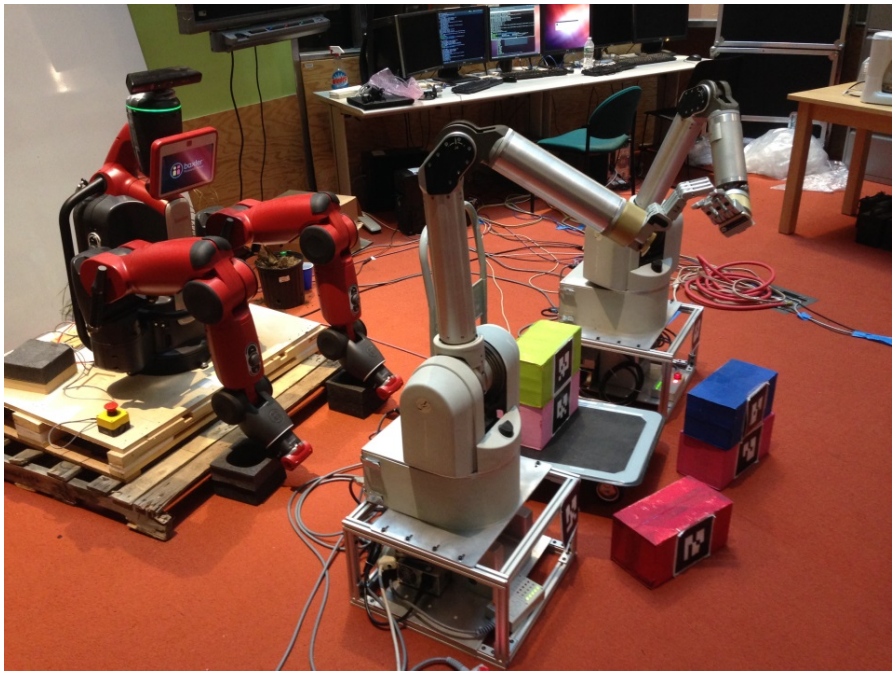
1st Summer School on Cognitive Robotics

I.e., How to create “thinking” robots



Introduction: Architectures for Autonomy

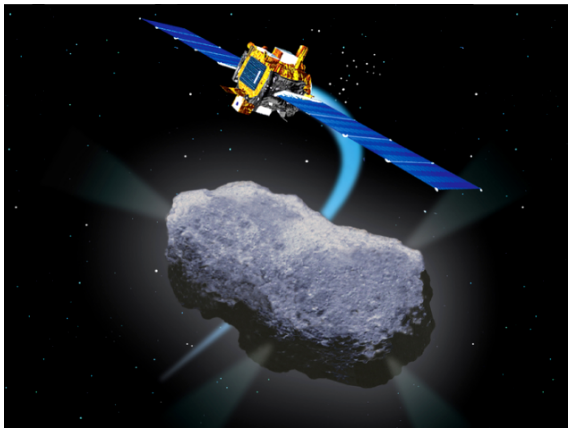
Brian Williams and Tiago Vaquero

June 12th, 2017

Outline

- **Summer School in a Nutshell**
- Architectures for Autonomy
- Programming Cognitive Robots

Summer School is about Creating Goal-directed Systems



1. Decision-making components for generating, refining, executing and monitoring plans on line (15 Tutorials).
2. Architectures and principles for creating goal-directed systems from decision-making components
   (3T, Remote Agent, Claraty, MDS, Idea, Enterprise/ROS, ROSplan).
3. Dynamic languages, elevated to the goal-level through partial specification and operation on state
   (PRS, RAPS, TCC, Golog, RMPL).

