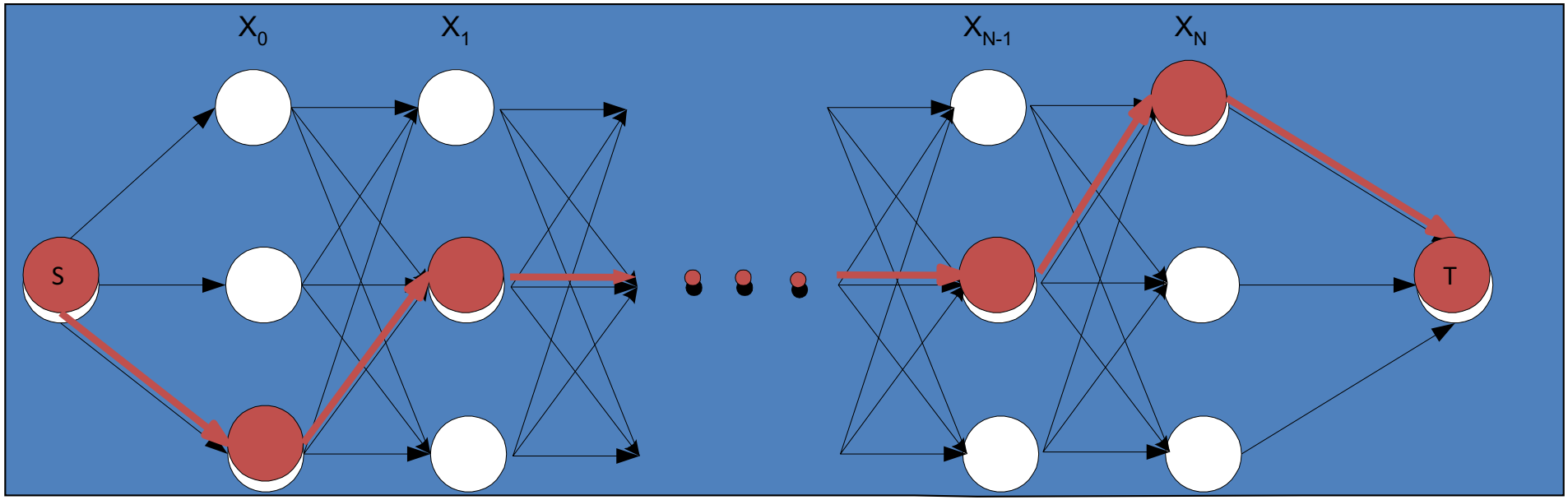
[Williams & Nayak AAAI 95, Williams et al. IEEE Proc. 01]





one per component … operating concurrently

Mode Estimation = Belief State Update



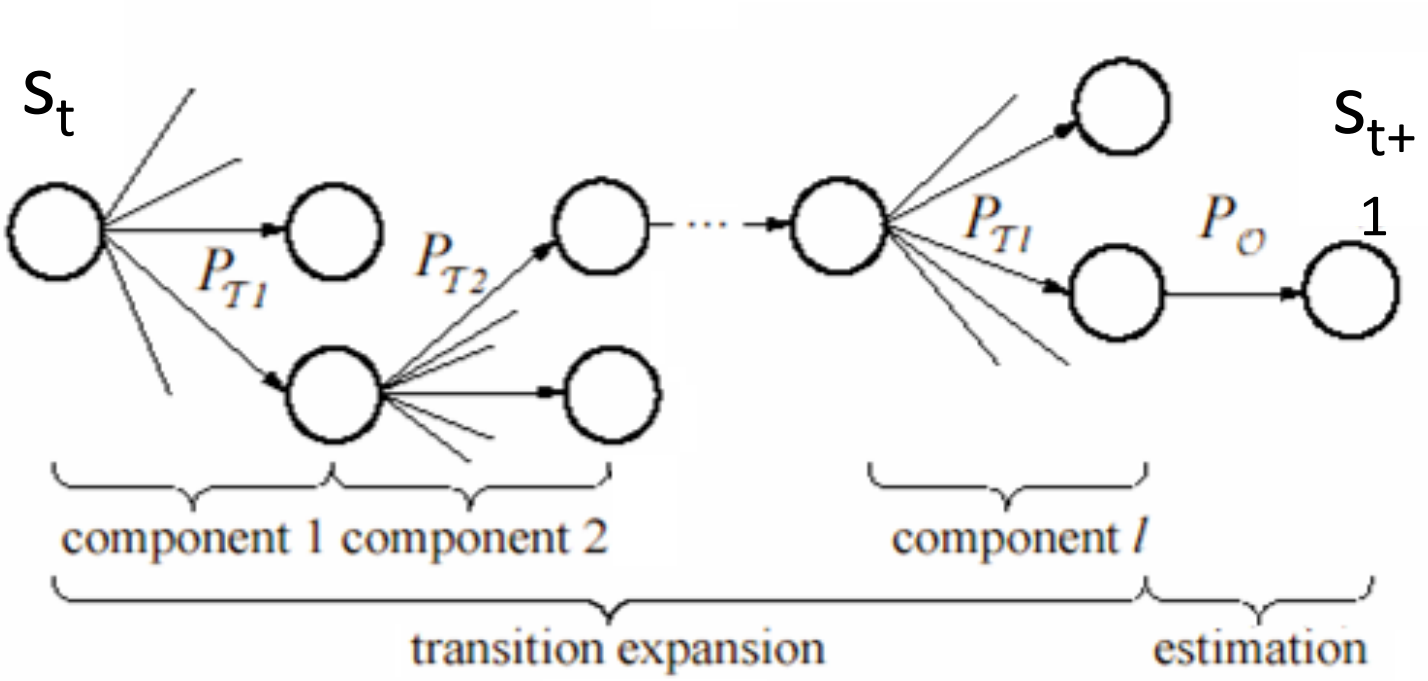
Given prior commands and observations:

1. What? Infer distribution on states (modes).
2. How? Infer most likely state *(mode)* trajectories.

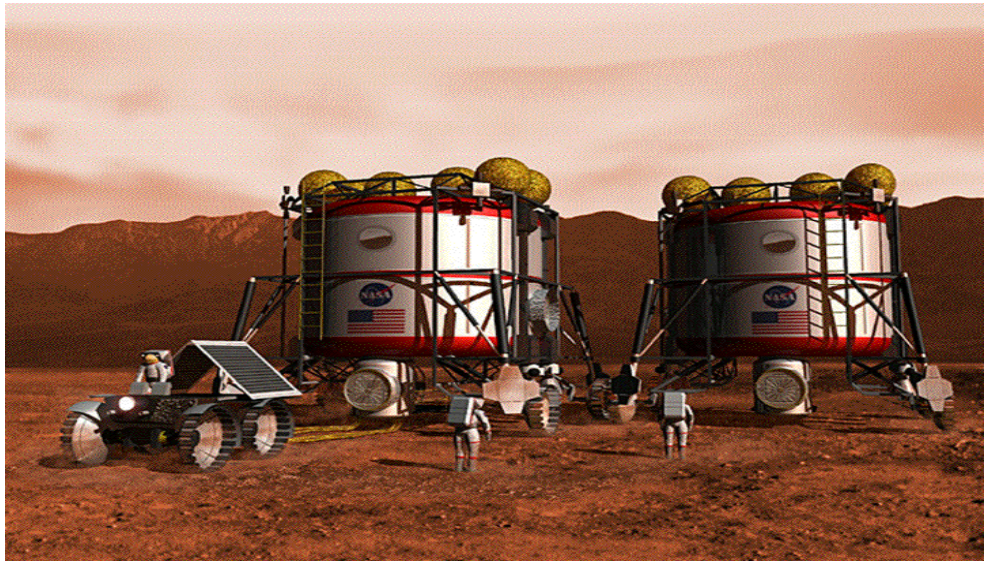
Approximate by enumerating most likely states or trajectories.

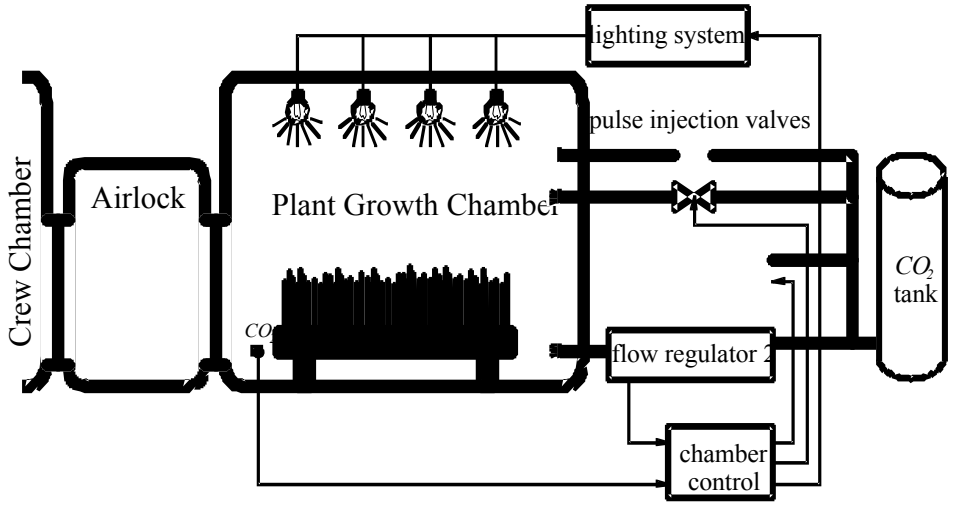
Best-first Tree Search at Each Time Step

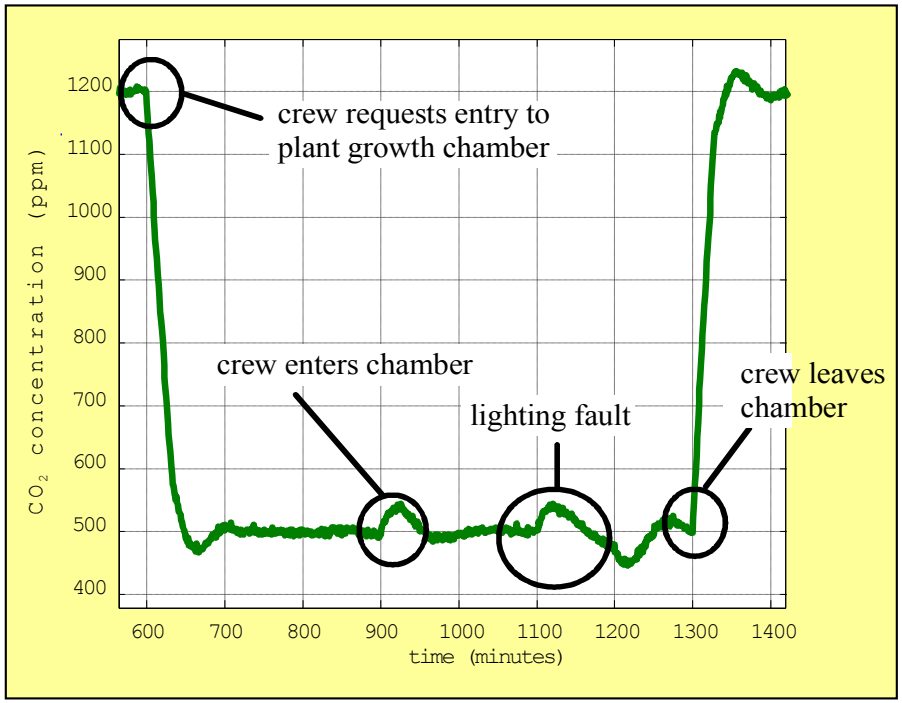
- Path search through component mode transitions, while checking constraints.
  - E.g., Enumerate best using conflict-directed A*



Detecting Subtle Failures: Hybrid Mode Estimation



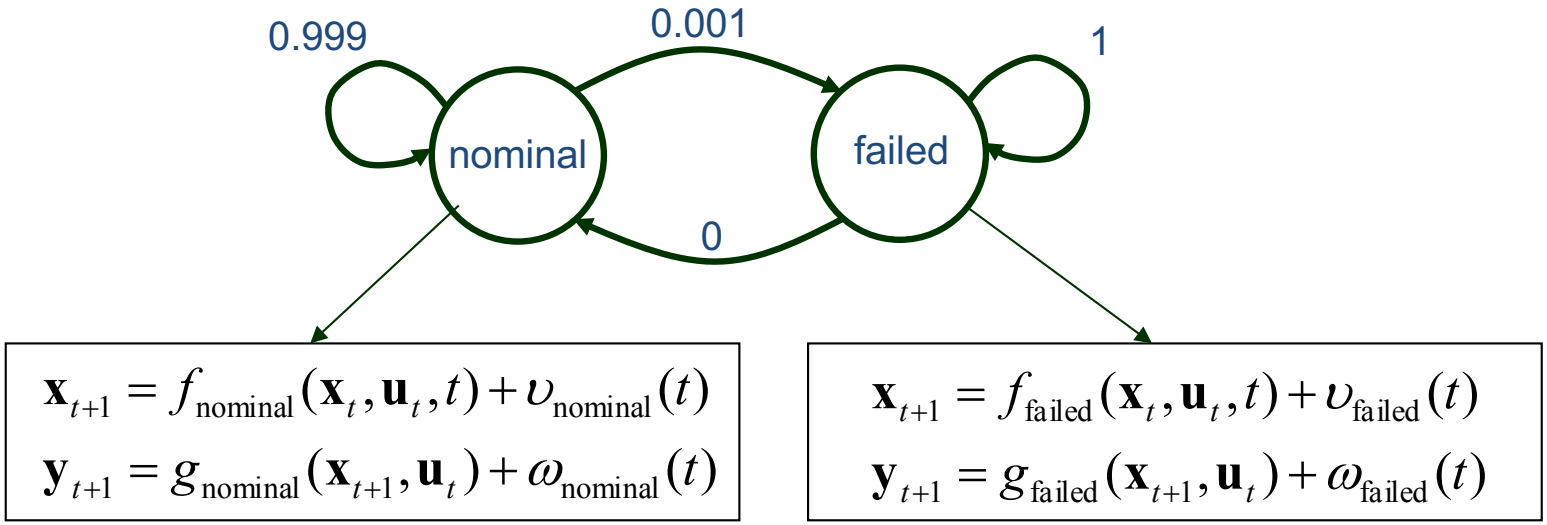




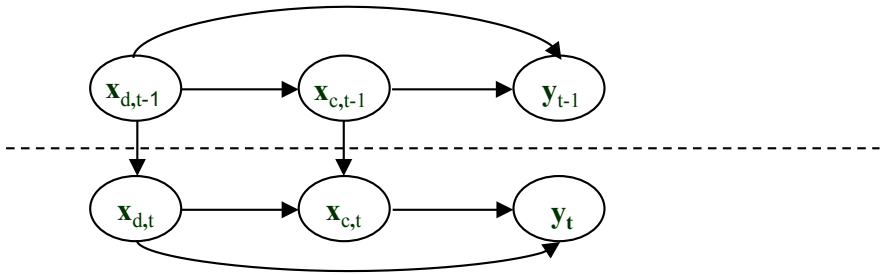
Hybrid:
- Identify discrete failure mode.
- Interpret observations using continuous model.

Model – Hybrid HMM

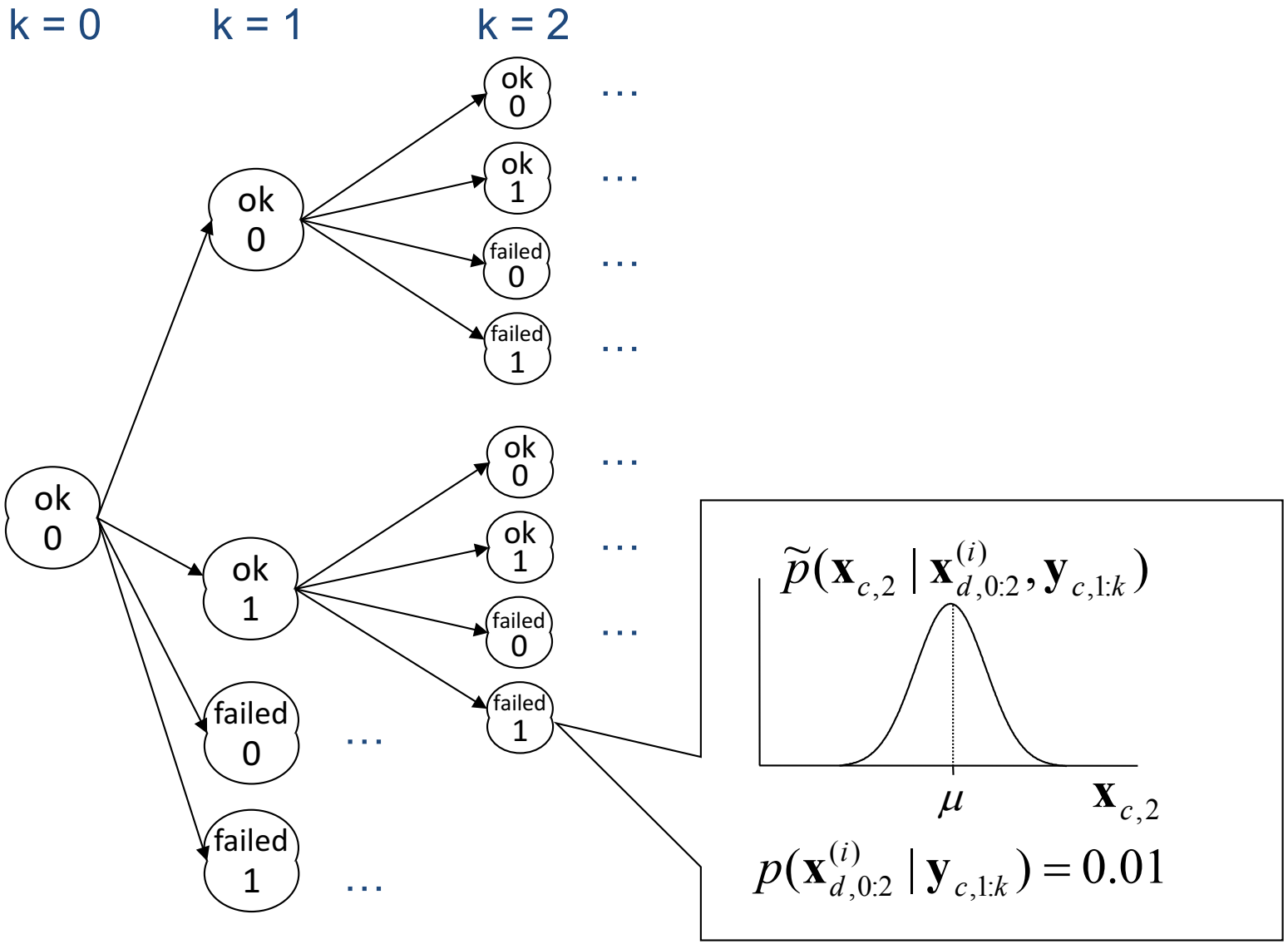
- **Discrete modes** and transitions between them



- **Continuous dynamics** corresponding to each mode



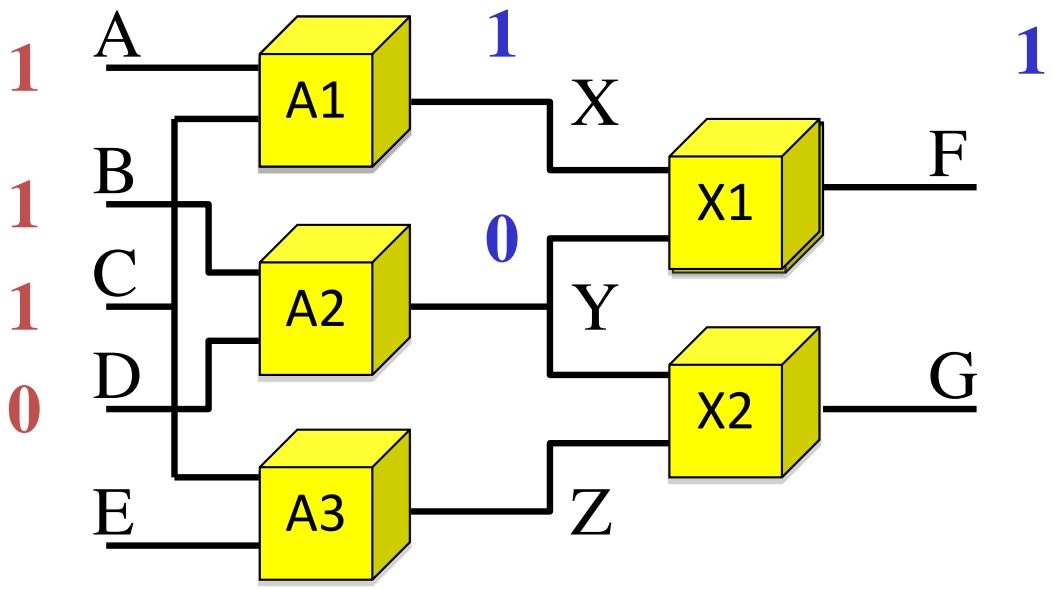
Kalman Filters Track Trajectories



Outline

- Programs that Monitor State
  - Sub-goal monitoring
  - Model-based diagnosis and mode estimation
    - Static mode estimation
    - Optimal CSPs and conflict-directed A*
- Programs that Self-Diagnose (opt)

Pedagogical Example: Polycell



OR	0	1
0	0	1
1	1	1

AND	0	1
0	0	0
1	0	1

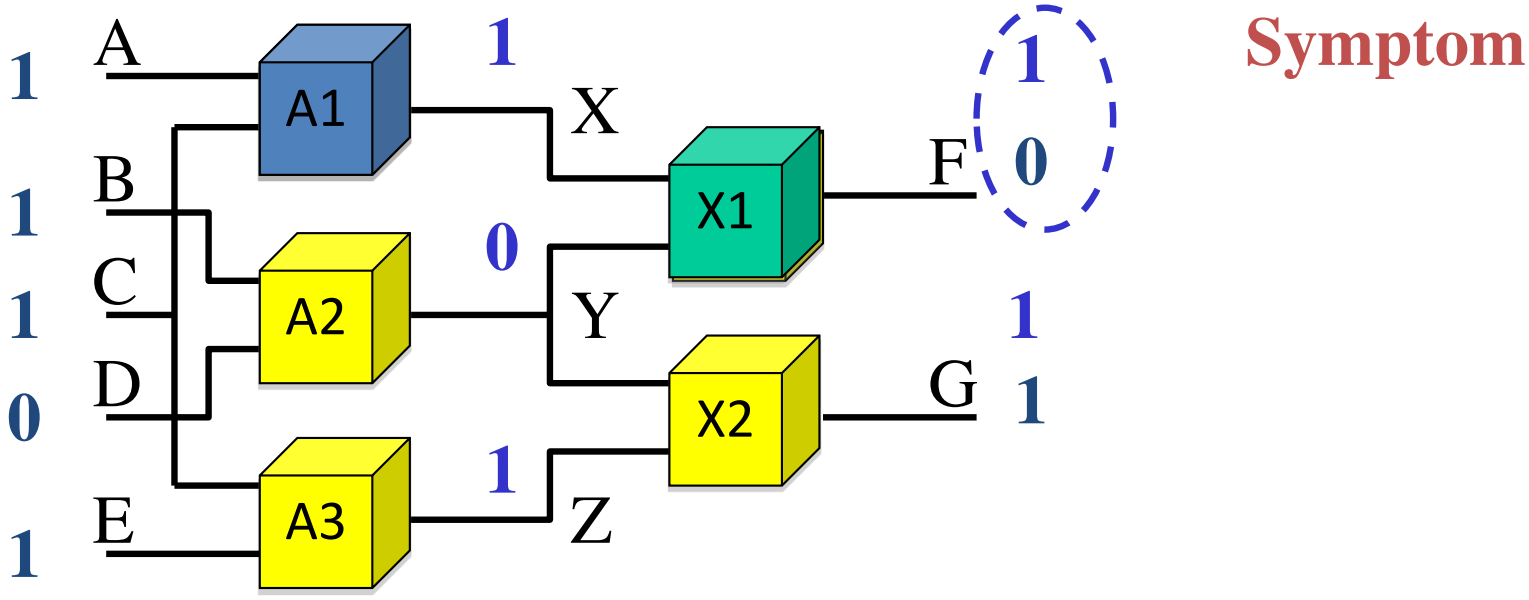
XOR	0	1
0	0	1
1	1	0

NOT	
0	1
1	0

Model-based Diagnosis

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

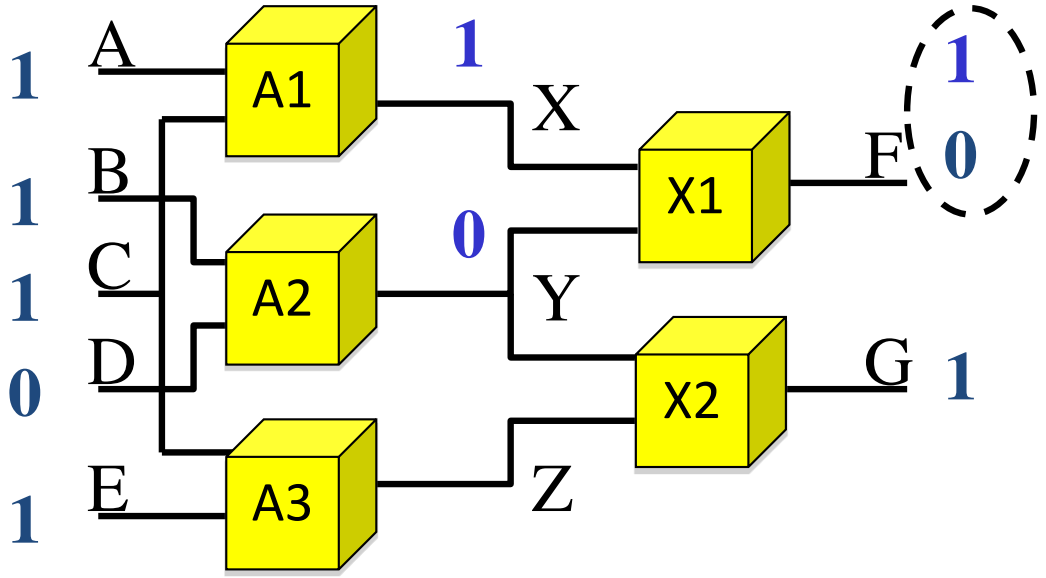
Output: Diagnoses that account for the symptoms.