Day Structure

		Today
8:30am - 10:00am	**Tutorial**	**9:30-10:20**
10:00am - 10:30am	Coffee break	10:20-10:30
10:30am - 12:00pm	**Tutorial**	
12:00pm - 1:00pm	Lunch	
1:00pm - 2:30pm	**Tutorial**	
2:30pm - 3:00pm	Coffee break	
3:00pm - 6:00pm	**Labs**	
6:00pm - 7:00pm	Open lab hours	

Tutorials and Labs

Day 1: Robust Execution Introduction 1: **Architectures for Autonomy** Tutorial 2: **Self-Monitoring, Self-Diagnosing Systems** Tutorial 3: **Temporal Networks for Dynamic Scheduling**	**Lab: Enterprise/ROS Familiarization** and **Robust Execution**
Day 2: Motion Planning Tutorial 4: Sampling-based Motion Planning Tutorial 5: Single and Multi-Robot Path Planning with Quality Guarantees Tutorial 6: Trajectory Optimization for Underactuated Robots	**Lab:** Incorporating Motion Planning for Autonomous Vehicles
Day 3: Activity Planning Tutorial 7: Classical Planning@Robotics: Methods and Challenges Tutorial 8: Planning in Hybrid Domains Tutorial 9: Planning of Concurrent Timelines	**Lab:** Incorporating Activity Planning
Day 4: Perception and Manipulation Tutorial 10: Multi-vehicle Routing with Time Windows Tutorial 11: Generative Models for Perception Tutorial 12: Fundamentals of Robotic Manipulation and Grasping	**Lab:** Manipulation and Multi-vehicle Routing
Day 5: Planning with Uncertainty and Risk Tutorial 13: Probabilistic Planning Tutorial 14: Localization and Mapping Tutorial 15: Risk-bounded Planning and Scheduling	**Lab:** Challenge, as it all comes together ….

Speakers

Andrew Coles
King's College London

John W. Fisher
MIT

Joerg Hoffmann
Saarland University

Luke Hunsberger
Vassar College

Lydia Kavraki
Rice University

Phil Kilby
CSIRO Data61 / ANU

Sven Koenig
USC

Scott Kuindersma
Harvard University

Mark Moll
Rice University

Alberto Rodriguez
MIT

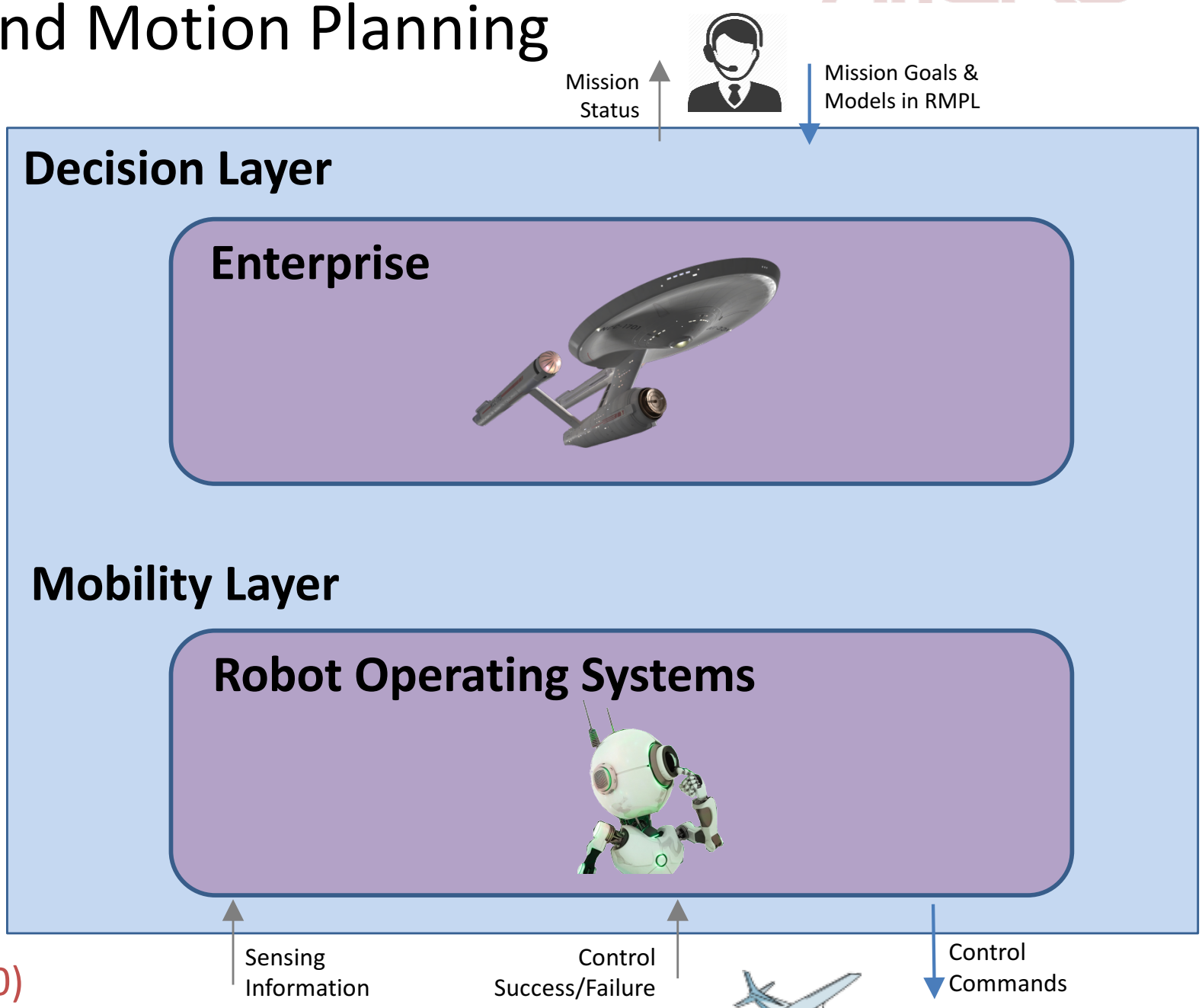
Nicholas Roy
MIT

David Wang
NuVu

Brian Williams
MIT

Shlomo Zilberstein
UMass Amherst

Labs use a Goal-directed Executive for Task and Motion Planning



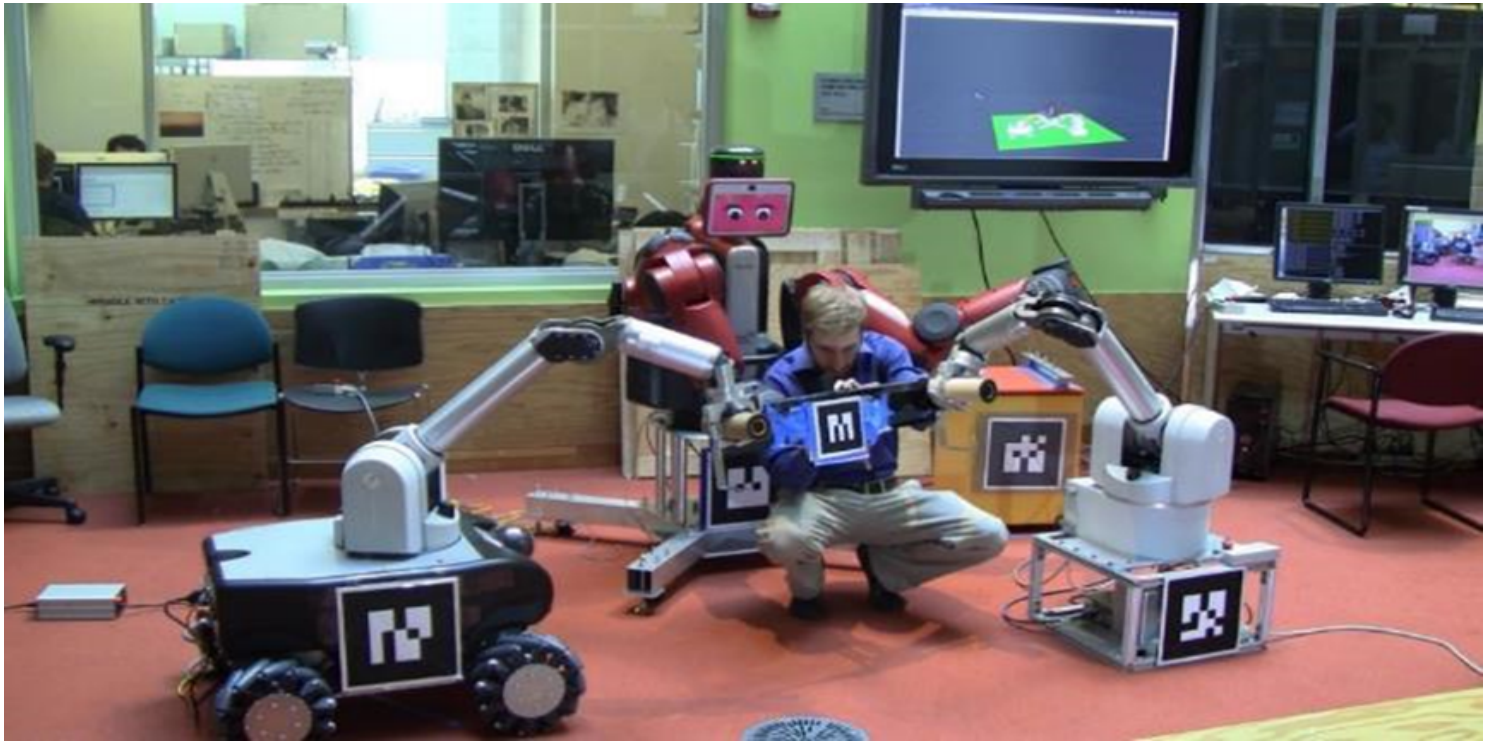
Similar to Claraty, NASA (Volpe 2000)