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.



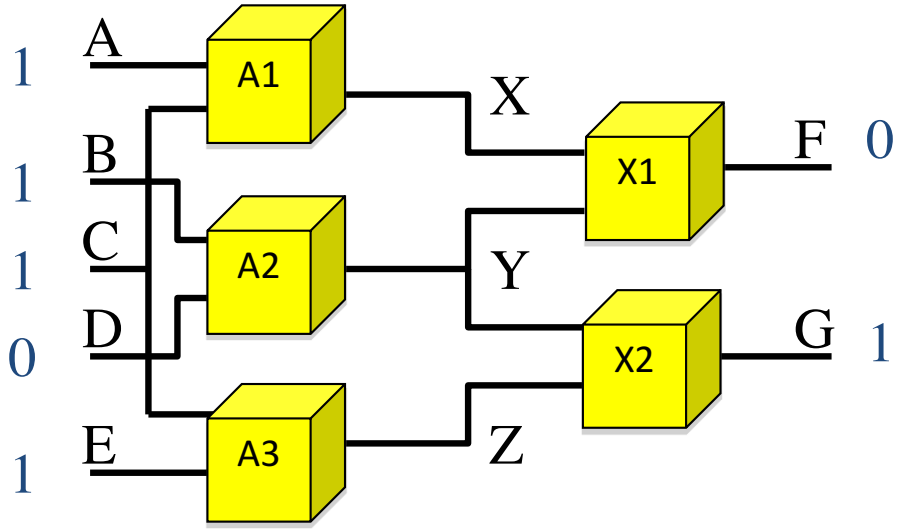
Symptom

[deKleer & Brown, 83]
[Davis, 84]
[Geneserth, 84]

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

And(I):

- I=G:
  Out(i) = In1(i) AND In2(i)
- I=U:
  True



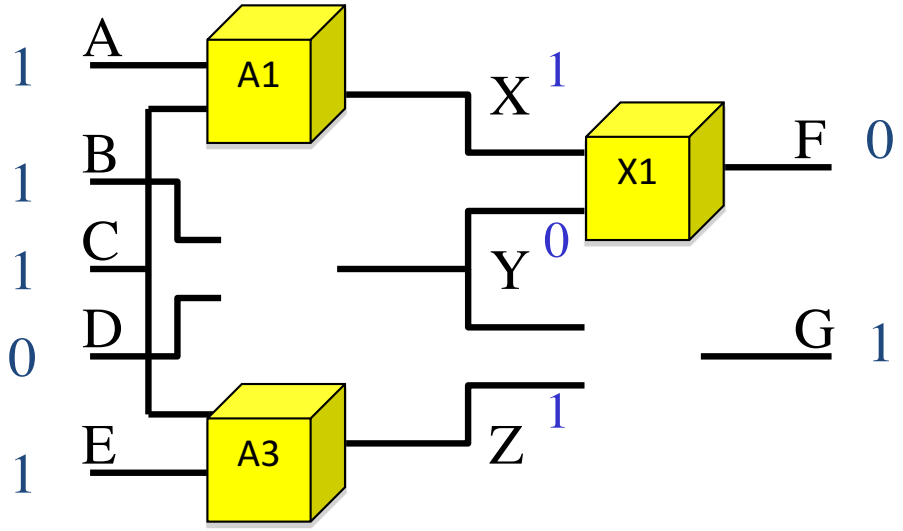
Candidate = {A1=G, A2=G, A3=G, X1=G, X2=G}

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

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

And(I):

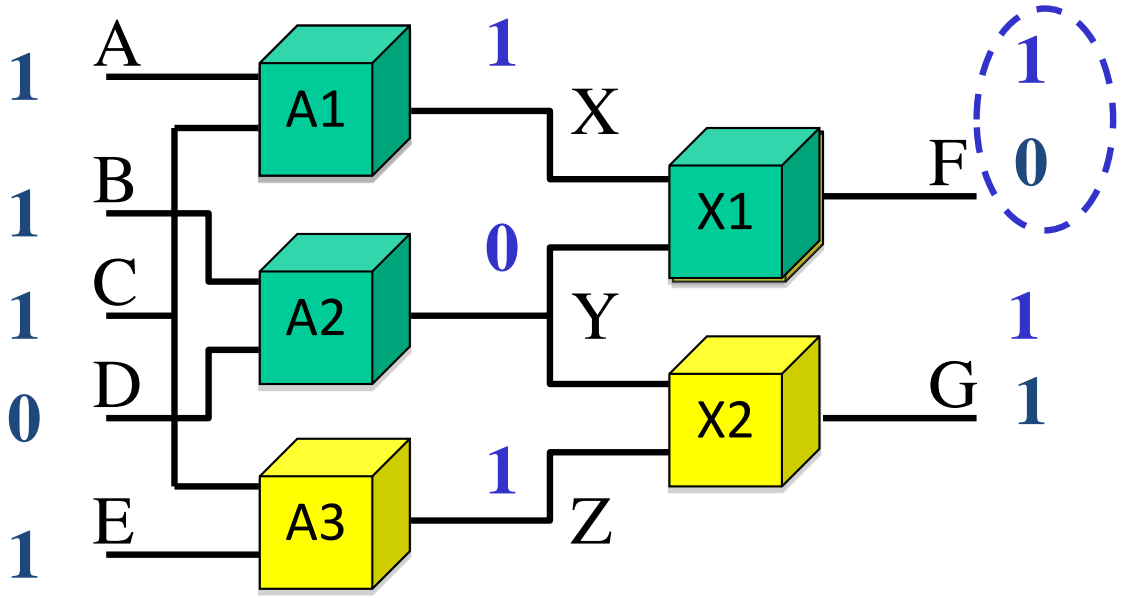
- I=G:
  Out(i) = In1(i) AND In2(i)
- I=U:
  True



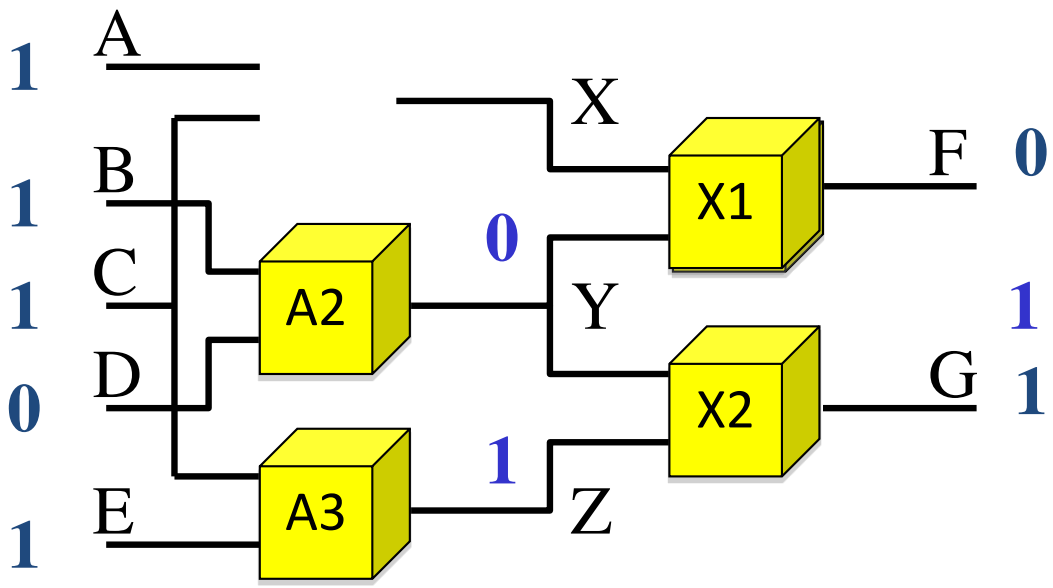
Diagnosis = {A1=G, A2=U, A3=G, X1=G, X2=U}

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

Consistency-based Diagnoses

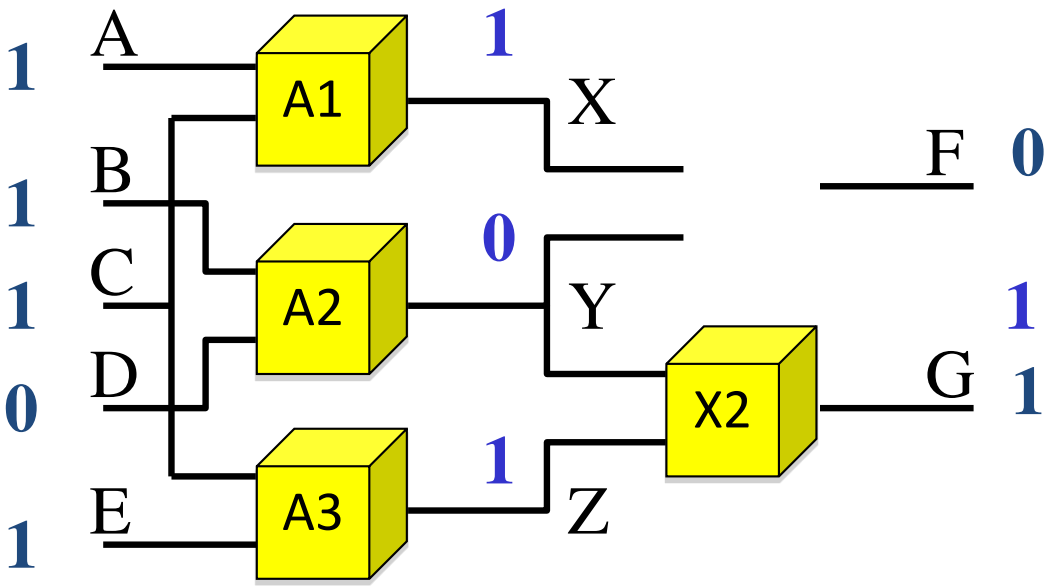


Consistency-based Diagnoses



{A1= U}

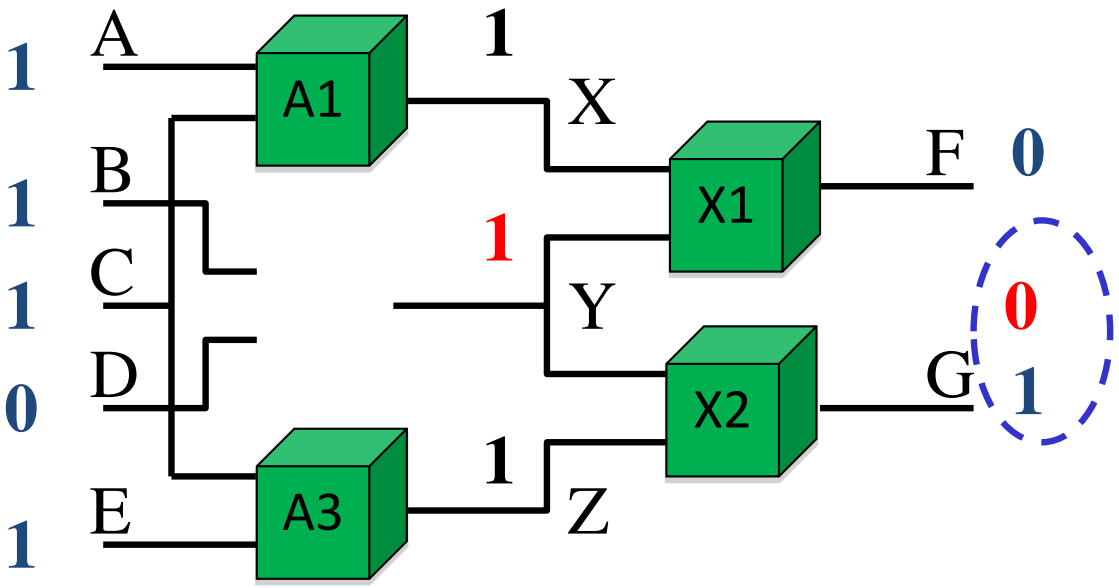
Consistency-based Diagnoses



{A1= U}

{X1 = U}

Consistency-based Diagnoses



Symptom

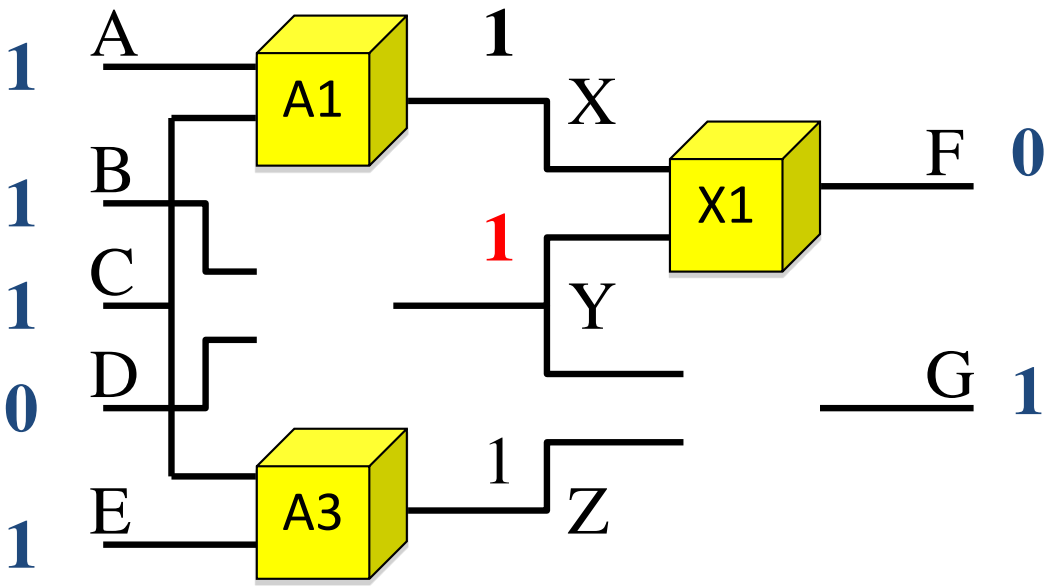
{A1= U}

{X1 = U}

~~{A2= U}?~~

What, in addition to A2, might be faulty?

Consistency-based Diagnoses



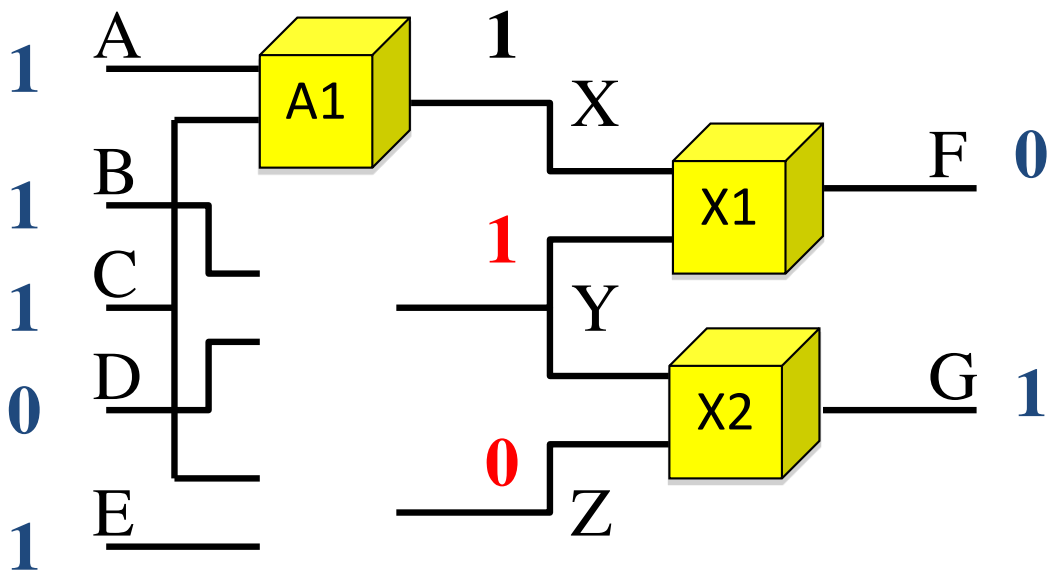
What, in addition to A2, might be faulty?

{A1= U}

{X1 = U}

{A2 = U, X2 = U}

Consistency-based Diagnoses



What, in addition to A2, might be faulty?

- These diagnoses are minimal.
- All extensions are also diagnoses.

⇒Kernel Diagnoses

{A1= U}

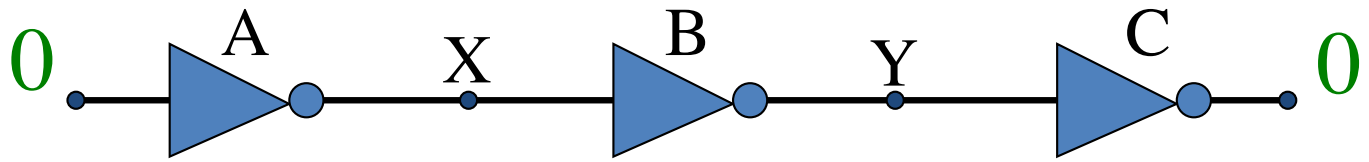
{X1 = U}

{A2 = U, X2 = U}

{A2 = U, A3 = U}

Incorporating Failure Modes: Mode Estimation

Sherlock
[de Kleer & Williams, IJCAI 89]



Inverter(I):

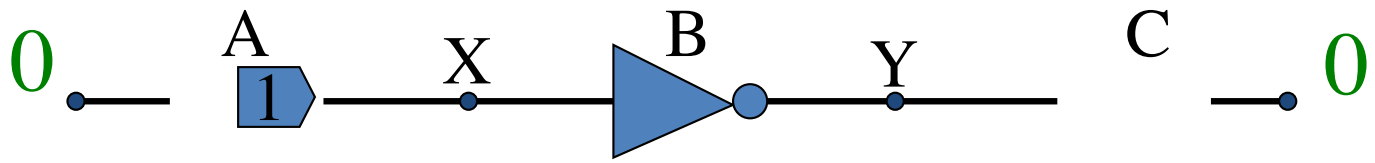
- I=G: Out(i) = not(In(i))
- I=S1: Out(i) = 1
- I=S0: Out(i) = 0
- I=U: True

- Isolates unknown.
- Explains.

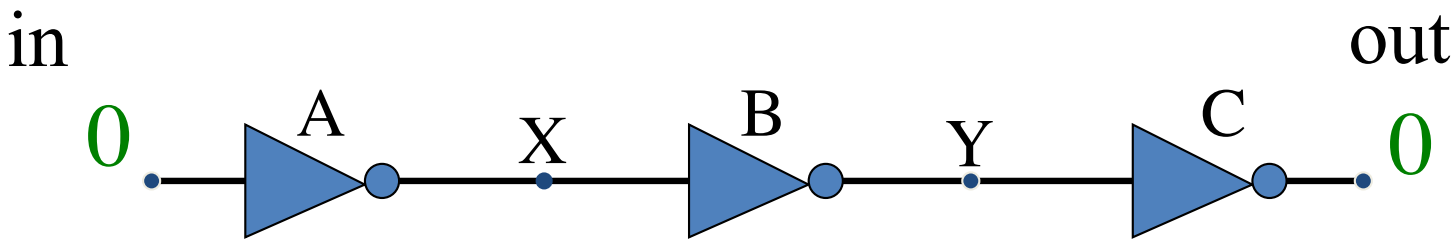
Good, Faulty and Unknown Modes

Example Mode Estimate

Sherlock
[de Kleer & Williams, IJCAI 89]



Diagnosis: [A=S1, B=G, C=U]



Diagnoses: (42 of 64 candidates)

Sherlock
[de Kleer & Williams, IJCAI 89]

Fully Explained Failures

- [A=G, B=G, C=S0]
- [A=G, B=S1, C=S0]
- [A=S0, B=G, C=G]

. . .

Partially Explained

- [A=G, B=U, C=S0]
- [A=U, B=S1, C=G]
- [A=S0, B=U, C=G]

. . .

Fault Isolated, But Unexplained

- [A=G, B=G, C=U]
- [A=G, B=U, C=G]
- [A=U, B=G, C=G]

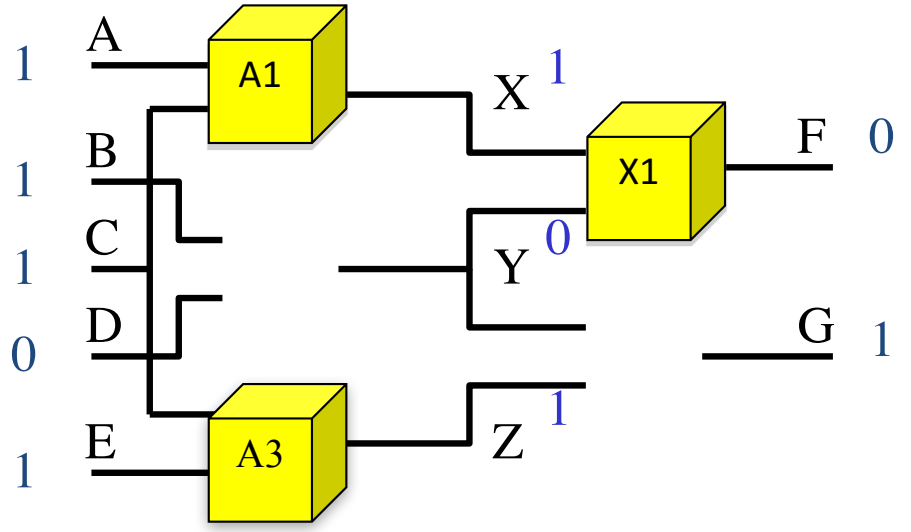
Mode Estimation Problem

Sherlock
[de Kleer & Williams, IJCAI 89]

Given:

- Mode, State, Observable Variables: X, Y, O ⊆ Y
- Model: Φ(X,Y) = components + structure

And(I):
  G(I):
    Out(i) = In1(i) AND In2(i)
  U(I):

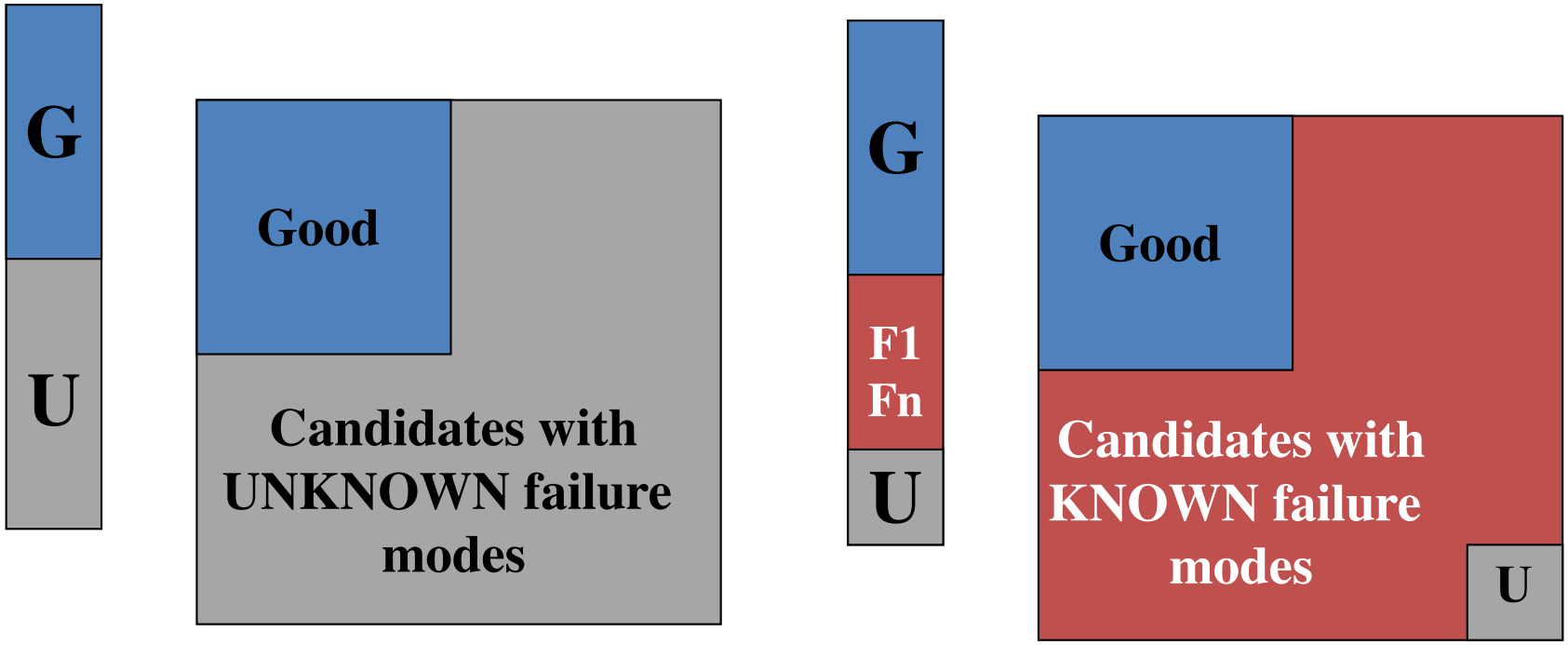


- All behaviors associated with modes.
- All components have "unknown mode" U, which enables no constraint.

Return: All diagnoses

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

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



Ideas:

1. Compute probability of each mode estimate.
2. Enumerate most likely mode estimates X_i, based on probability.

Prefix (k) (Sort $\{X_i\}$ by decreasing $P(X_i \mid O)$)

Probabilistic, Mode Estimation

Inputs:

- Mode X, State Y, and Observation (O subset Y) variables, with finite domains.
- Model Φ(X;Y).
- Observations O = o.
- **Prior distribution P(X).**
- **Observation function P(o | X, Φ(X;Y)) = P(o | X).**

Outputs:

- Exact: P(X | o) Posterior, given observations.
- Approximate: k most likely mode estimates X_i,
  [Prefix (k) (Sort {X_i} by decreasing P(X_i | o))].