Labs embedded in simulations and hardware

Working out-of-the-box with your ‘15 holiday gift

Labs Driven by a Grand Challenge

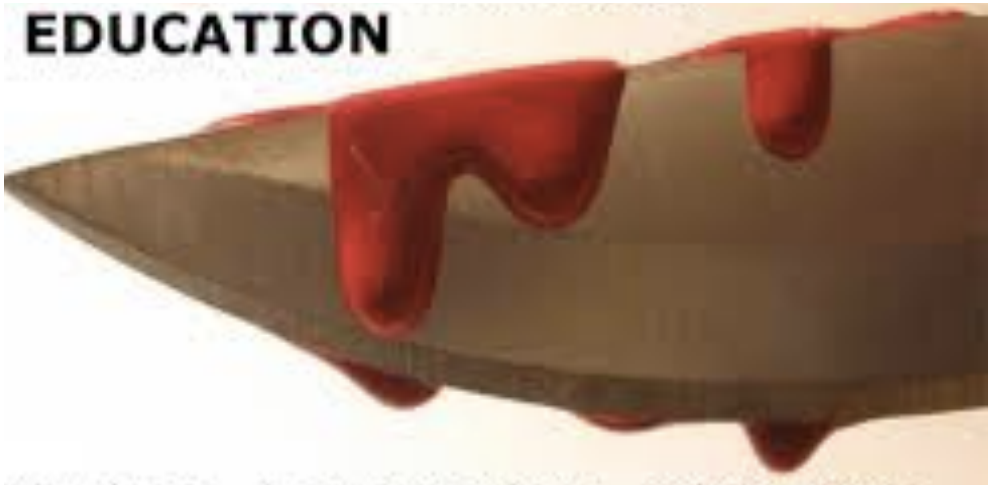
Labs



Model-based Embedded and Robotic Systems Group (MERS)

Room 226, 2nd Floor, The Stata Center, Building 32

Be warned – bleeding edge!



Lunch

You supply! Many options around campus . . Explore!

Student Center include Cambridge Grill, Shawarma Shack (Middle Eastern Food), Cafe Spice, Laverde's, and Subway, food trucks, sub shops, take out, brew pubs and gourmet restaurants

Clover (Vegetarian), Flour Bakery, Koch Cafe, Sebastians, Sata Grill Food Truck, Chinese Food Truck, Chipotle Mexican Grill, and Panda Express.

Social Dinner – Summer Shack

Tuesday, June 13 @ 7pm

Transportation not provided. Taking subway recommended

Need Help or Have Questions?

Talk to one of the organizers or the MERS group team.
Look for folks with **blue lanyard**

Need directions?
Use the ***Summer School Booklet*** as a reference

Most Important ….

Learn and

Have Fun!