A Simple Approximation for Probabilistic, Mode Estimation

$$P(X|o) = \frac{P(o|X)P(X)}{P(o)} \quad = \alpha P(o|X)P(X)$$

$$P(o) = \frac{1}{\alpha} = \sum_{c_i \in C} P(o|c_i)P(c_i) \qquad \text{for candidates C consistent with o}$$

1. Assume modes are *a priori* independent:

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

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

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

A Simple Approximation for Probabilistic, Mode Estimation

$$P(X|o) = \frac{P(o|X)P(X)}{P(o)} \quad = \alpha P(o|X)P(X)$$

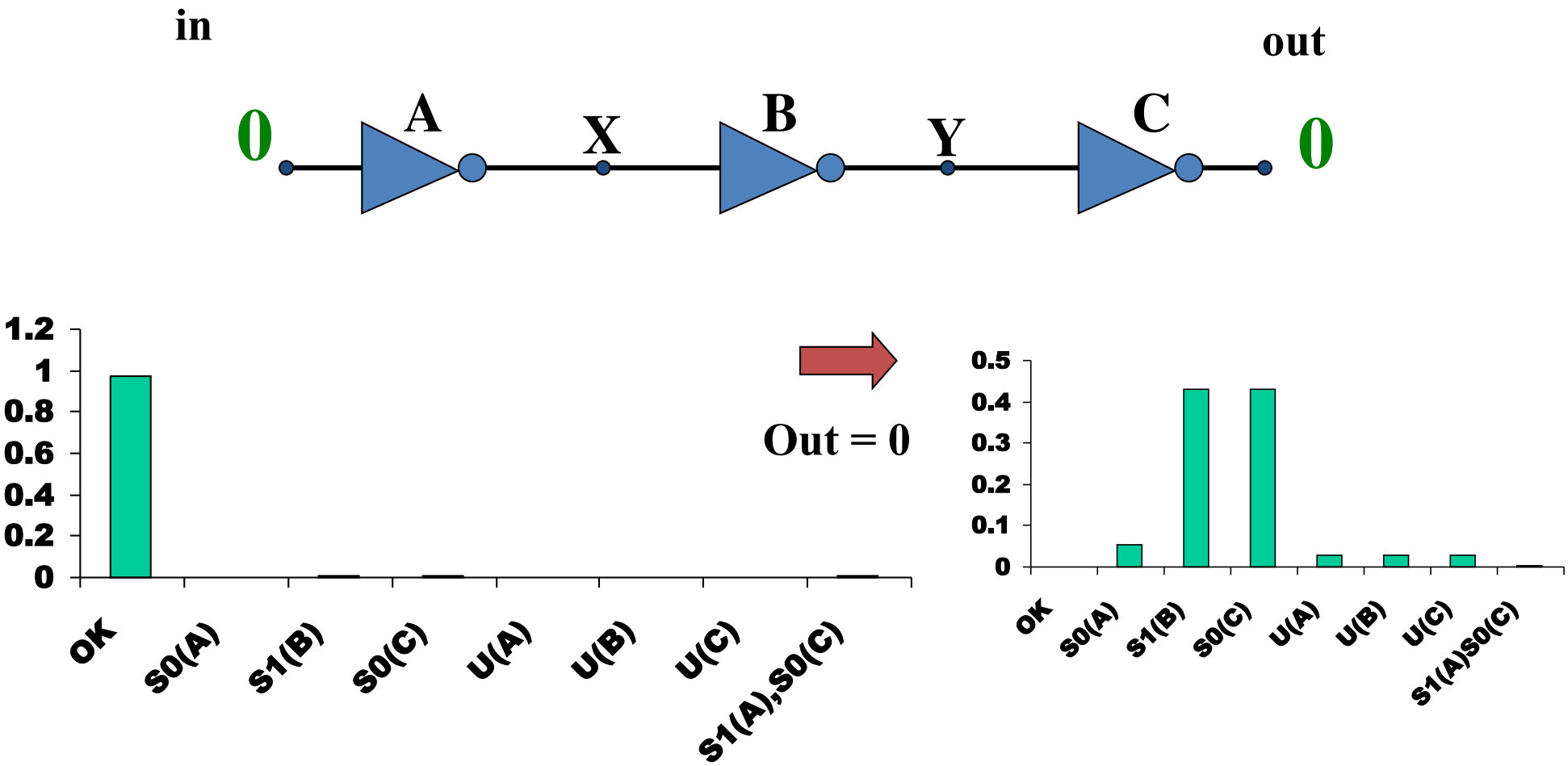
$$P(o) = \frac{1}{\alpha} = \sum_{c_i \in C} P(o|c_i)P(c_i)$$ for candidates C consistent with o

1. Assume modes are *a priori* independent:

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

2. Assume consistent ***interpretations*** are equally likely, given observations, model and mode assignment.

⇒ Model Counting: Use exhaustive DPLL to count models.



Top 6 of 64 = 98.6% of P

Leading mode estimates before and after output observed.

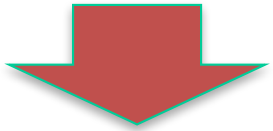
Outline

- Programs that Monitor State
  - Sub-goal monitoring
  - Model-based diagnosis and mode estimation
    - Static mode estimation
    - Optimal CSPs and conflict-directed A*
- Programs that Self-Diagnose (opt)

Optimal CSP

Mode Estimation: k most likely mode estimates X_i,

Prefix (k) (Sort $\{X_i\}$ by decreasing $P(X_i \mid o)$)



Given OCSP= <X, *f*, CSP>

- X are *decision variables*, with finite domain D_X.
- f: $D_X \rightarrow \Re$ is a utility function.
- CSP is a set of constraints over variables <X;Y>.

Find leading arg max $f(X)$

$$X \in D_X$$

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

Example: Encode Constraints in Propositional State Logic

$Y \in \{1, 0\}$

$I \in \{G, S1, S0, U\}$

$Y=1 \vee Y=0$

$\neg[Y=1 \wedge Y=0]$

$\cdots$

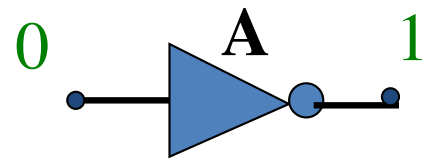
Inverter(I):

- I=G: Out(I) = NOT In(I)
- I=S1: Out(I) = 1
- I=S0: Out(I) = 0

$I=G \rightarrow [In(I)=1 \text{ iff } Out(I)=0]$

$I=S1 \rightarrow Out(I)=1$

$I=S0 \rightarrow Out(I)=0$



Alternative constraints:
- Simple temporal networks
- Linear programs
- Global constraints

$I=G \rightarrow [In(I)=1 \text{ iff } Out(I)=0]$

$[\neg(I=G) \vee \neg(In(I)=1) \vee Out(I)=0] \wedge$

$[\neg(I=G) \vee \neg(Out(I)=0) \vee In(I)=1]$

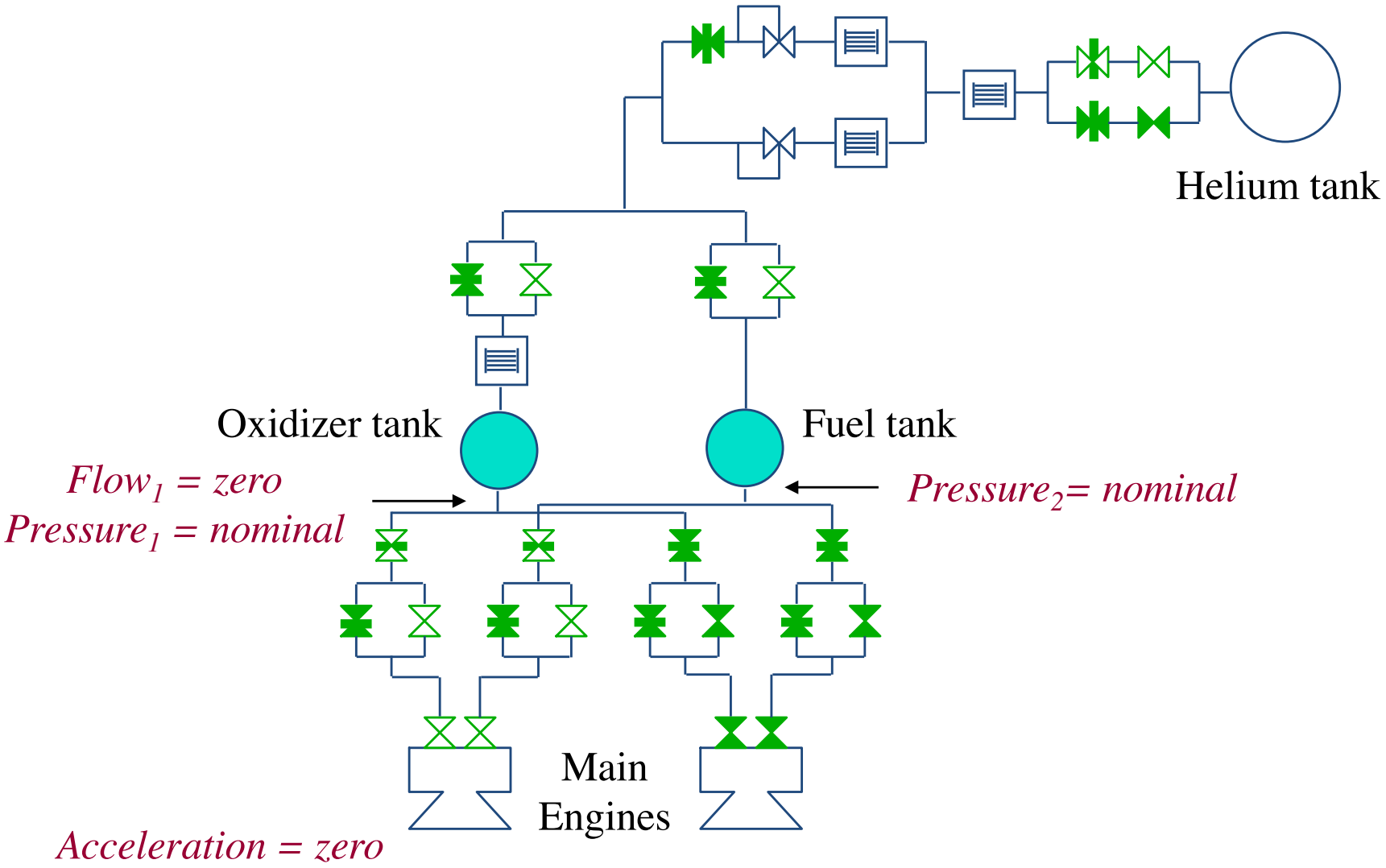
Model-based Diagnosis as Conflict-directed Best First Search

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

- Sherlock Holmes. The Sign of the Four.

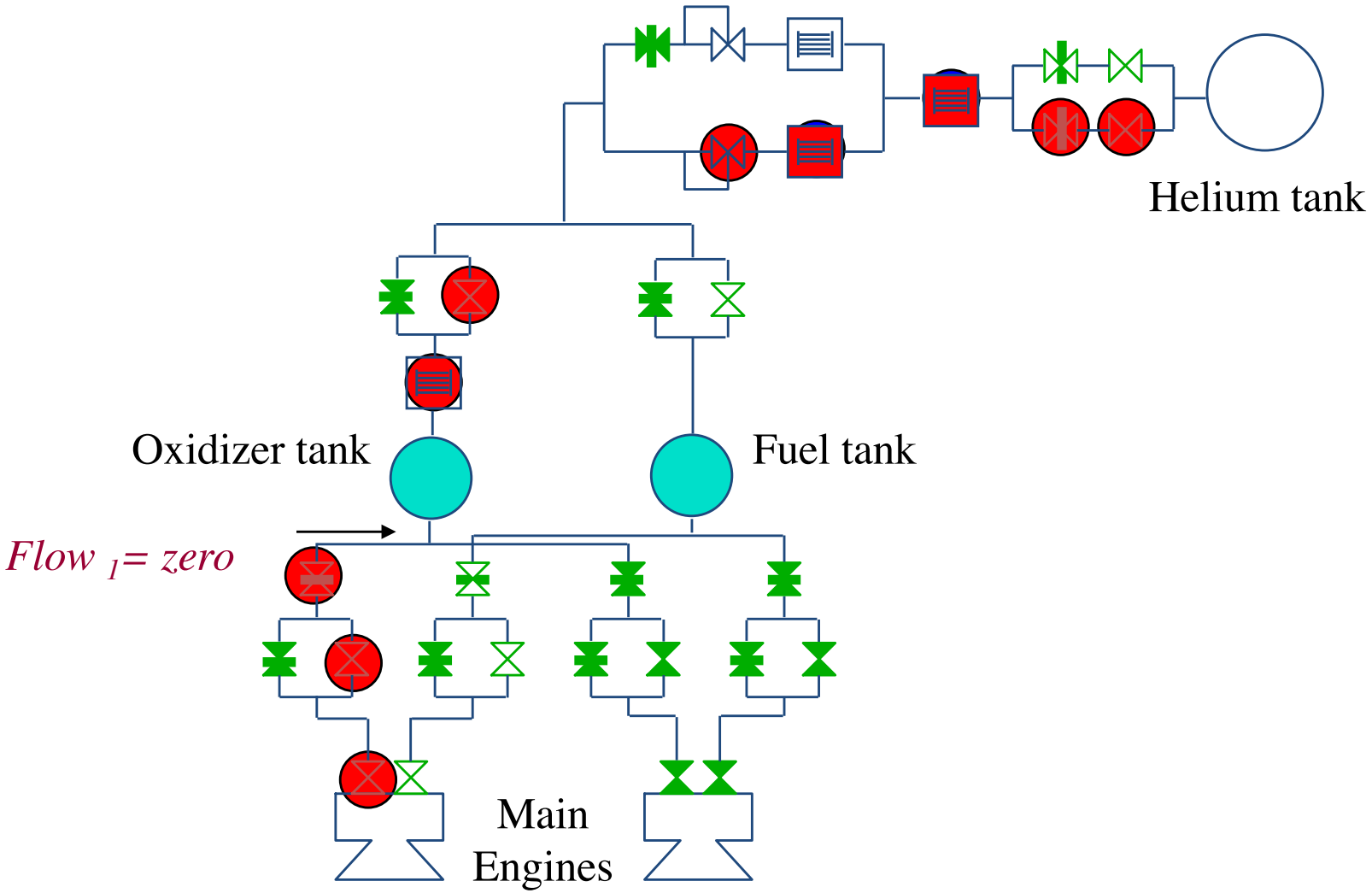
1. Generate most likely candidate.
2. Test candidate.
3. If Inconsistent, learn reason for inconsistency (a conflict).
4. Use conflicts to leap over similarly infeasible options to the next best candidate.

Compare Most Likely Candidate to Observations



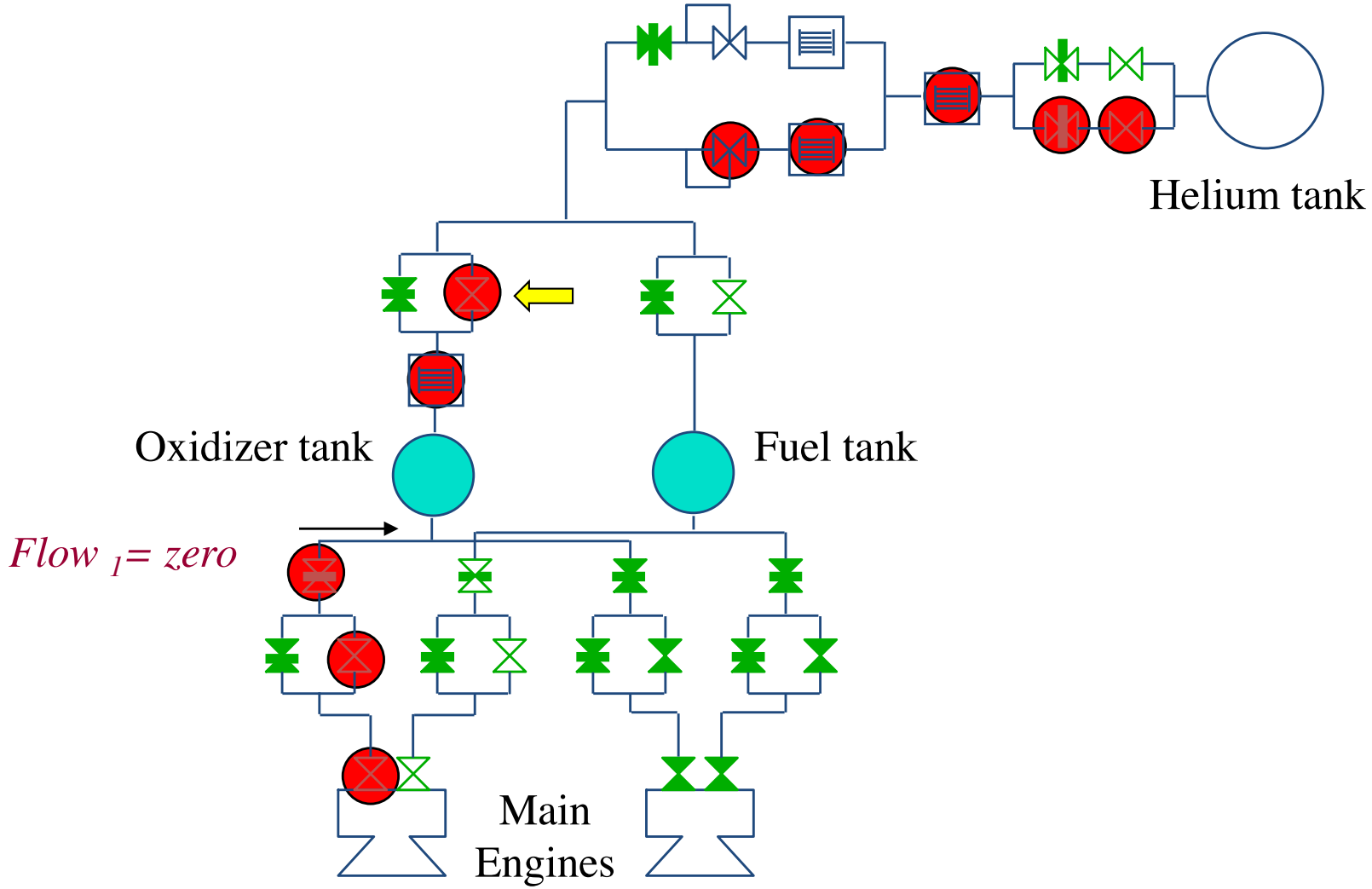
It is most likely that all components are okay.

Isolate Conflicting Information



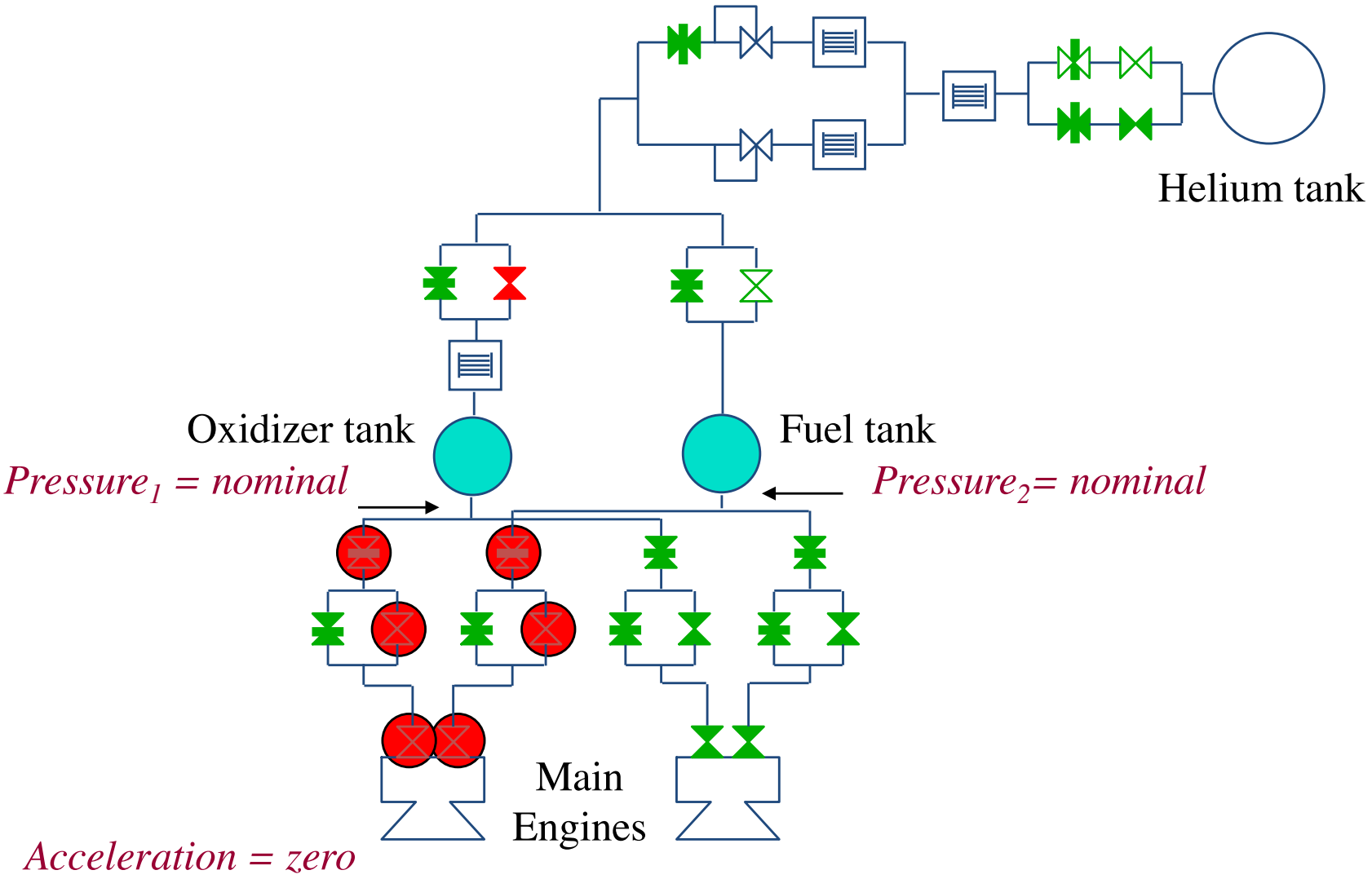
The red component modes *conflict* with the model and observations.

Leap to the Next Most Likely Candidate that Resolves the Conflict



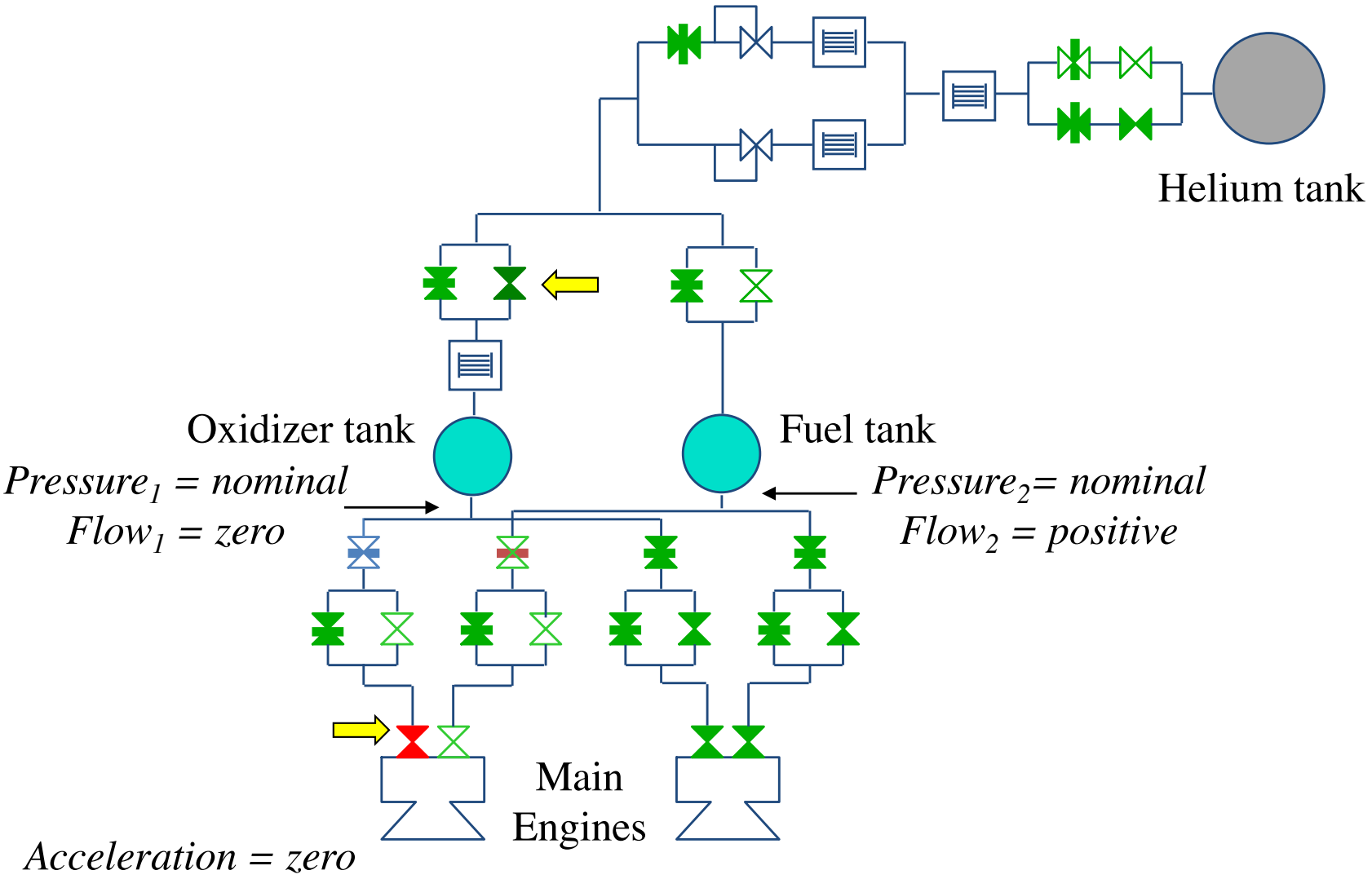
The next candidate must remove the conflict.