Tiago, Brian and the MERS Team

Outline

- Summer School in a Nutshell
- **Architectures for Autonomy**
- Programming Cognitive Robots

Some Papers to Read

- R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand, An Architecture for Autonomy, The International Journal of Robotics Research, Vol 17, Issue 4, 1998.
- R. Bonasso, D. Kortenkamp, D. Miller, M. Slack. Experiences with an architecture for intelligent, reactive agents, JETAI, 1996.
- G. Brown, D. Bernard, R. Rasmussen, Attitude and articulation control for Cassini spacecraft: A fault tolerance overview. In 14th AIAA/IEEE Digital Avionics Systems Conference, 1995.
- M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, M. Carreras. ROSPlan: Planning in the Robot Operating System, In Twenty-Fifth International Conference on Automated Planning and Scheduling, 2015.
- A. Ceballos, S. Bensalem, A. Cesta, L. De Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini. A goal-oriented autonomous controller for space exploration. Proceedings of ESA Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA), vol 11.
- K. Currie , A. Tate , S. Bridge. O-Plan: the Open Planning Architecture, Artificial Intelligence, Volume 52, Issue 1, pp 49-86, November 1991.
- R. Firby. Adaptive execution in complex dynamic worlds. Ph.D. Dissertation, Yale University. 1978.
- R. Firby, Task networks for controlling continuous processes, In Proceedings of the Second International Conference on AI Planning Systems Chicago IL, June 1994.
- E. Gat. Integrating planning ans reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In Proc. of AAAI, 1992.
- E. Gat. On Three-Layer Architectures, in Artificial Intelligence and Mobile Robots. MIT Press, 1997.
- K. Haigh, M. Veloso. Interleaving Planning and Robot Execution for Asynchronous User Requests. Autonomous Agents, pp 79-95, 1998.
- B. Hayes-Roth. An Architecture for adaptive Intelligent Systems. Artificial Intelligence 72, 1995.
- J. Laird, A. Newell, P. Rosenbloom. SOAR: An architecture for general intelligence. Artificial Intelligence, 33(1), 1987.
- C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, R. McEwen. T-rex: A model=based architecture for AUV control, in 3rd Workshop on Planning and Plan Execution for Real-World Systems, vol 2007.
- D. Musliner, E. Durfee, K. Shin. CIRCA: A cooperative intelligent, real-time control architecture. IEEE Transactions and Systems, Man, and Cybernetics 23(6), 1993.
- N. Nilsson. Shakey The Robot, Technical Note 323. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.
- N. Nilsson, Teleo-Reactive Programs for Agent Control, Journal of Artificial Intelligence, Volume 1, pages 139-158, 1994.
- D. Tran, S. Chien, R. Sherwood, R. Castano, B. Cichy, A. Davies, G. Rabideau. The Autonomous Sciencecraft Experiment Onboard the EO-1 Spacecraft. In Proc. of AAAI 2004.

Examples of Cognitive Robotic Systems

Auto

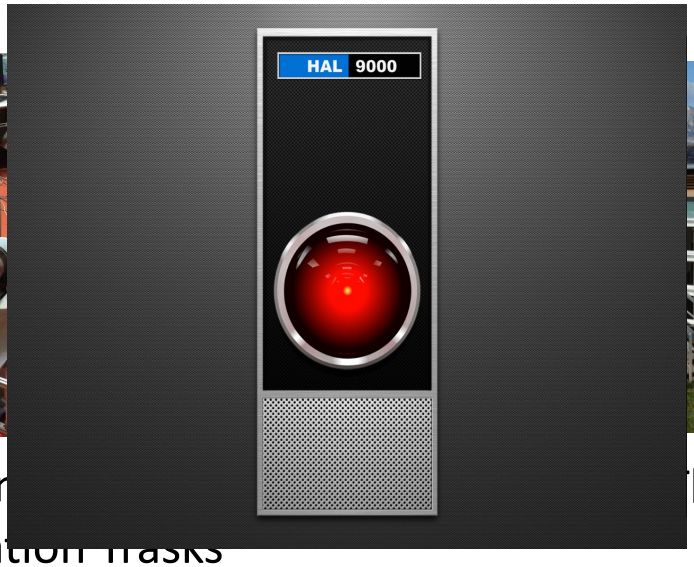
nformation gathering Scouts

cted

tive

tive

rmation



A

and

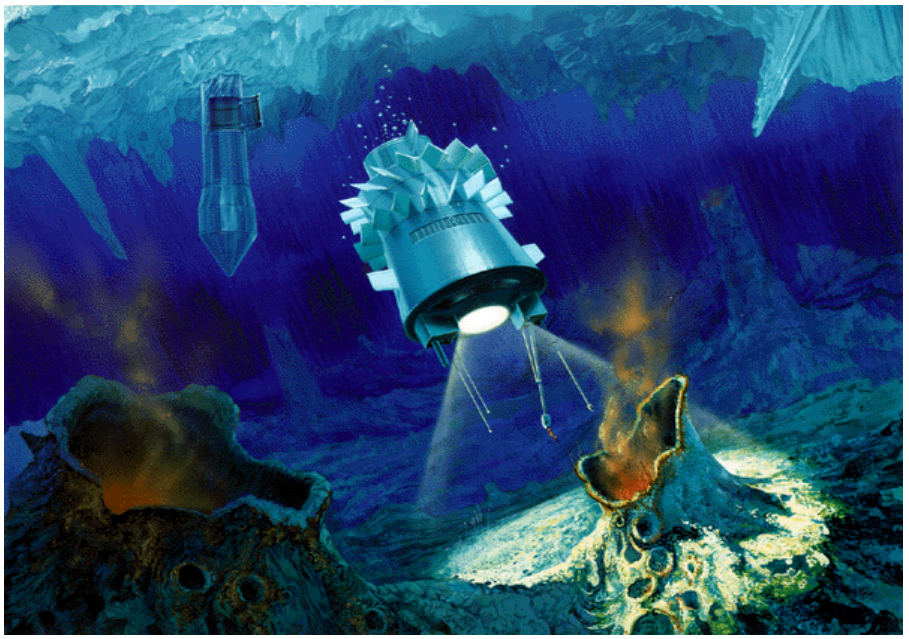
or Mar

nipulation Tasks

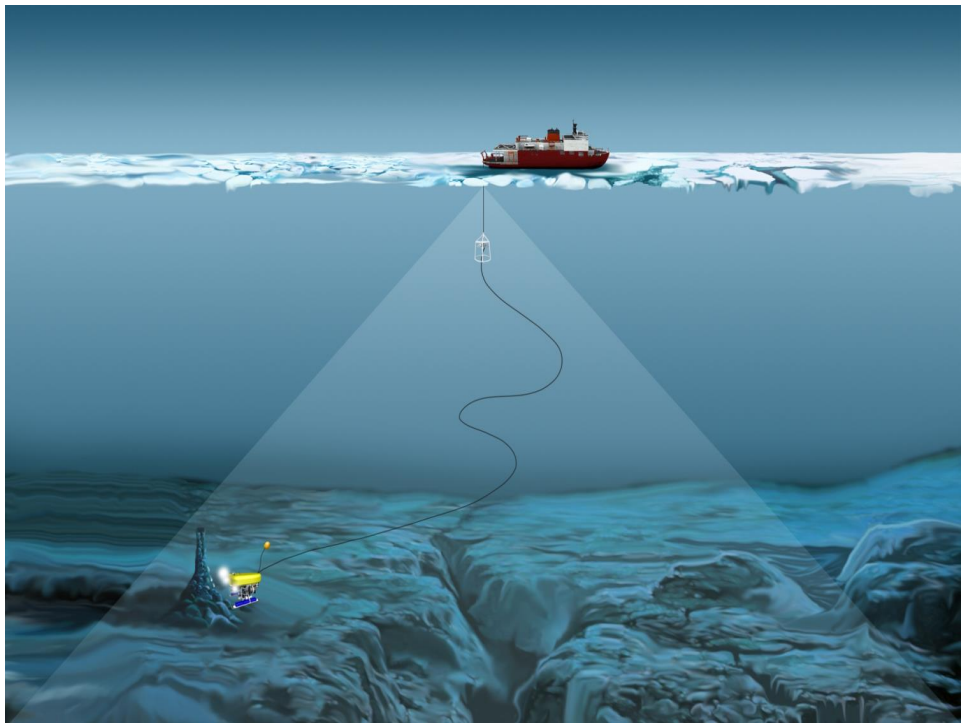
hings

The Challenge of Coordinating Robotic Systems

- We are creating increasingly autonomous embedded systems that perform critical functions over long periods of time.
- These long-lived systems achieve robustness by coordinating a complex network of devices.
- Architecting and programming these systems robustly is becoming an increasingly daunting task.