New Candidate Exposes Additional Conflicts



Another conflict, try removing both.

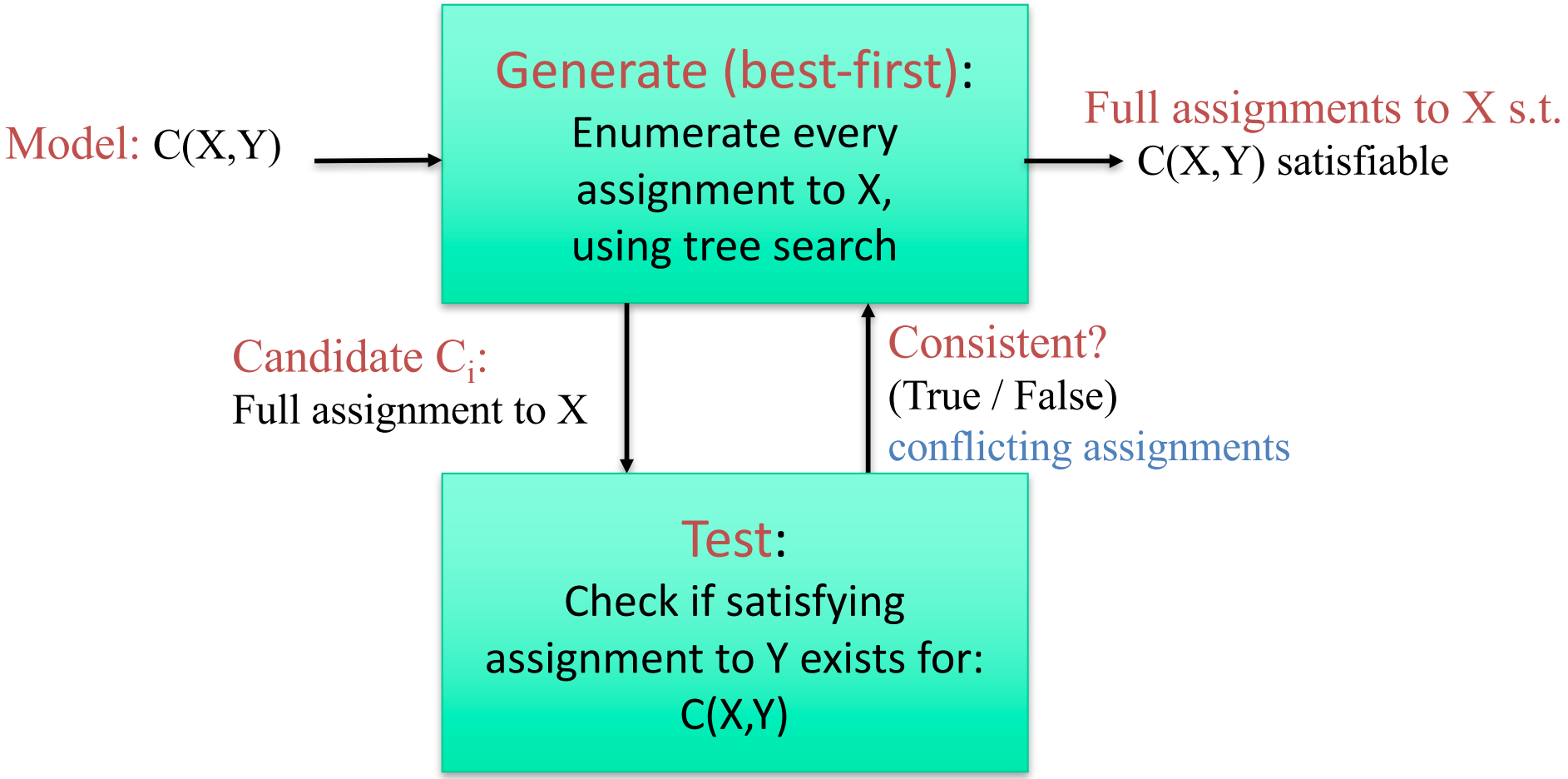
Final Candidate Resolves all Conflicts



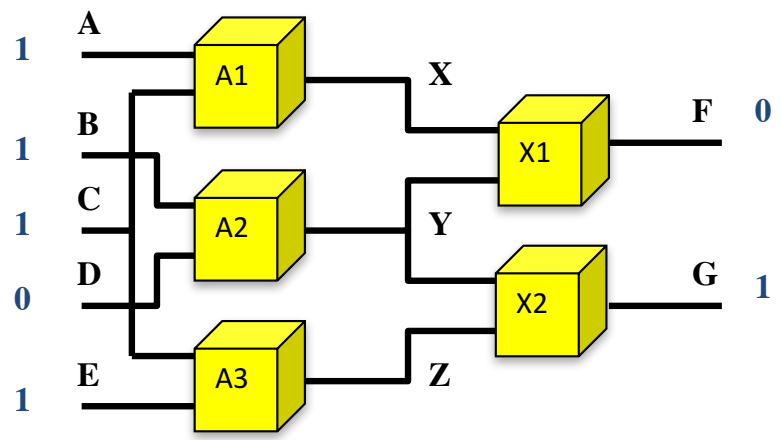
Implementation: Conflict-directed A* search.

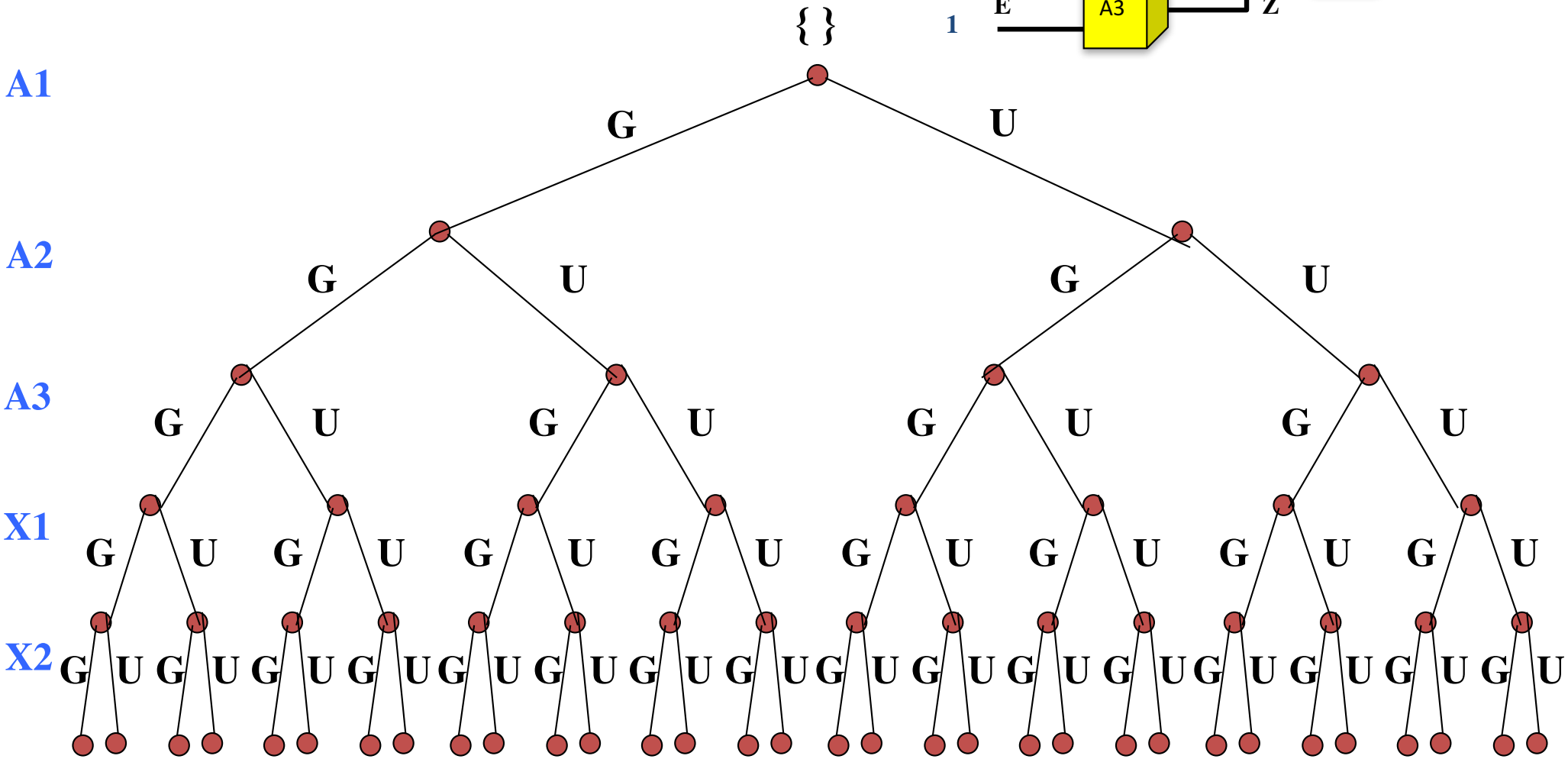
Solver: OpSat

$$C_\Phi \equiv \{X \in D_X | C(X,Y)\ satisfiable\}$$



Search Tree of Mode Assignments



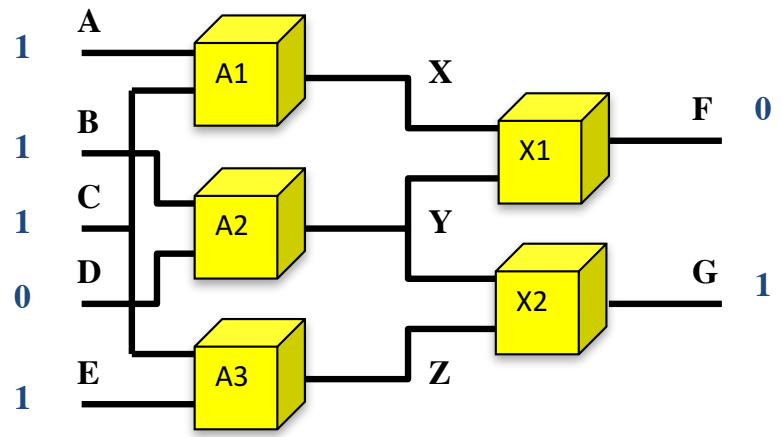


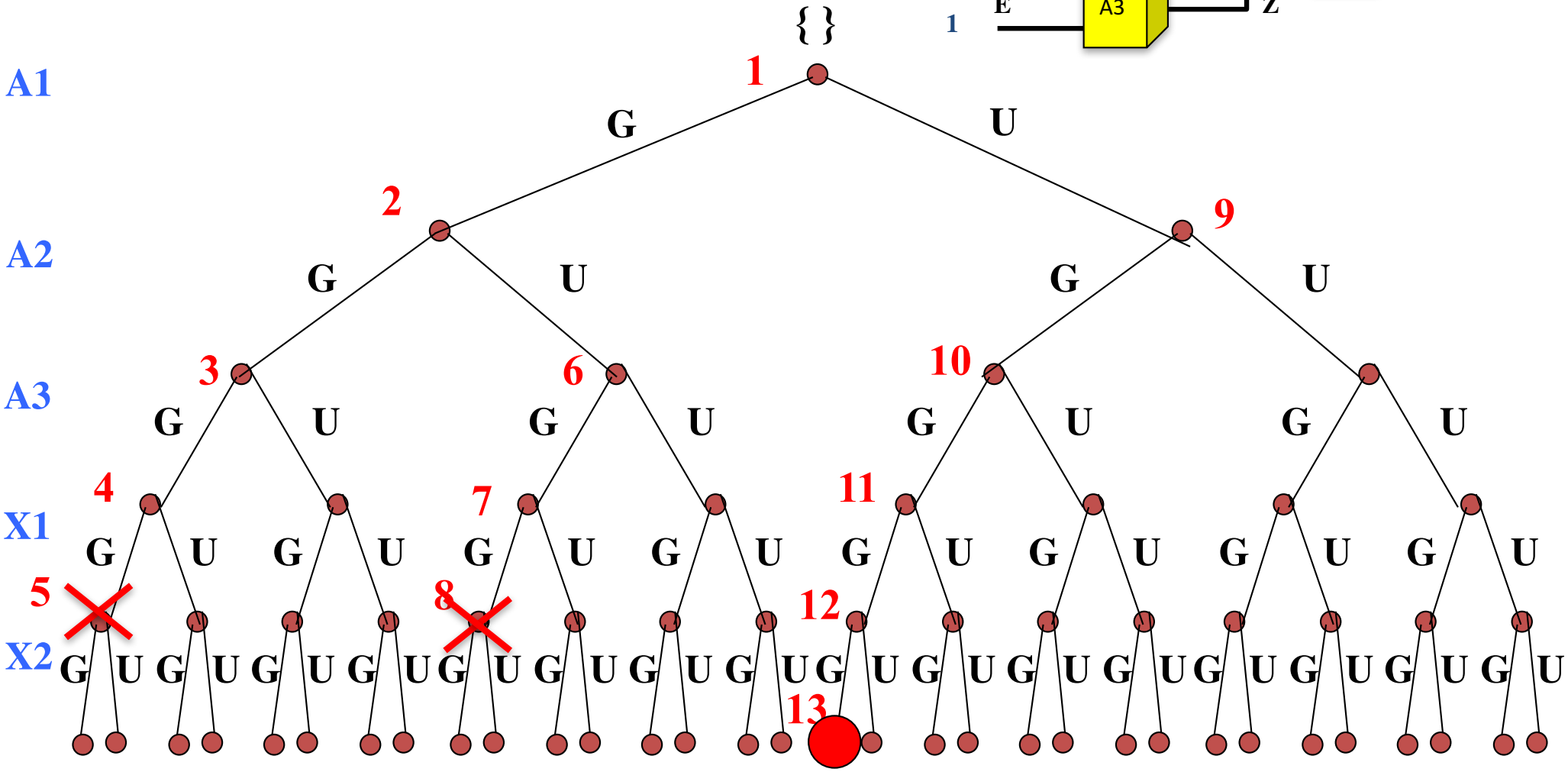
Constraint-based A*

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

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

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

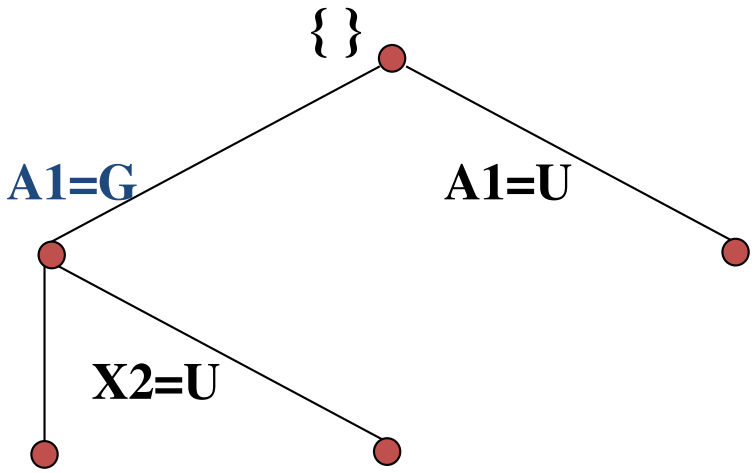




Optimal CSP Candidate Generation using Constraint-based A*

Initial State:

- Empty assignment {}.



Child Expansion:

- Select unassigned variable y_i.
- Each child adds an assignment from D_i.

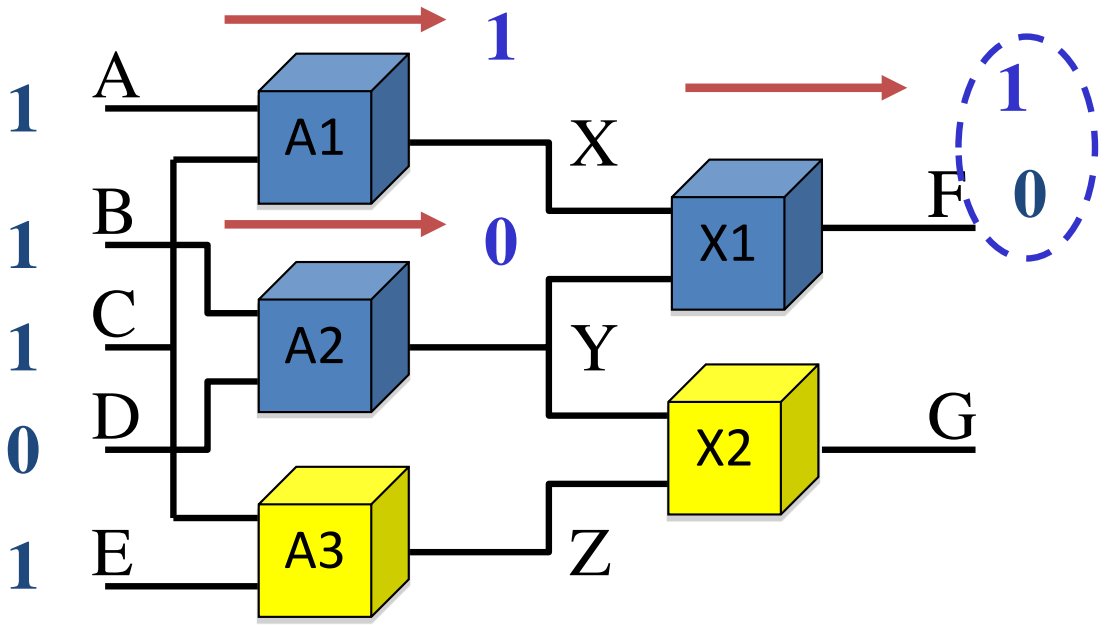
Cutoff:

- C(X,Y) is inconsistent.

Goal Test:

- Full assignment to X.
- $\exists\, Y \in D_Y\,.\, C(X,Y)$

Conflicts Indicate How to Remove Symptoms



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

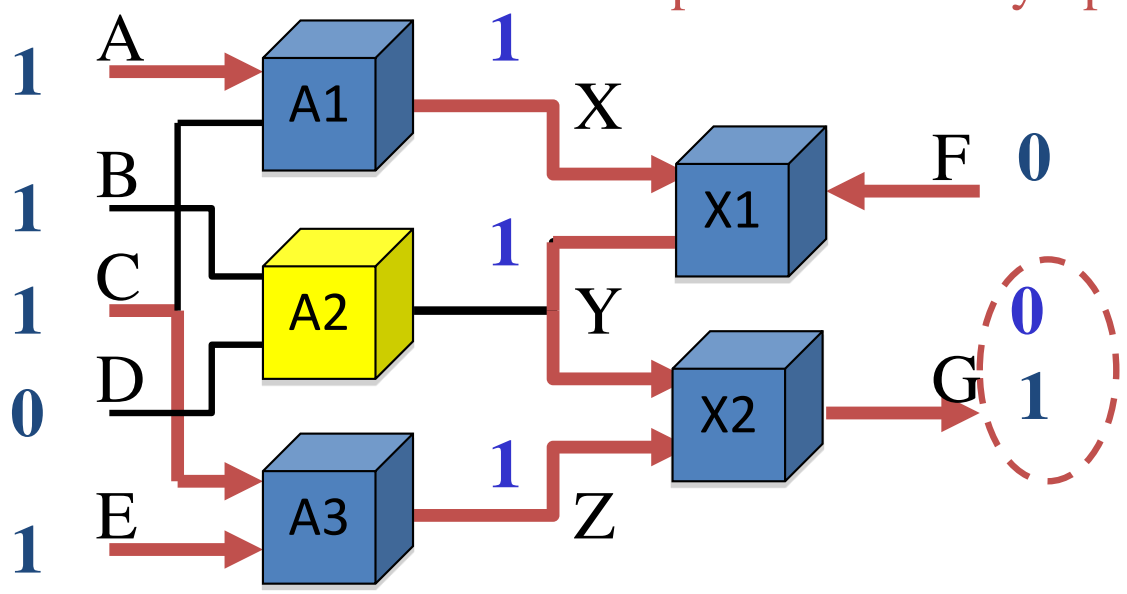
Conflict 1: Learn {A1=G, A2=G, X1=G} is inconsistent.

→ One of A1, A2 or X1 must be broken (*diagnosis of conflict*).

Conflict: Partial assignment to mode variables X, inconsistent with $\Phi \wedge O=o$.

Second Conflict

Conflicting modes aren't always upstream from symptom.



Symptom: G is observed 1, but predicted 0.

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

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

Generate Kernel Diagnoses by Set Covering

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

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

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

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

Kernel Diagnoses =

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

"Smallest" sets of modes that remove all conflicts.

Generate Kernel Diagnoses by Set Covering

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

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

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

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

Generate Kernel Diagnoses by Set Covering

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

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

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

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

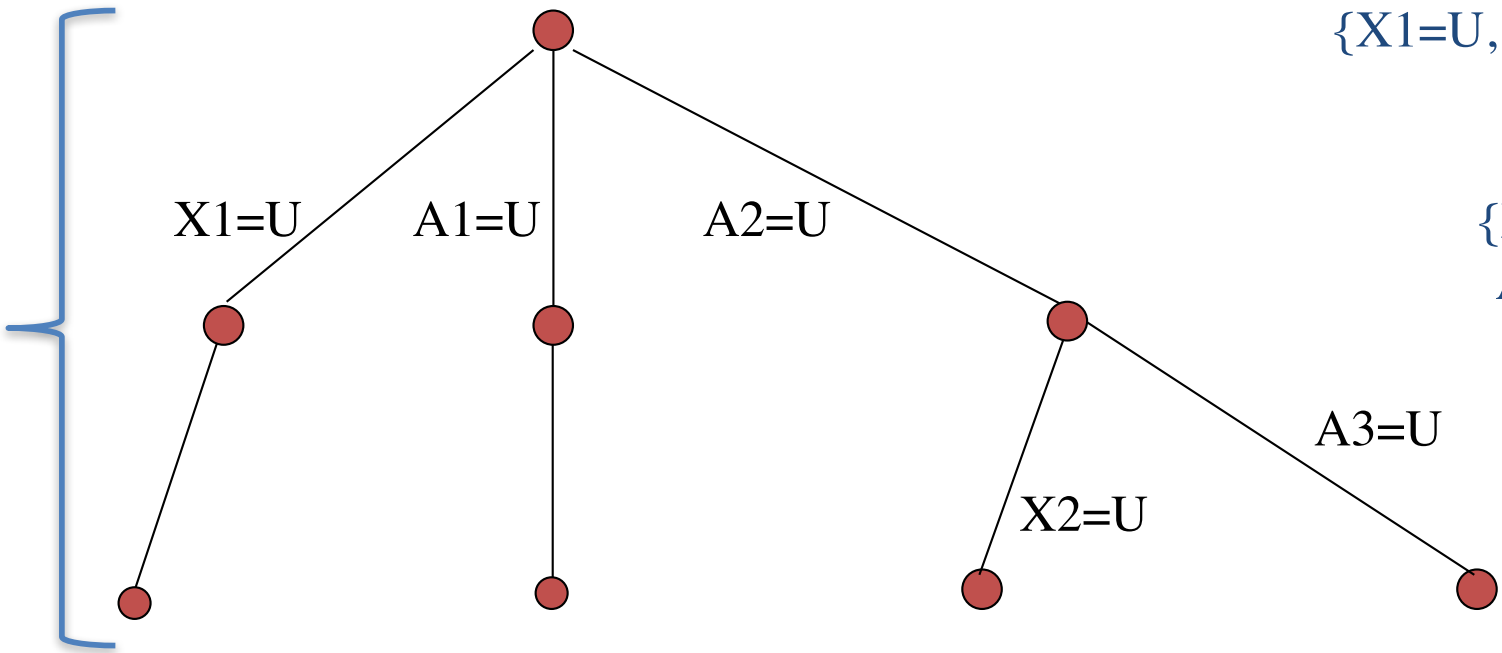
Kernel Diagnoses = ~~{A1=U, A3=U}~~
{A1=U}

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

"Smallest" sets of modes that remove all conflicts.

Kernel Generation Tree

Constituent Kernels



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

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

cover conflicts

X1=U

A1=U

A2=U

A3=U

X2=U

X1=U

A1=U

A2=U ∧ X2=U

A2=U ∧ A3=U

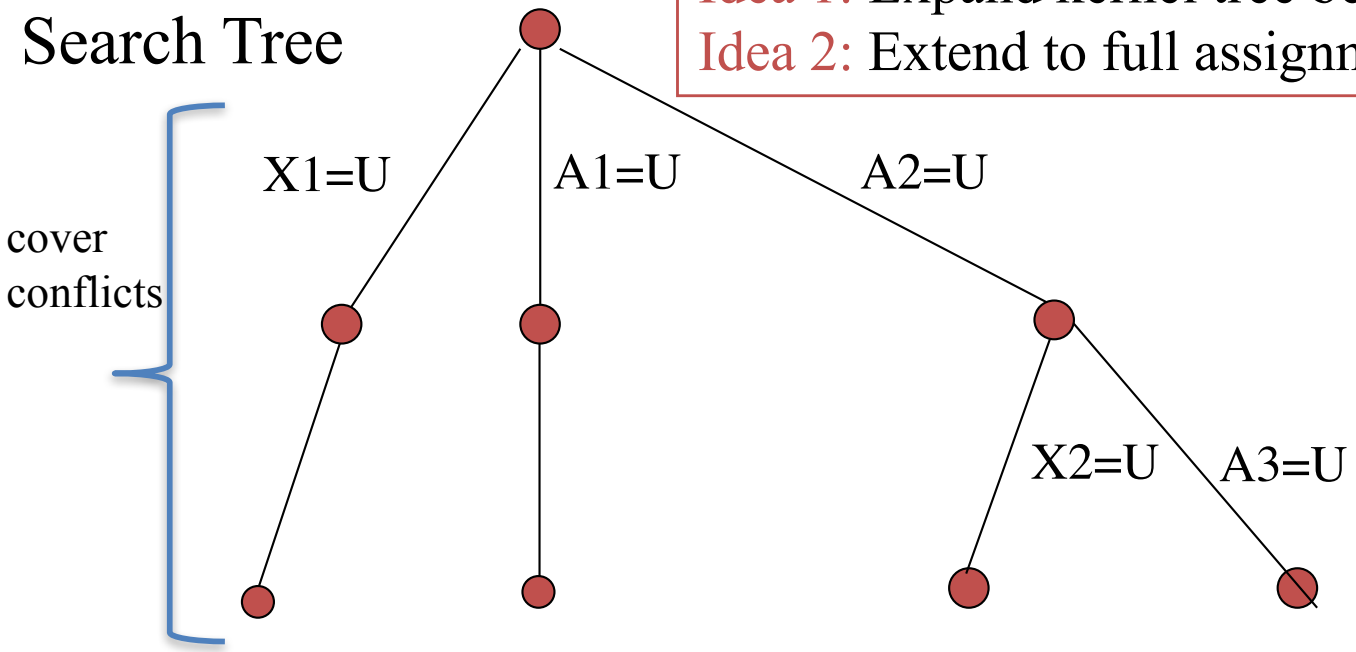
Conflict-directed A* Search Tree

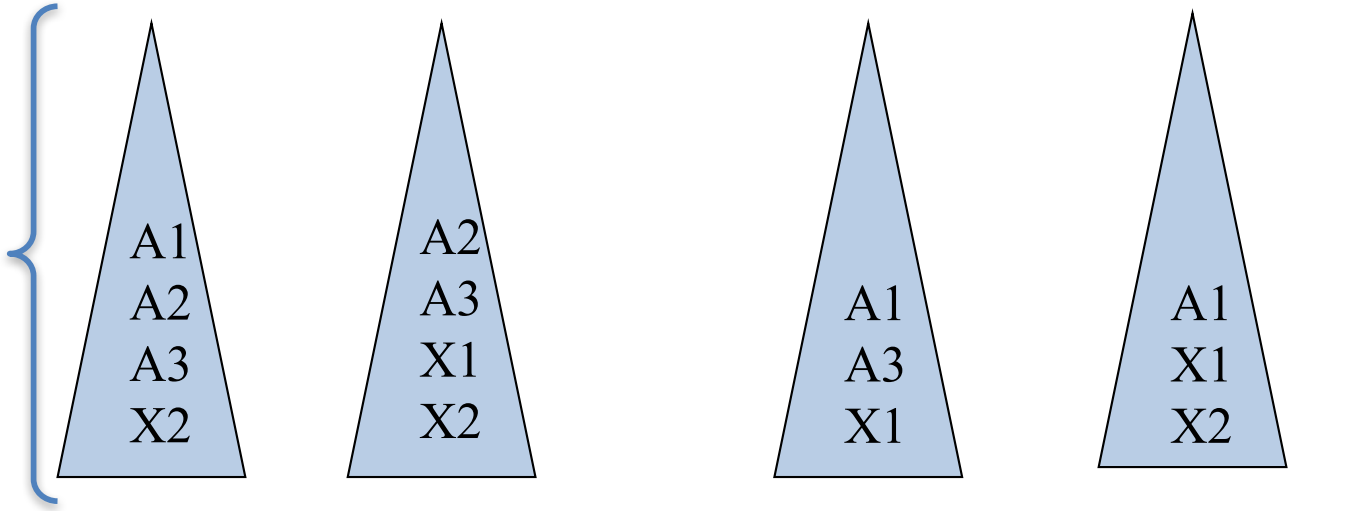
Idea 1: Expand kernel tree best-first.
Idea 2: Extend to full assignment.

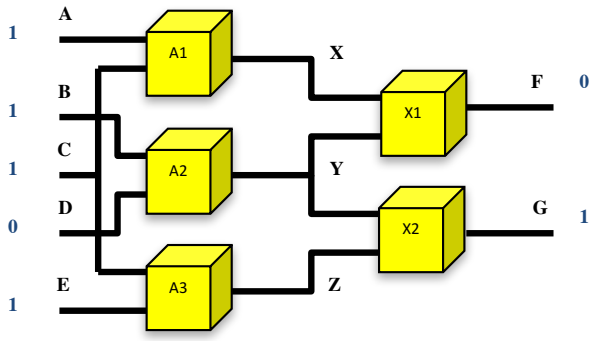
Constituent Kernels

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

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



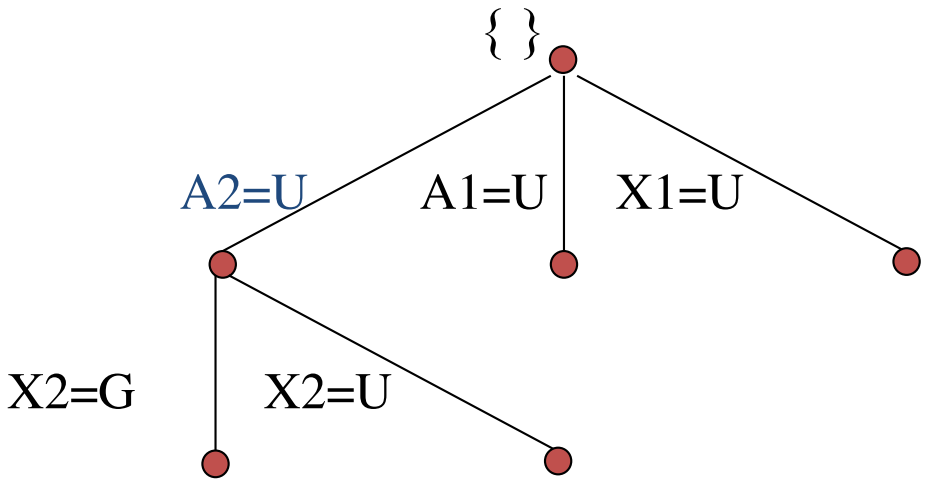




Conflict-directed A* Child Expansion

Conflict

$\neg$ (A2=G $\wedge$ A1=G $\wedge$ X1=G)



If Unresolved Conflicts:

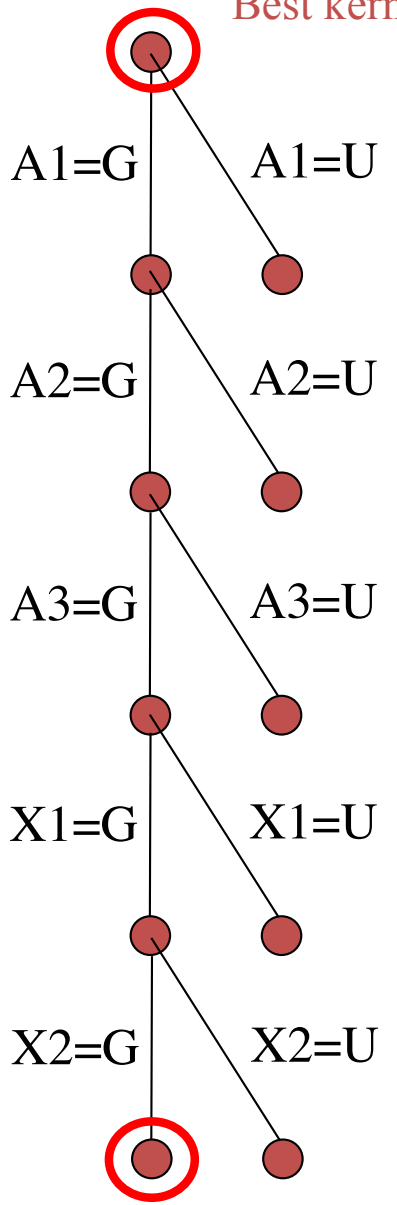
- Select unresolved conflict.
- Each child adds a conflict diagnosis (*constituent kernel*).

If All Conflicts Resolved:

- Select unassigned variable y_i.
- Each child adds an assignment from D_i.

1st Round:

cover conflicts



assign variables

Conflict Diagnoses

Ø

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

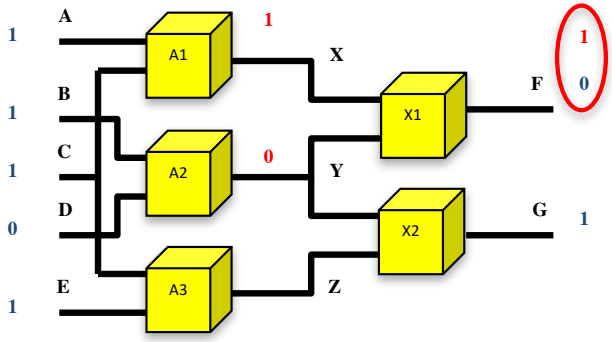
$P_{single} >> P_{double}$

$P_{A2=U} > P_{A1=U}$

$> P_{A3=U} > P_{X1=U}$

$> P_{X2=U}$

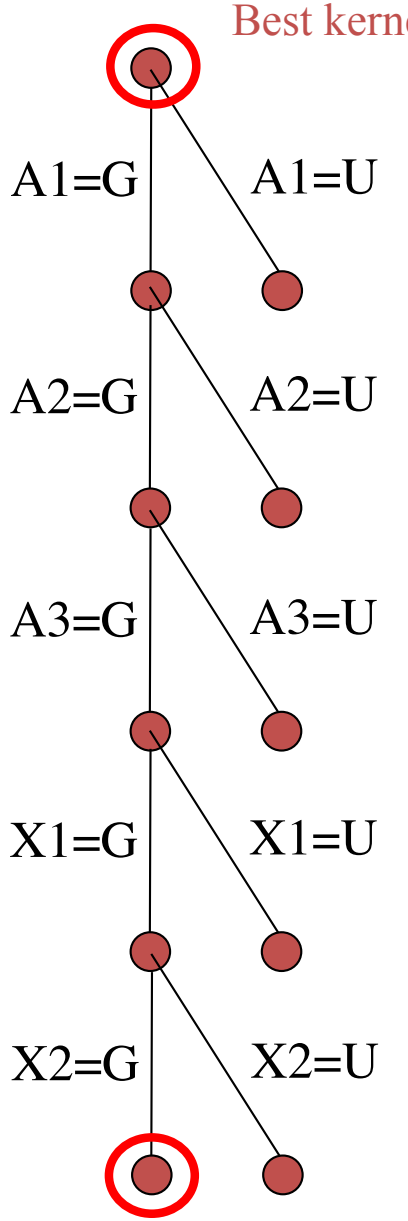
Test Candidate 1:



1st Round:

cover conflicts

assign variables



Best kernel

Best candidate

Conflict Diagnoses

∅

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

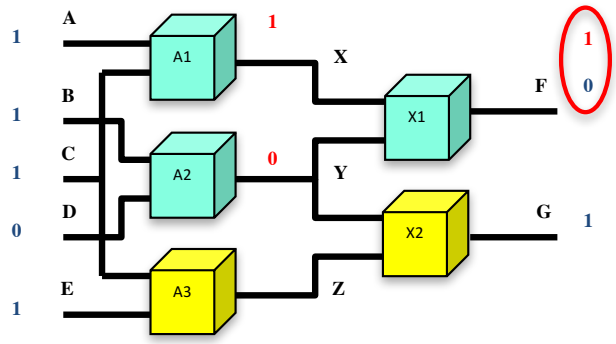
$P_{single} >> P_{double}$

$P_{A2=U} > P_{A1=U}$

$> P_{A3=U} > P_{X1=U}$

$> P_{X2=U}$

Test Candidate 1:



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

2nd Round:

cover conflicts

X1=U A1=U A2=U

Best kernel

A1=G A1=U

A3=G A3=U

X1=G X1=U

X2=G X2=U

assign variables

Best candidate

Conflict Diagnoses

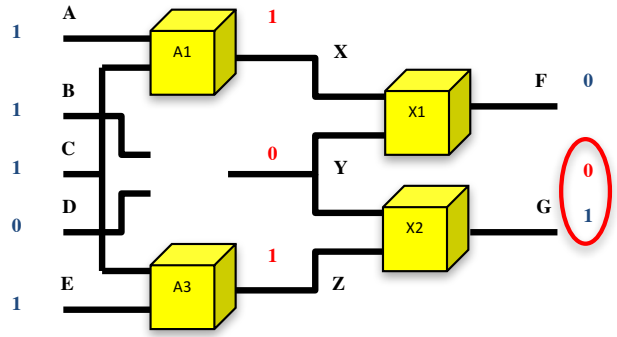
{X1=U, A1=U , A2=U}

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

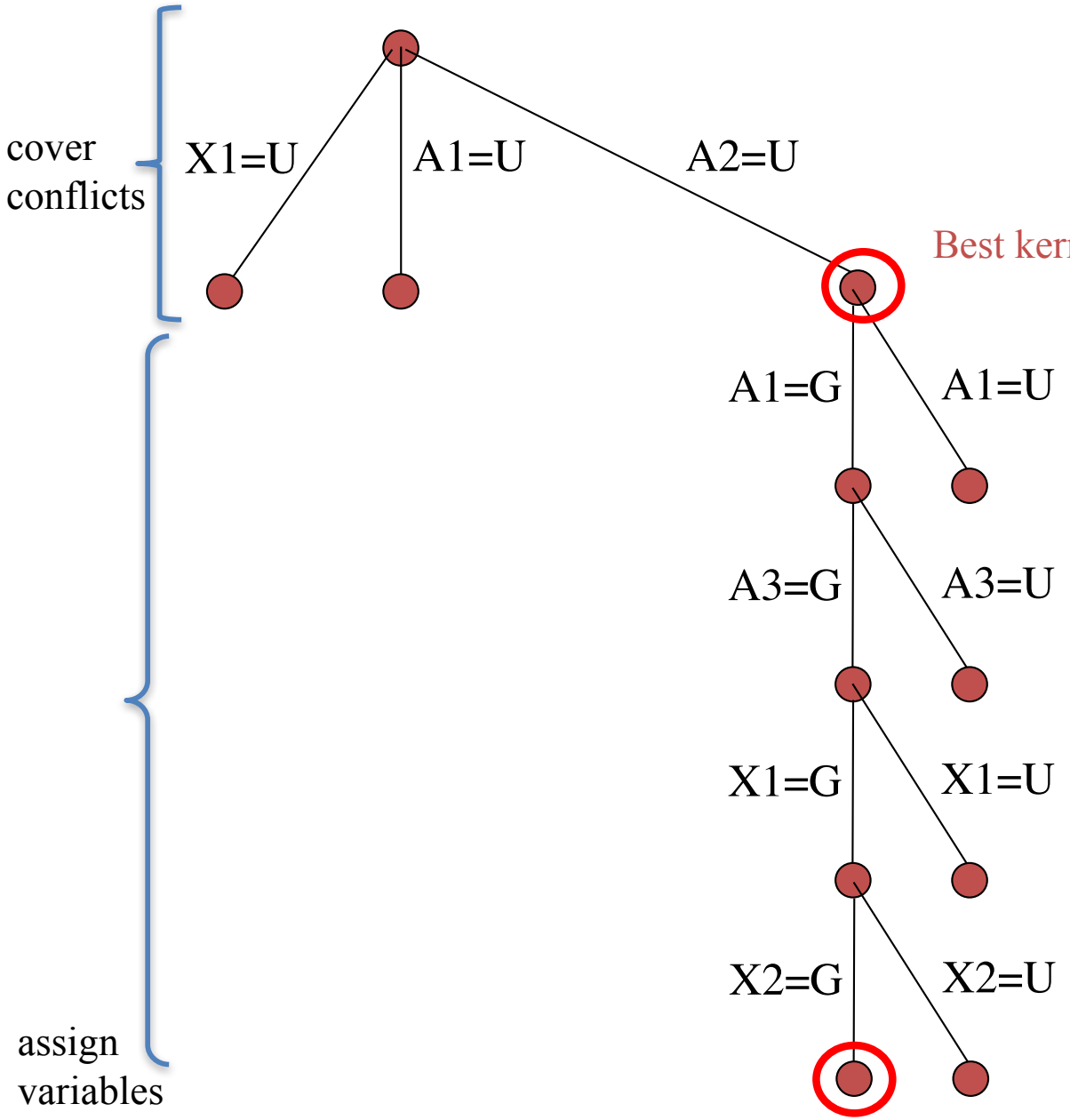
$P_{single} >> P_{double}$

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

Test Candidate 2:



2nd Round:



Conflict Diagnoses

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

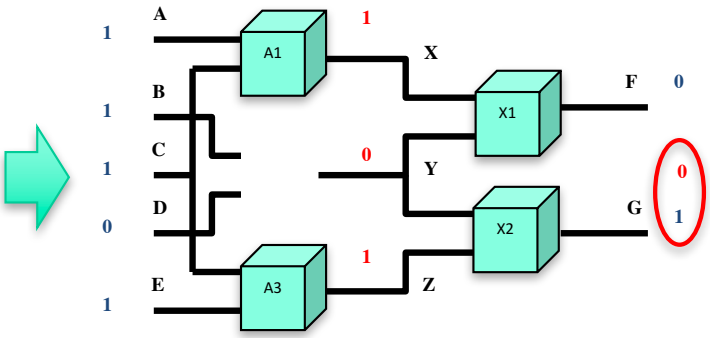
$P_{Xi=G} >> P_{Xi=U}$

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

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

Best kernel

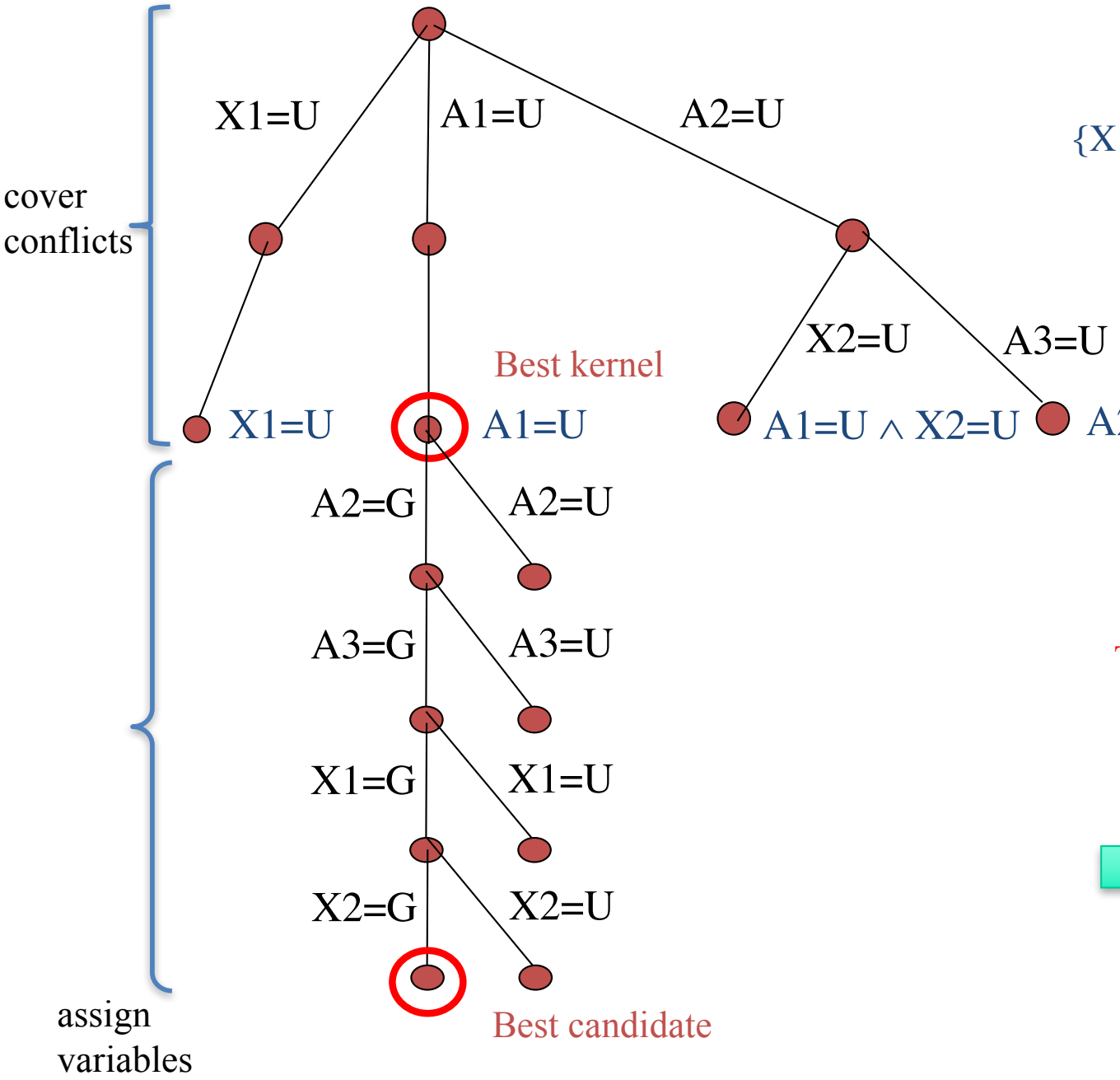
Test Candidate 2:



Best candidate

Conflict 2: {A1=G, A2=G, X1=G, X2=G}

3rd Round:



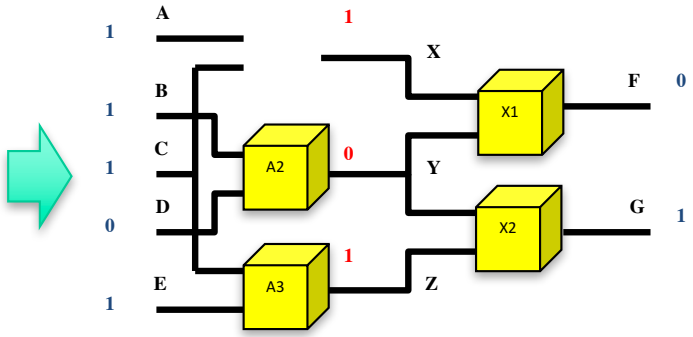
Conflict Diagnoses

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

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

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

Test Candidate 3:

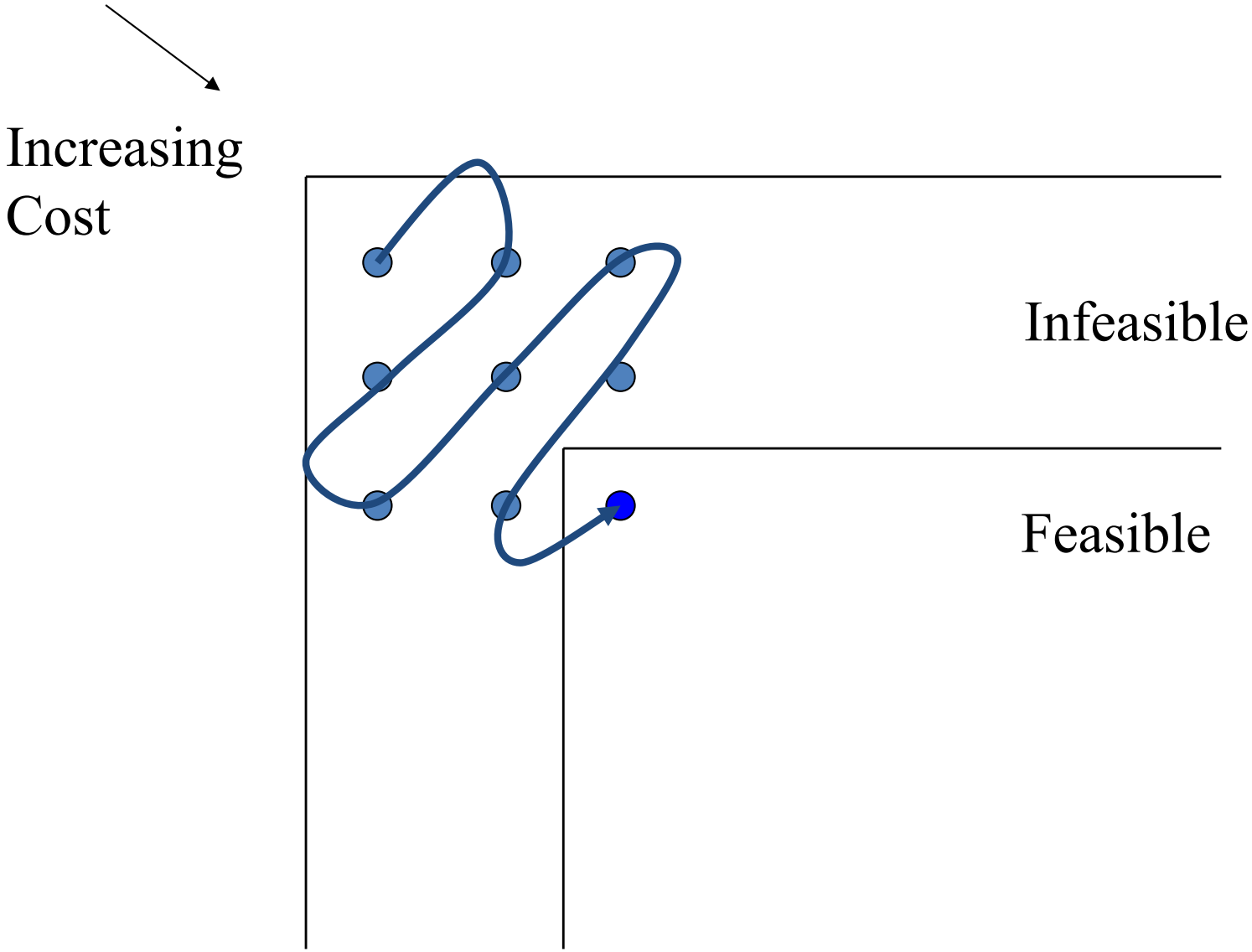


Consistent!