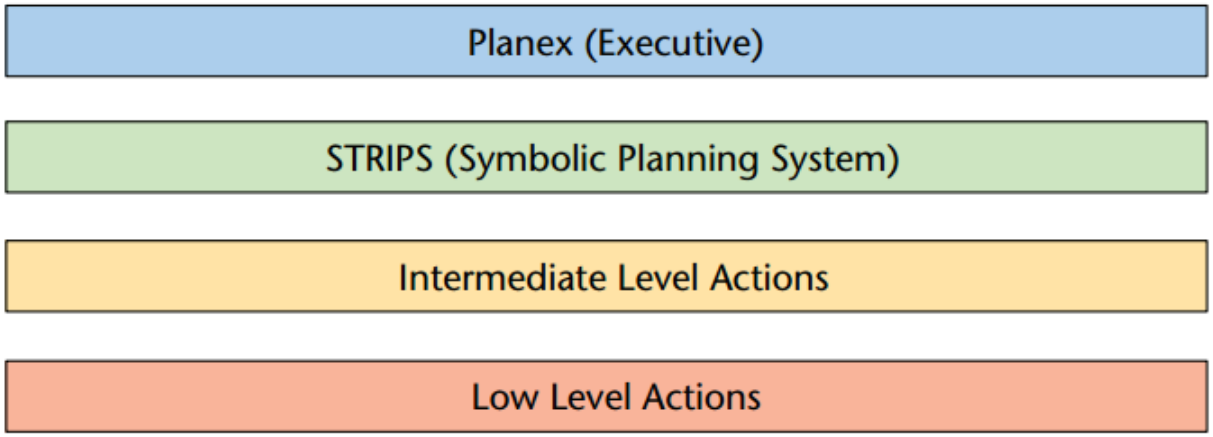
“Autonomous” vehicles explore far away places .. but can end in disaster!



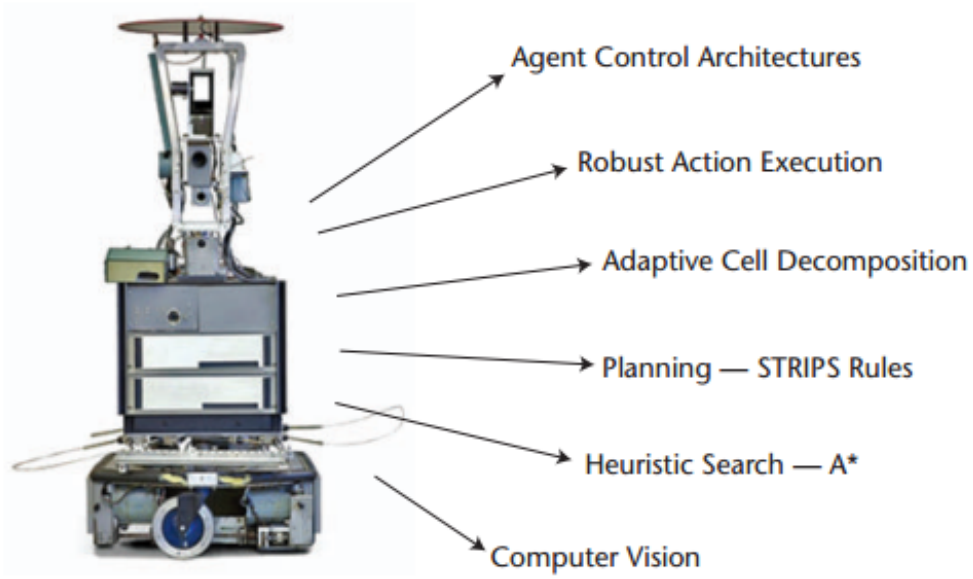
Most are just scripted!