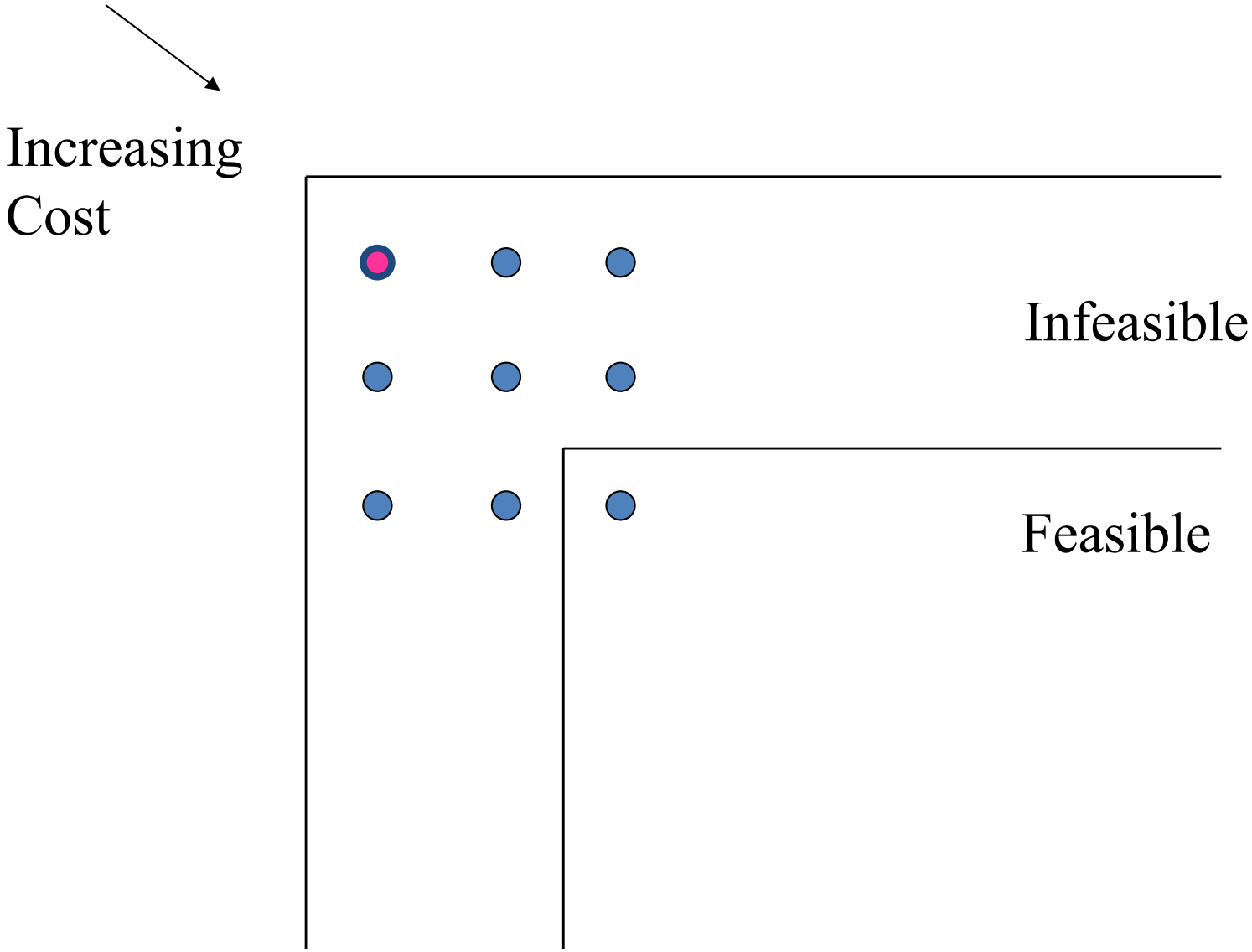
- All conflicts discovered.

⇒ All remaining candidates are diagnoses.

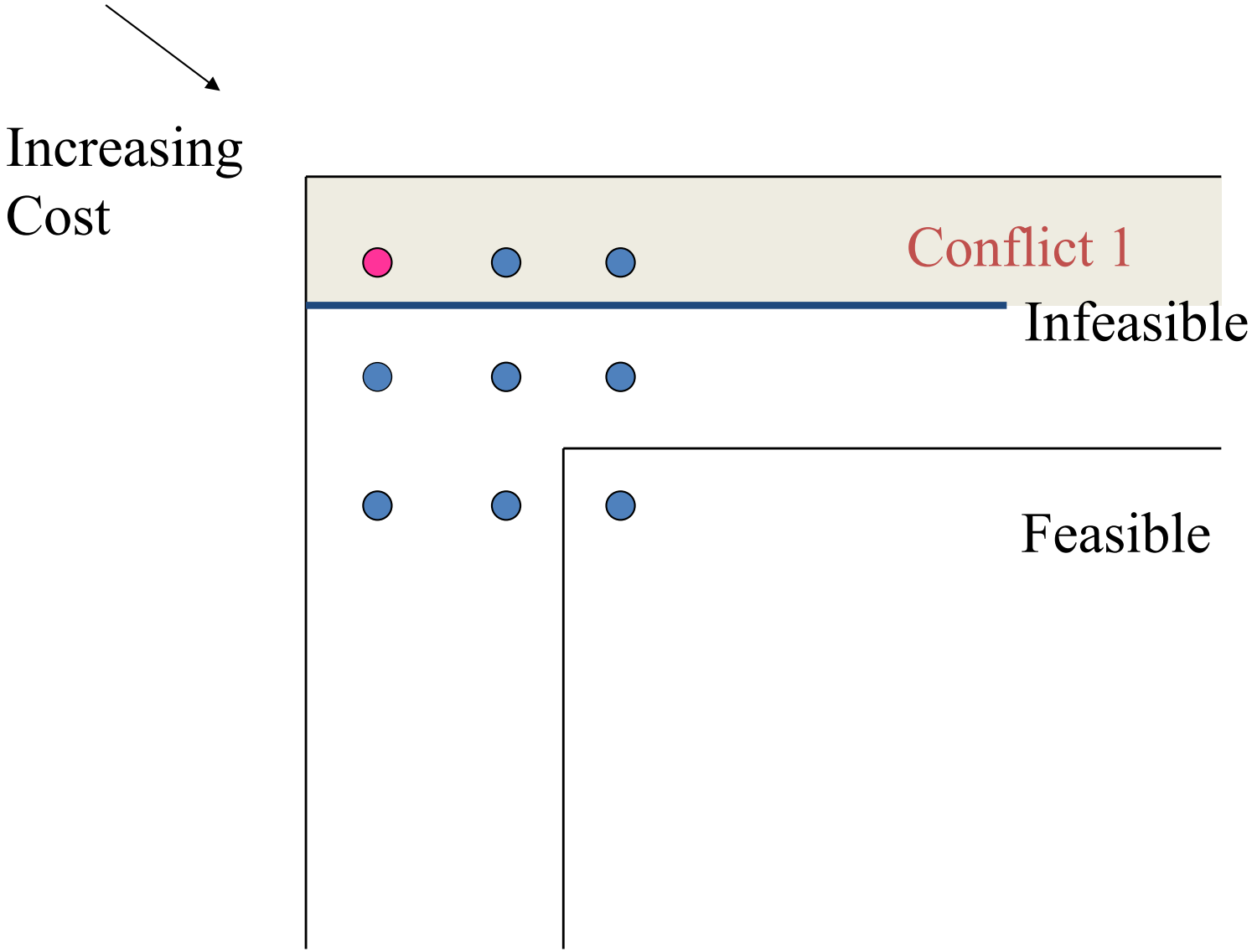
A*



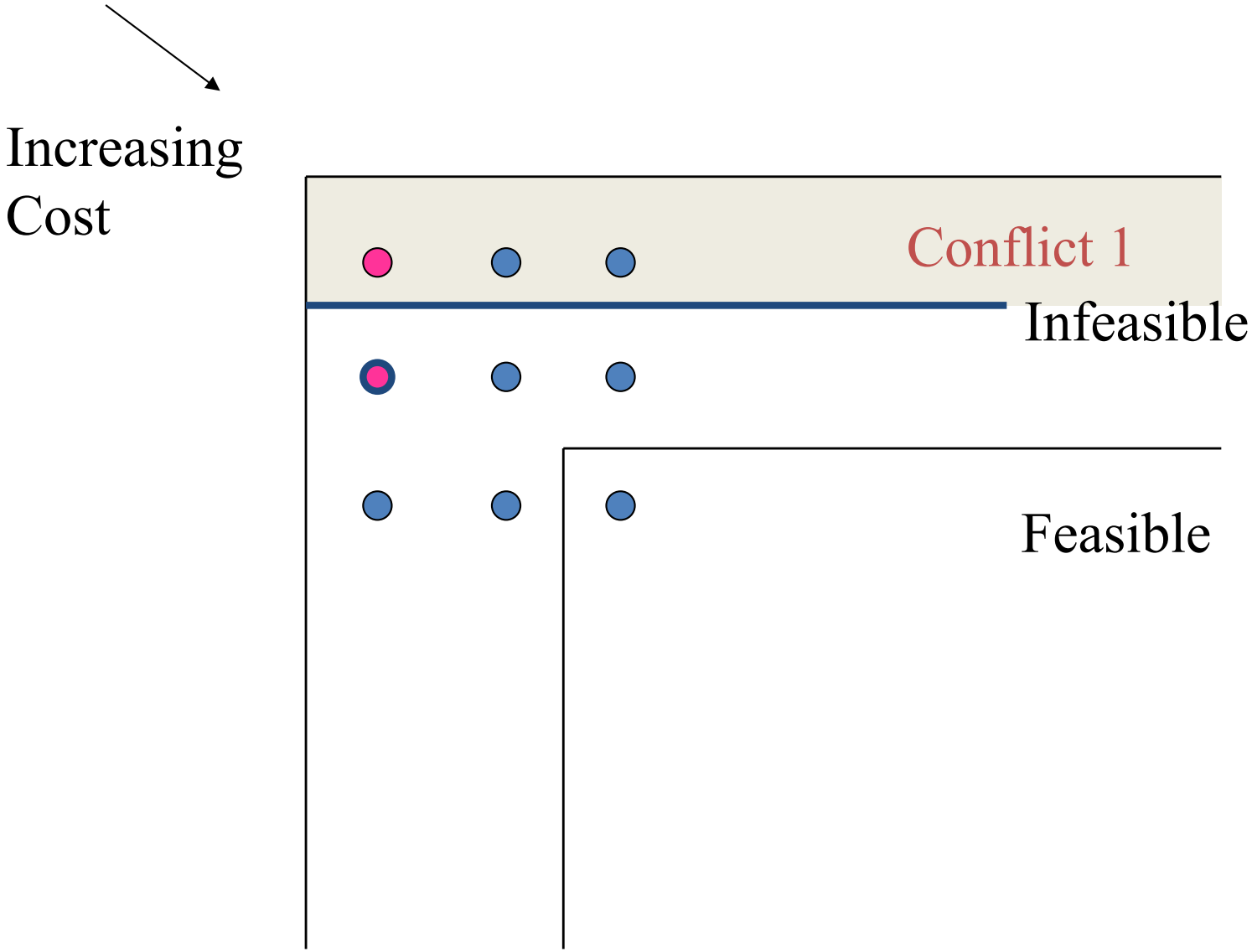
Conflict-directed A*



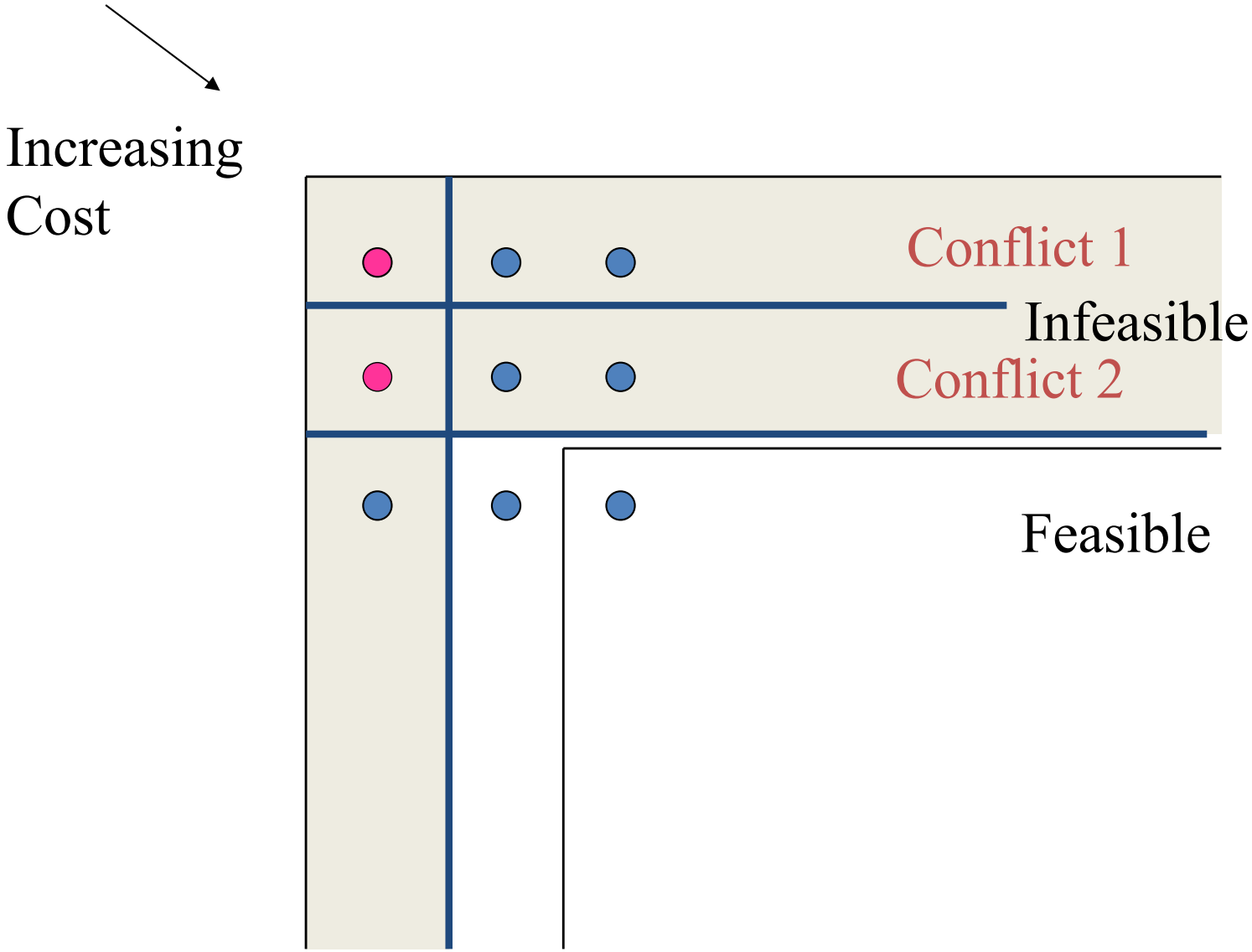
Conflict-directed A*



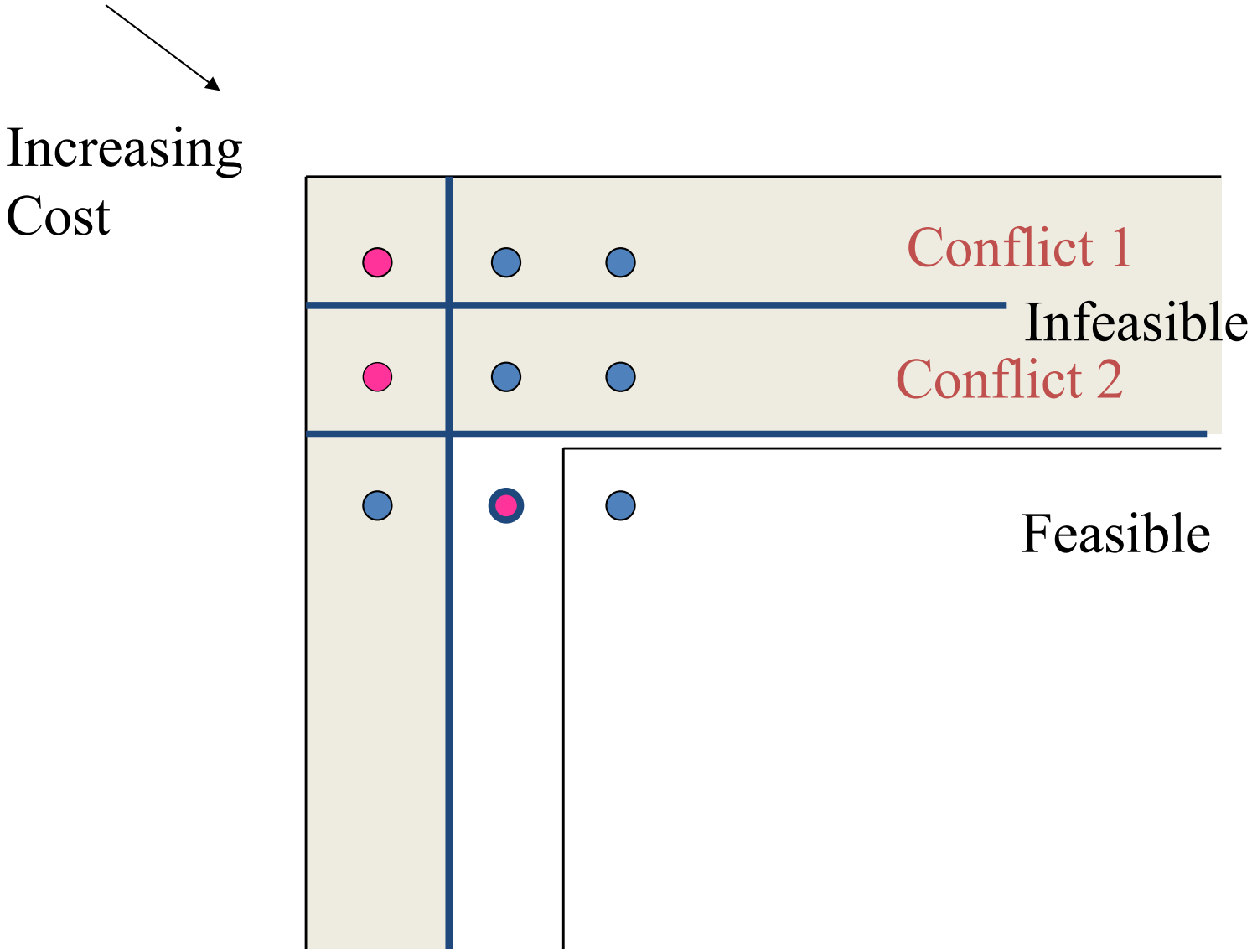
Conflict-directed A*



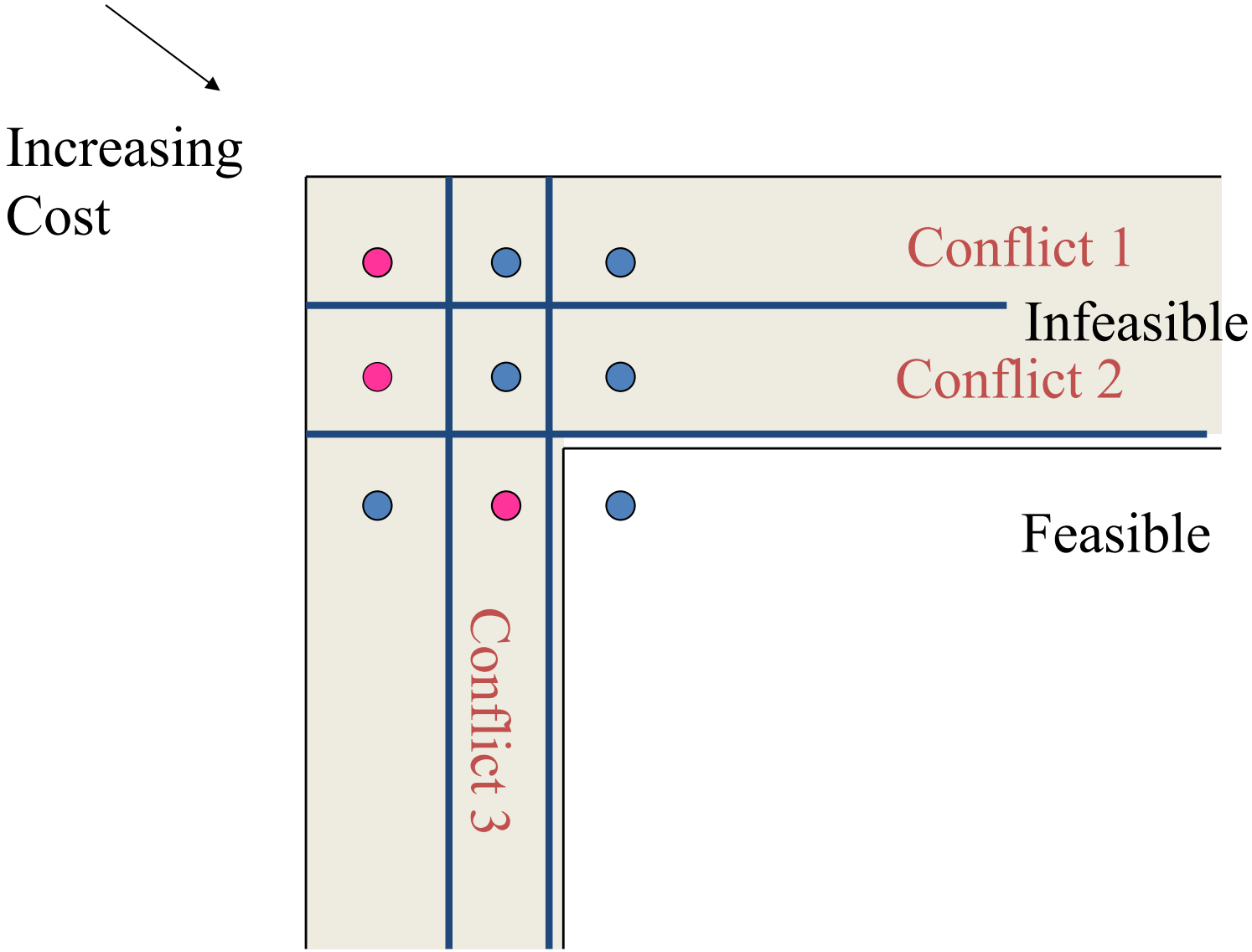
Conflict-directed A*



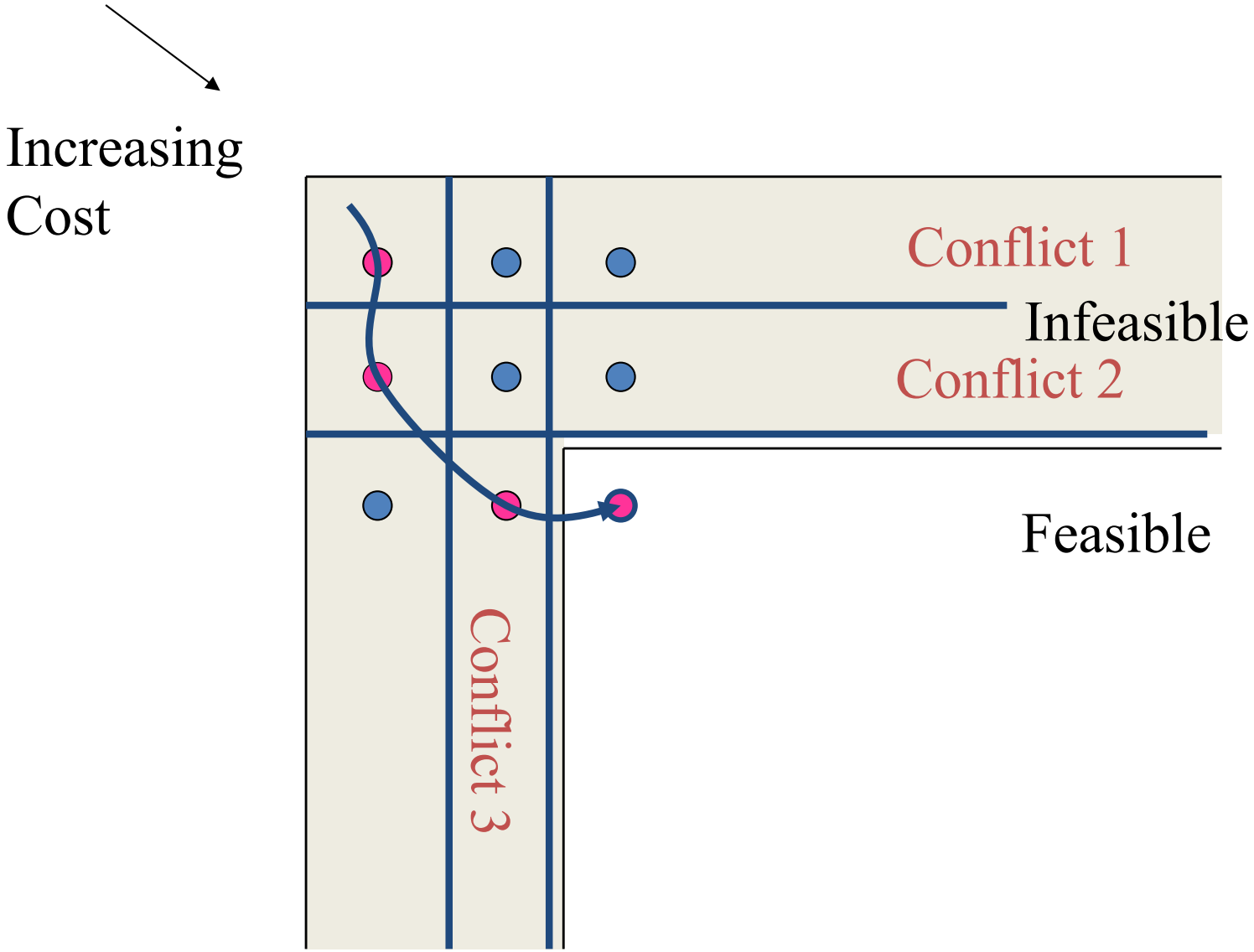
Conflict-directed A*



Conflict-directed A*



Conflict-directed A*

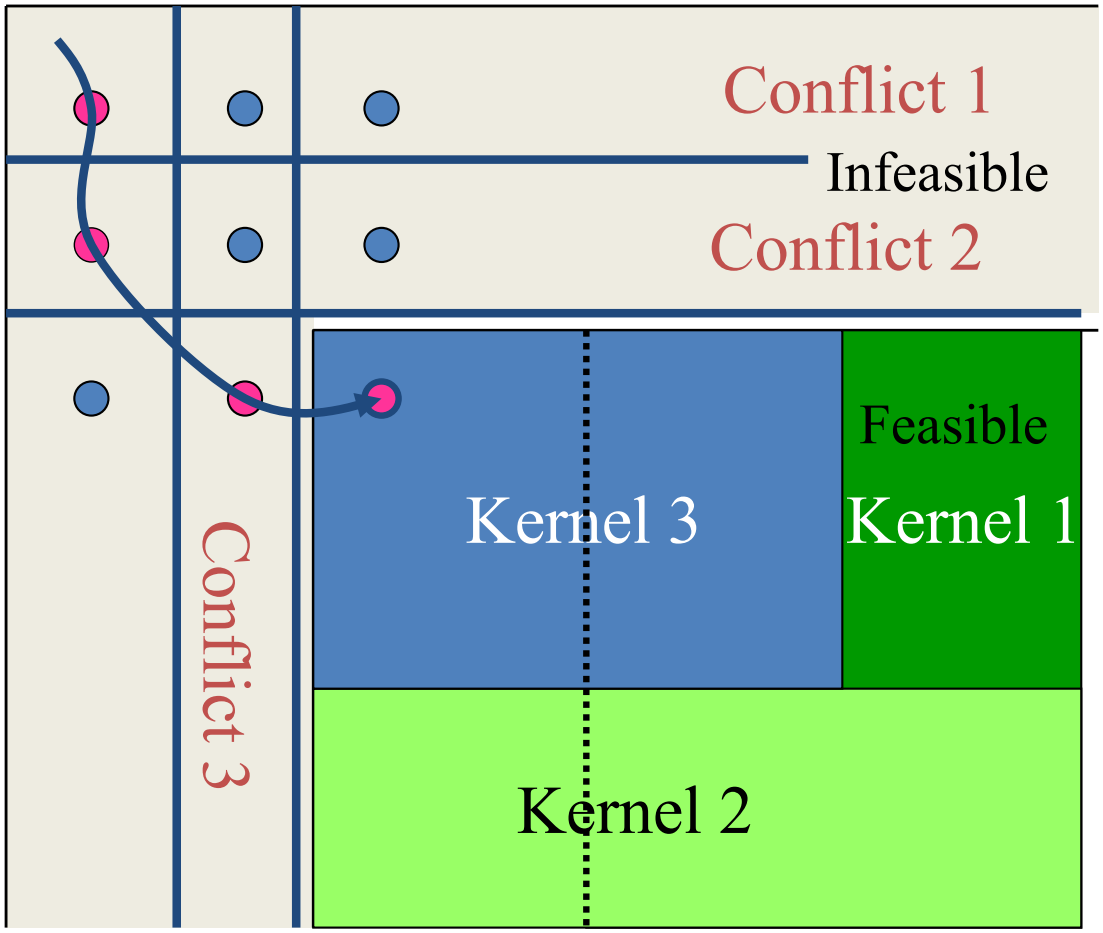


Conflict-directed A*

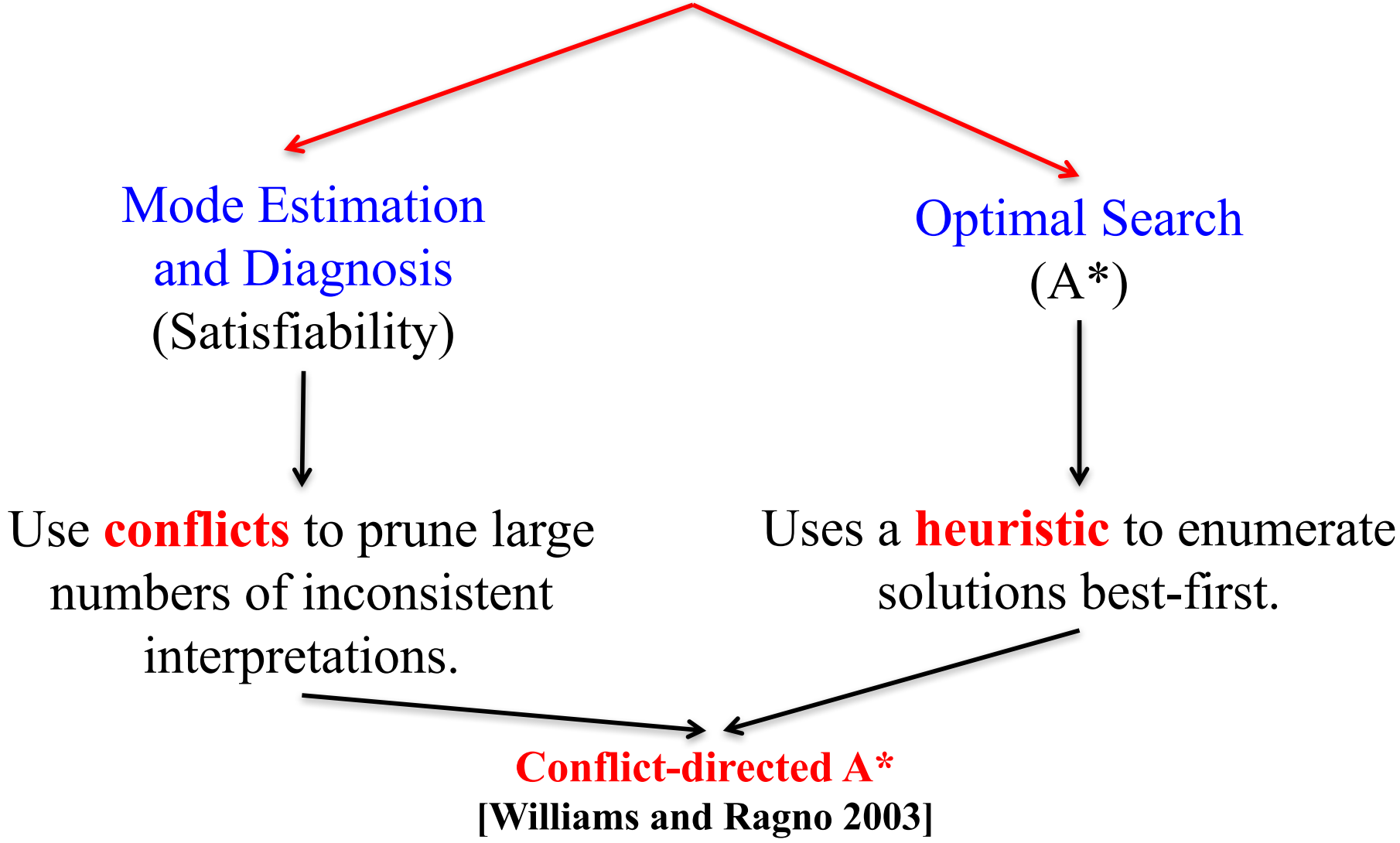
- Each feasible subregion described by a kernel assignment.

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

Increasing Cost

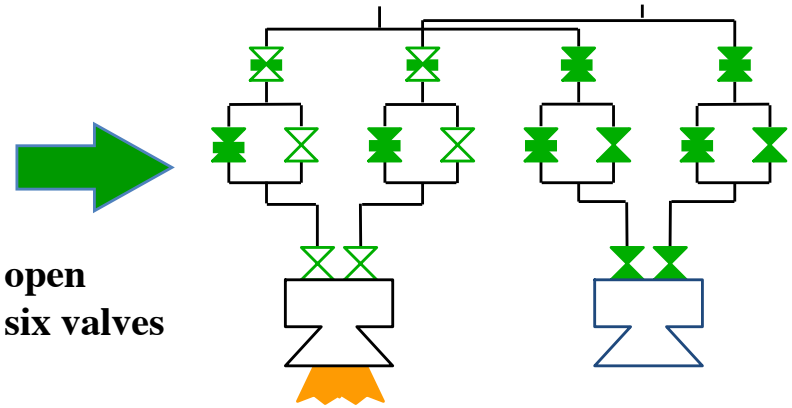


Enumerating Likely Mode Estimates



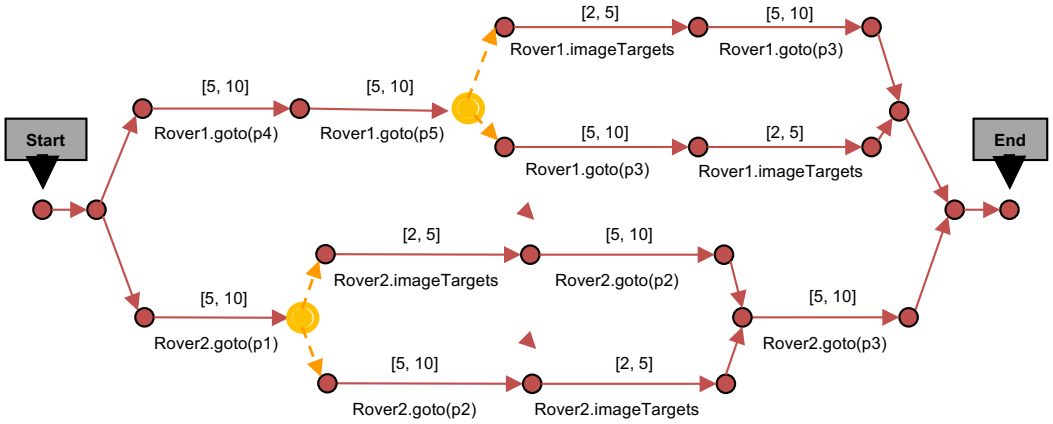
Use OpSat to . . .

Reconfigure Modes



Livingstone
[Williams & Nayak, AAAI 96]

Execute Programs with Choice

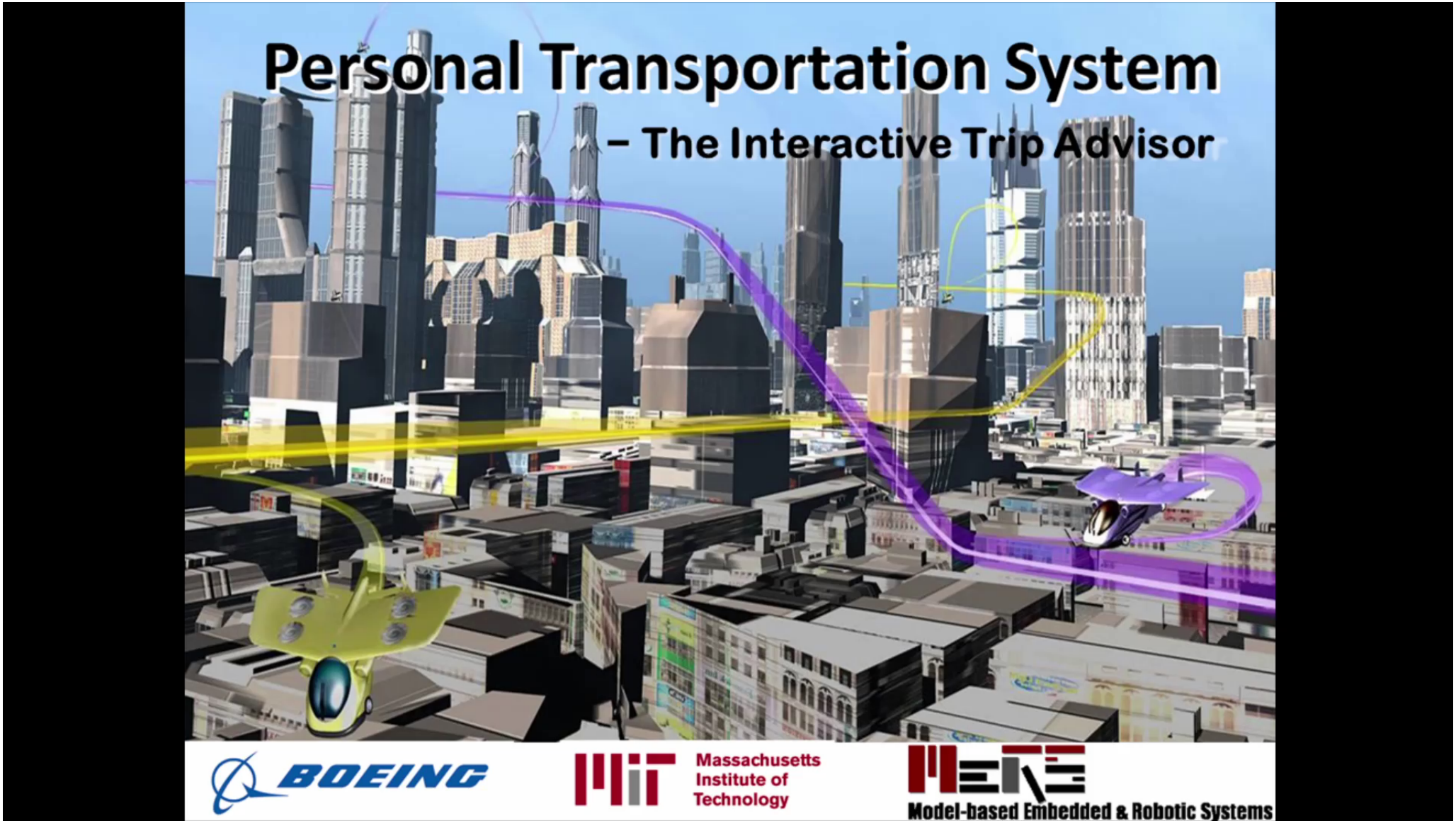


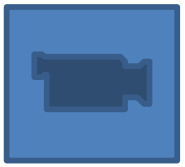
Kirk
[Kim, Williams & Abramson, IJCAI 01]

Outline

- Programs that Monitor State
  - Sub-goal monitoring
  - Model-based diagnosis and mode estimation

- Programs that Self-Diagnose (opt)


Personal Transportation System
– The Interactive Trip Advisor
BOEING
Massachusetts
Institute of
Technology
Model-based Embedded & Robotic Systems



The Problem

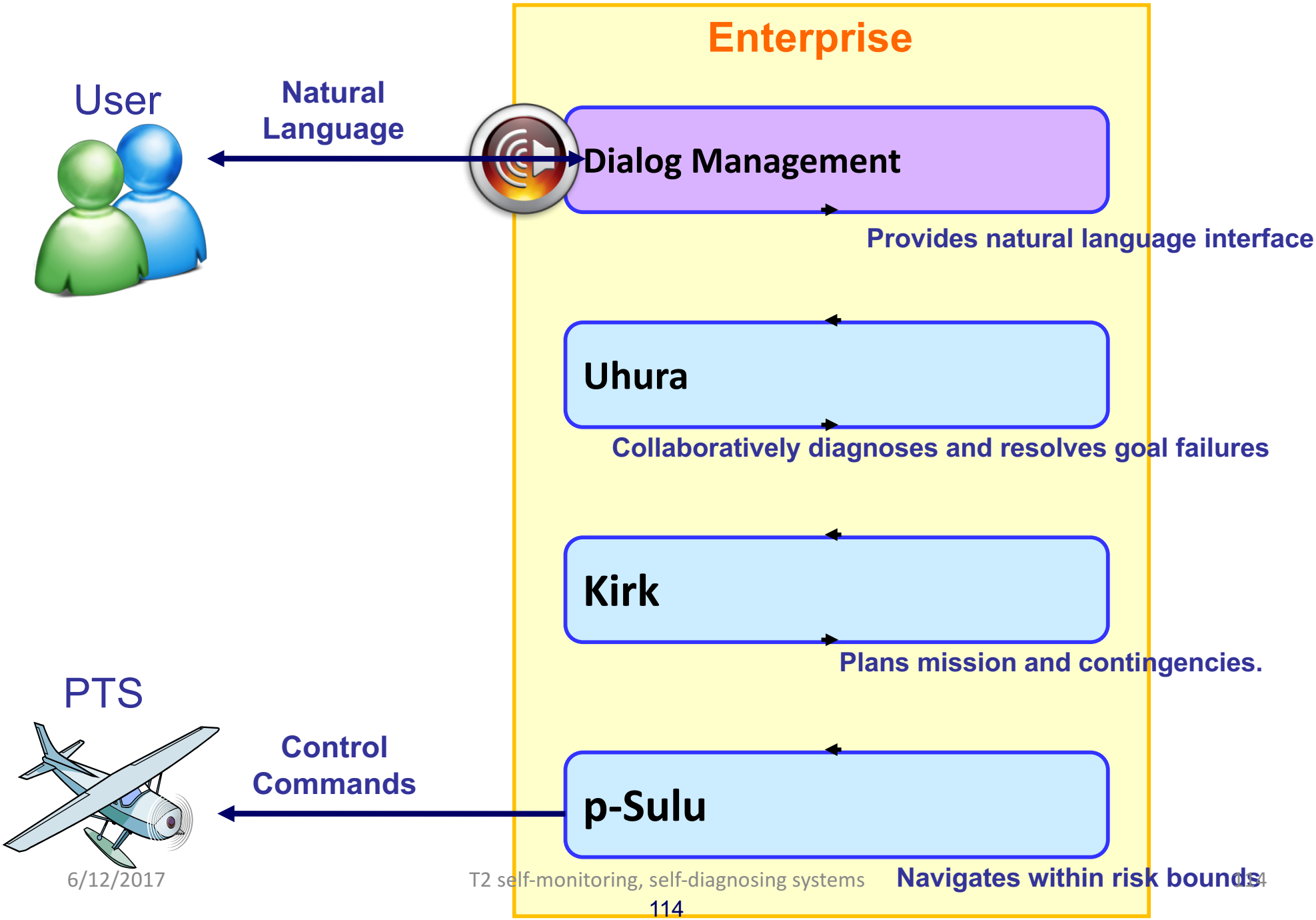
- As humans, we often plan to do too much. . .
- . . . our study plans, working schedules, and travel itineraries are often over-subscribed.
  - 'I want to see the new movie premiere tonight!'
  - 'Sorry, you cannot. You need submit your project by **mid night**, and it will keep you busy until **11pm**. '

Approach

View human robot collaboration as a form of collaborative diagnosis.

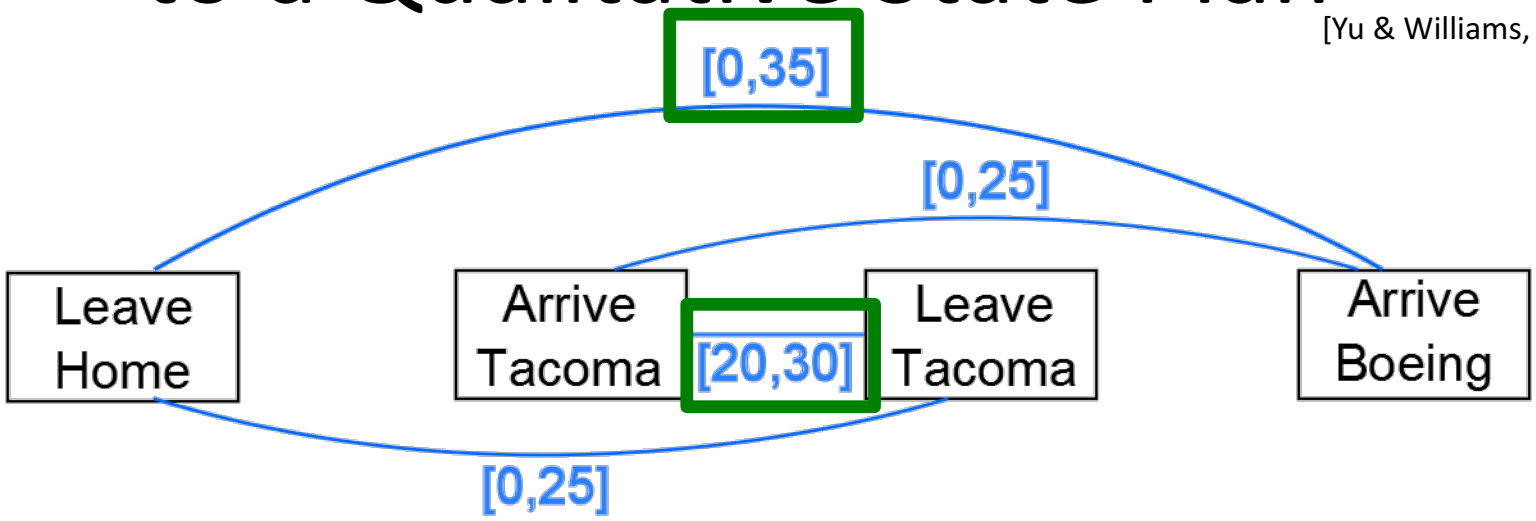
1. Identify user goals that are over-subscribed, within a temporal plan (QSP).
2. Explain their source of inconsistency, using conflicts.
3. Use constraint suspension and kernels to suggest goals to drop.

- Use continuous relaxation to weaken rather than drop user goals.
- Generalize OpSat to disjunctive linear programs.


Enterprise
User
Natural Language
Dialog Management
Provides natural language interface
Uhura
Collaboratively diagnoses and resolves goal failures
Kirk
Plans mission and contingencies.
PTS
Control Commands
p-Sulu
Navigates within risk bounds

Uhura Translates User Goals to a Qualitative State Plan

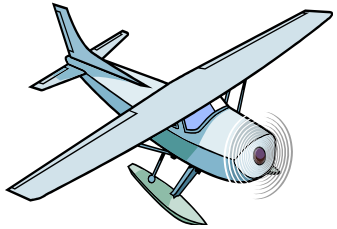
[Yu & Williams, IJCAI 13]



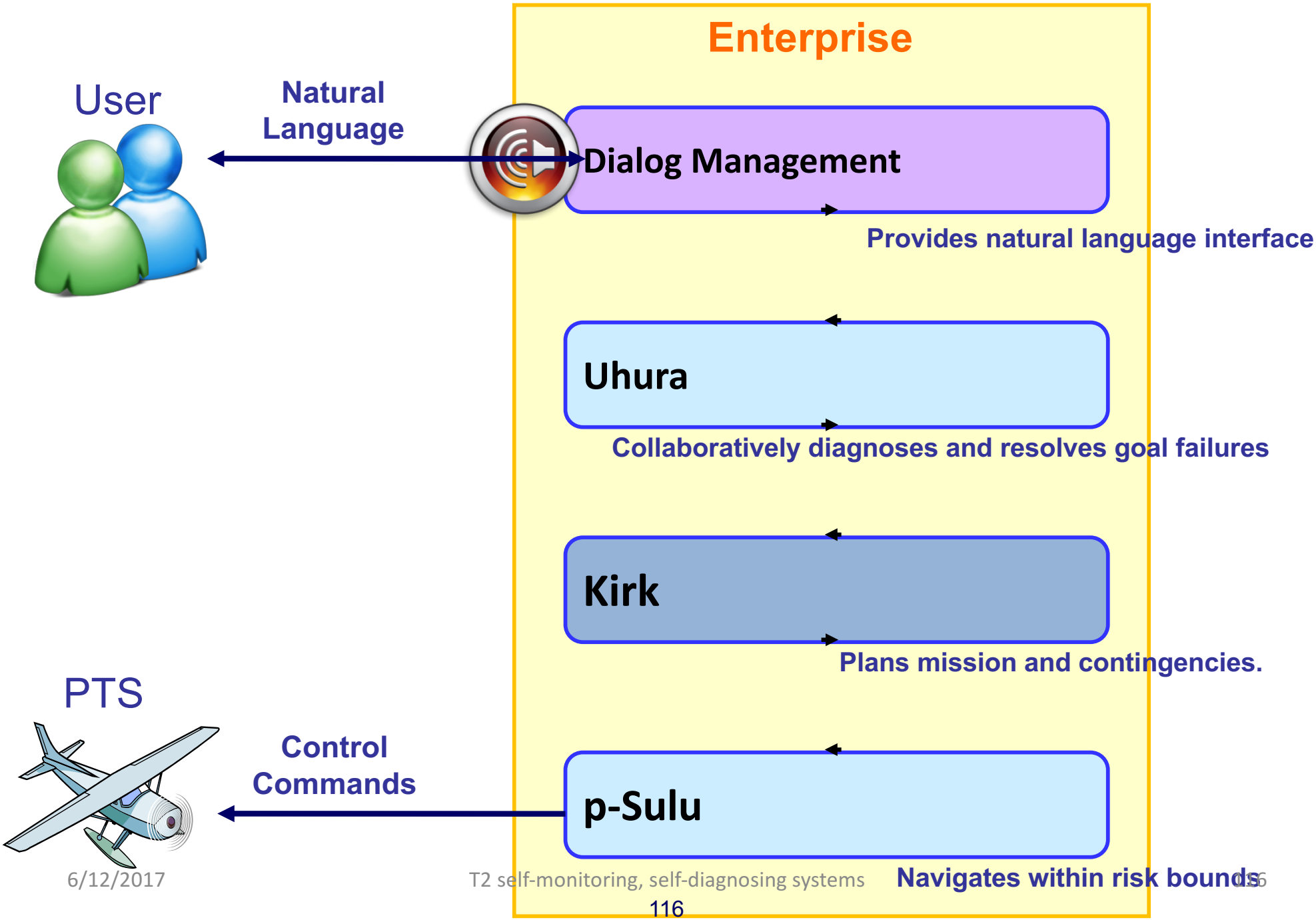
I want to go to Boeing
in 35 minutes.
I want to stop at Tacoma airport
for 20 to 30 minutes.
I've applied for a slot at Tacoma.
That's all.

User

PAV

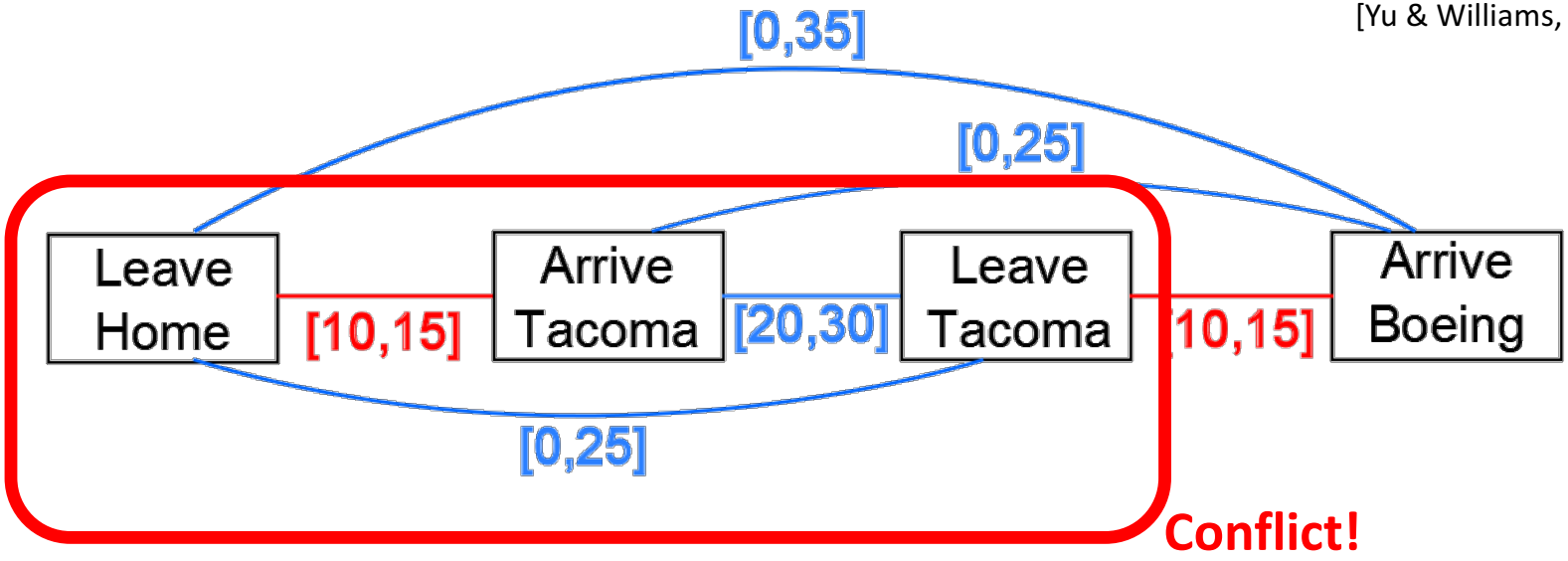


Kirk searches for an executable travel plan



Uhura Collaborates with User to Diagnose and Repair Plan Failure

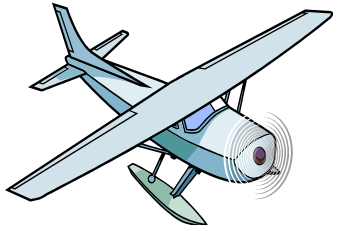
[Yu & Williams, IJCAI 13]



User

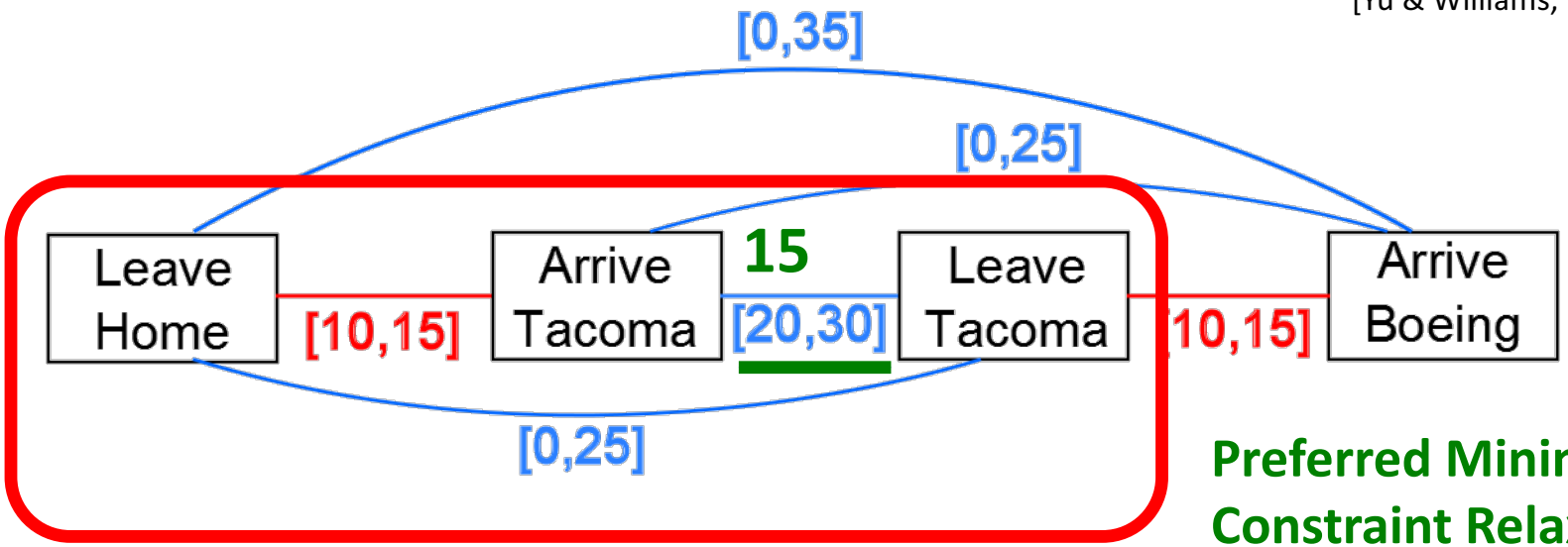
I have a problem, navigating around the storm will delay us.

PAV



Uhura Collaborates with User to Diagnose and Repair Plan Failure

[Yu & Williams, IJCAI 13]

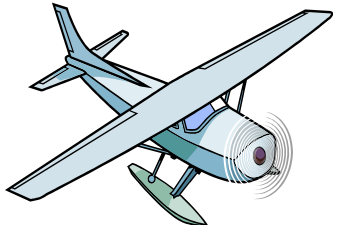


Preferred Minimal Constraint Relaxation

Can you shorten your stay at Tacoma to 15 minutes.

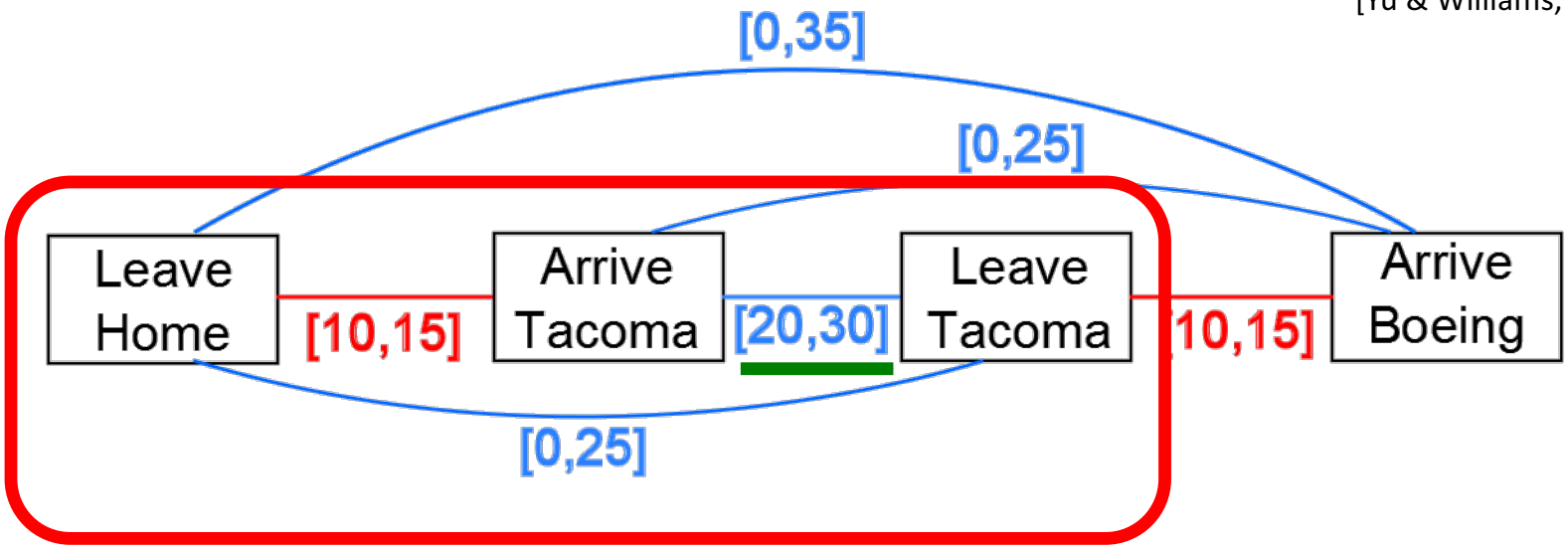
User

PAV



Uhura Collaborates with User to Diagnose and Repair Plan Failure

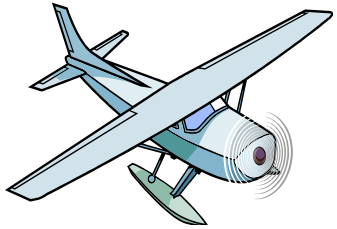
[Yu & Williams, IJCAI 13]



No, I cannot do that.

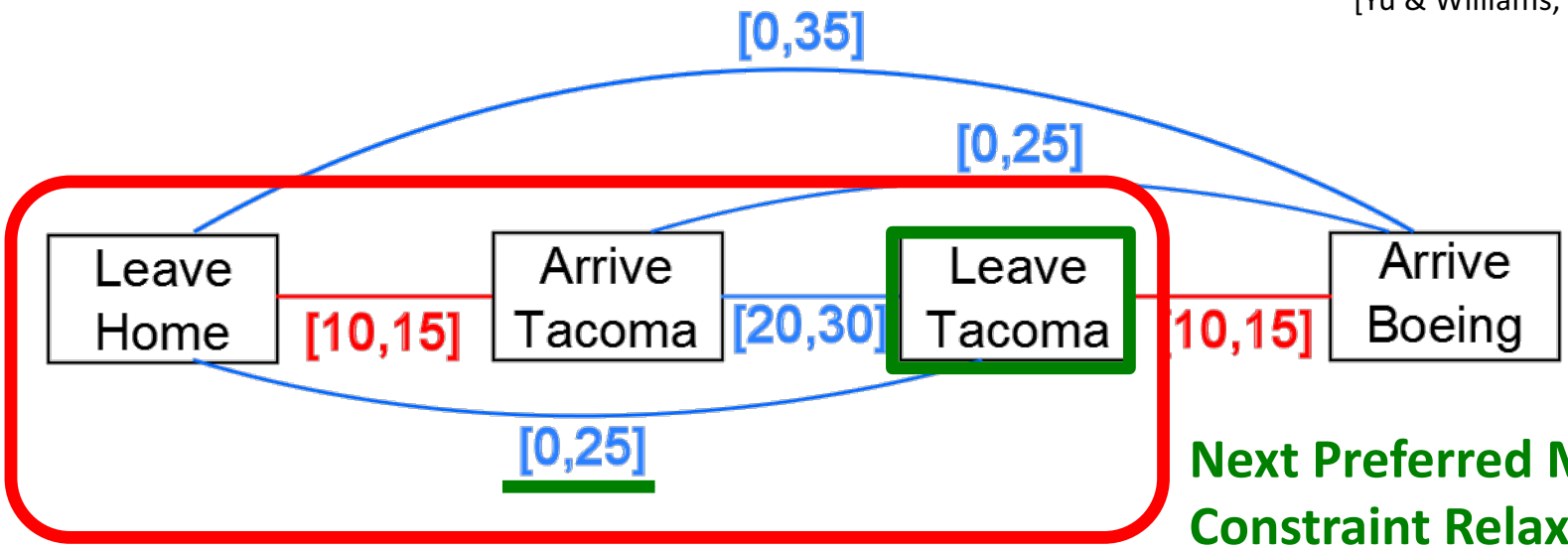
User

PAV



Uhura Collaborates with User to Diagnose and Repair Plan Failure

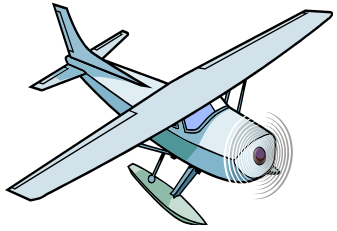
[Yu & Williams, IJCAI 13]



Next Preferred Minimal Constraint Relaxation

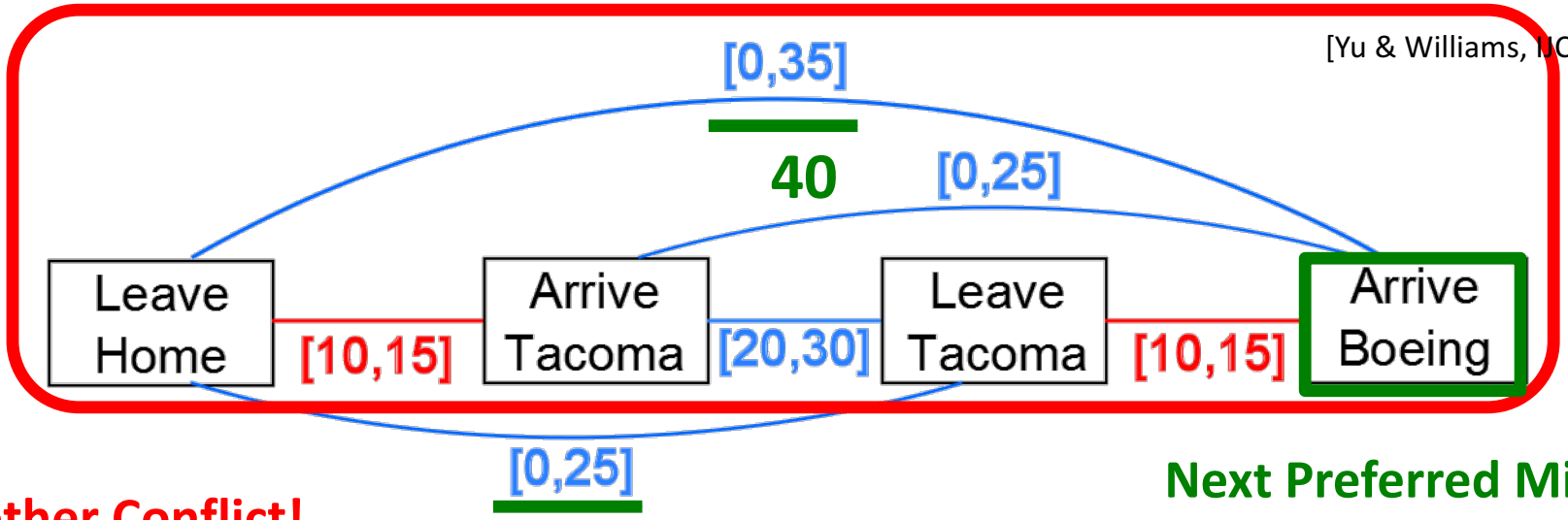
User

PAV



Uhura Collaborates with User to Diagnose and Repair Plan Failure

[Yu & Williams, IJCAI 13]

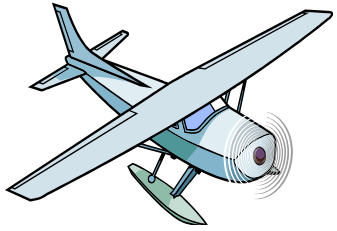


Another Conflict!

Next Preferred Minimal Constraint Relaxation

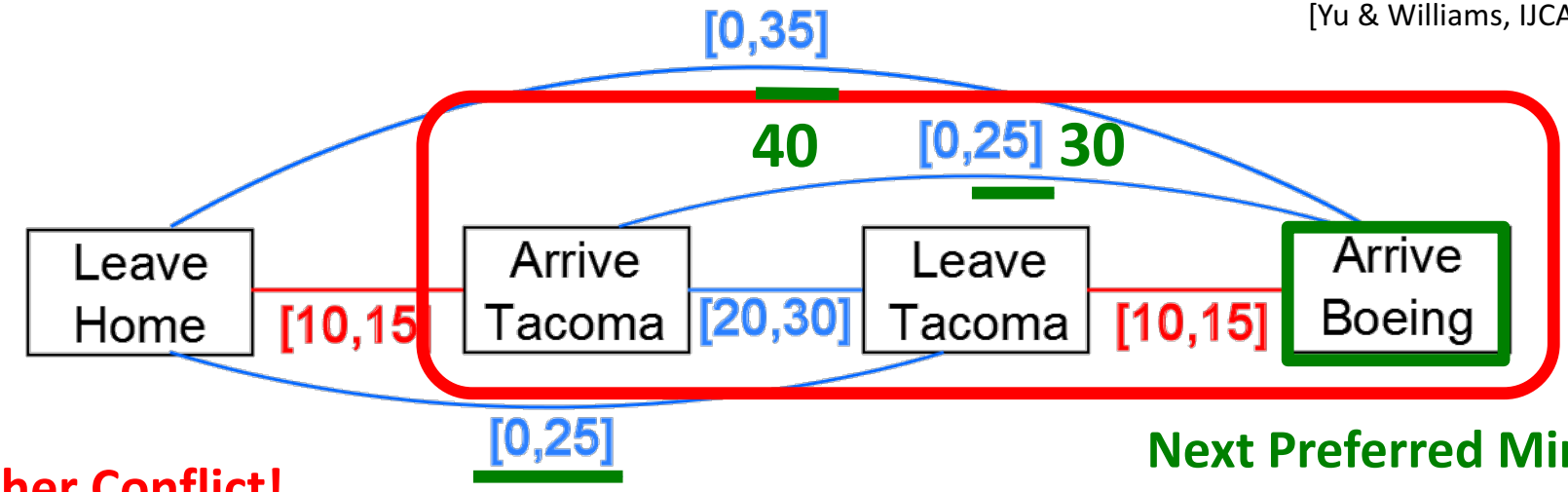
User

PAV



Uhura Collaborates with User to Diagnose and Repair Plan Failure

[Yu & Williams, IJCAI 13]



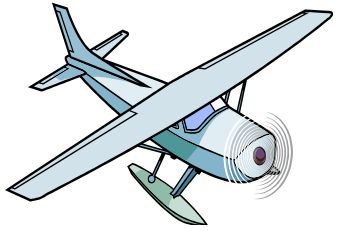
Another Conflict!

Next Preferred Minimal Constraint Relaxation

Can you delay your take off slot at Tacoma by 5 minutes and your arrival slot at Boeing by 5 minutes? I cannot fly too fast.

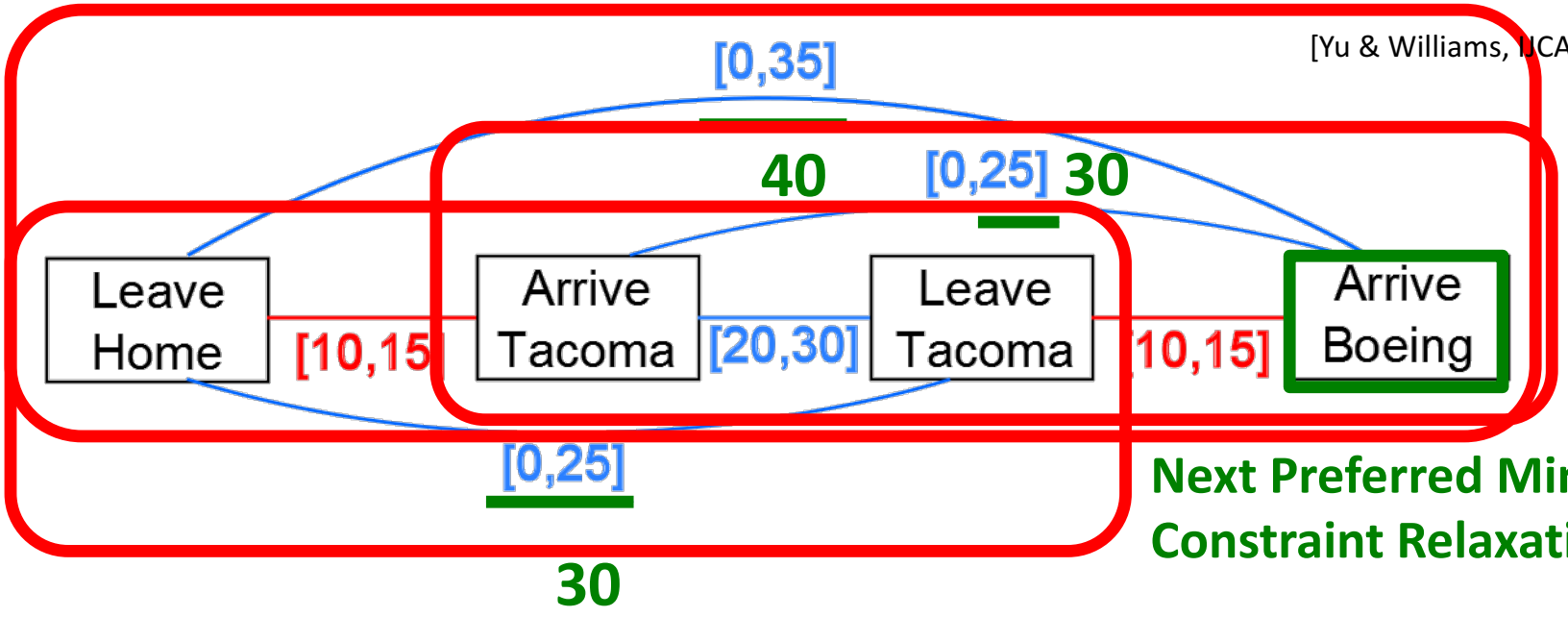
User

PAV



Uhura Collaborates with User to Diagnose and Repair Plan Failure

[Yu & Williams, IJCAI 13]

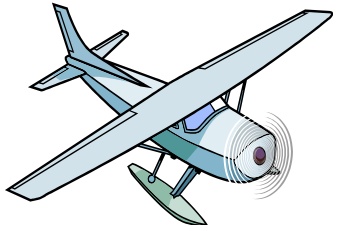


Next Preferred Minimal Constraint Relaxation

Yes, that's fine.

User

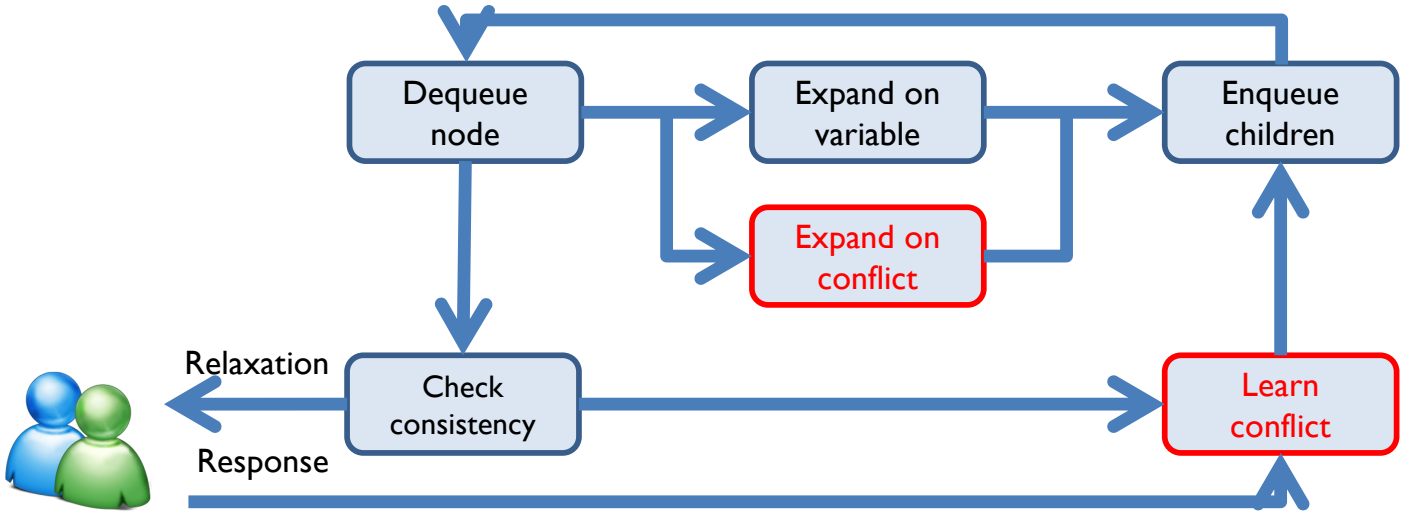
PAV



Best-first, Conflict-directed Relaxation

- **Generate**:
  - Enumerates minimal relaxations in best-first order.
  - Generalizes conflict-directed A* to continuous conflicts.
  - Framed as a disjunctive linear program with only positive literals.

- **Test:**
  - Scheduler with conflict extraction.



Goals can be relaxed in many ways

Self-Monitoring, Self-Diagnosing Systems

- Self-monitoring is essential to languages and architectures used to build robust robotic systems.
- Causal link monitoring enables robots to anticipate and adapt to sub-goal failures.
- Probabilistic mode estimation enable robots to diagnose component faults that lead to action failure.
- Conflict-direct search solves large decision problems by learning why hypotheses fail.
- Conflict-directed relaxation can be used to collaboratively resolve inconsistent plans and programs.

Questions?