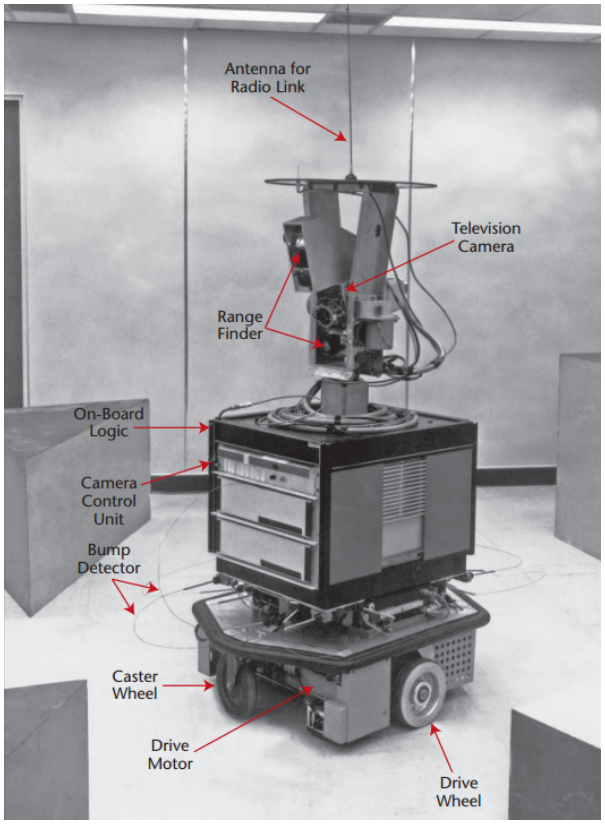
A Brief History of Architecting Robust Autonomous Systems

Shakey the Robot (SRI, 1966-72)



Layered Software Architecture





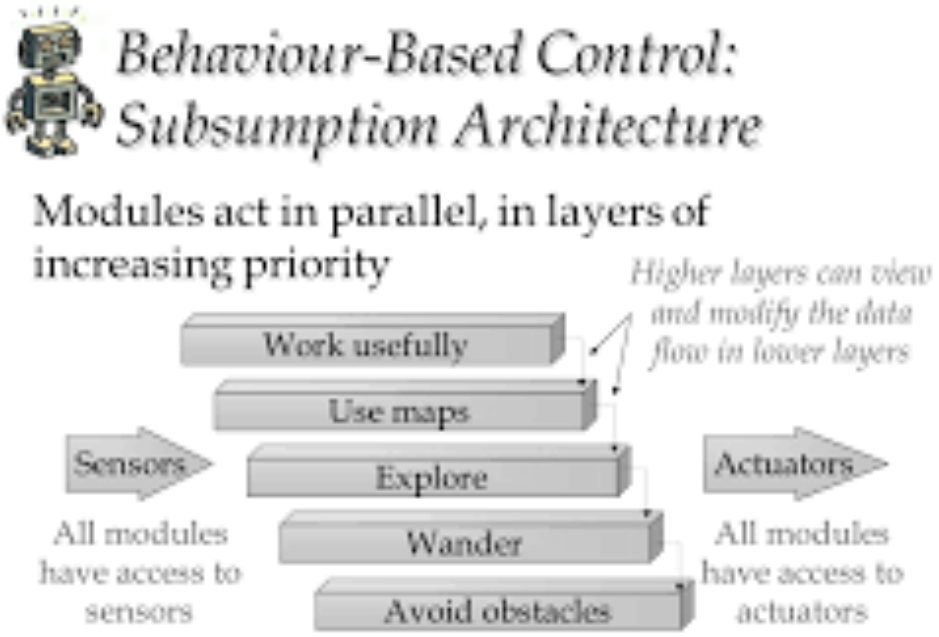
Shakey

Emphasizes Planning

- Activity
- Motion

Some similarities to RosPlan (Cashmore et al., 2015)

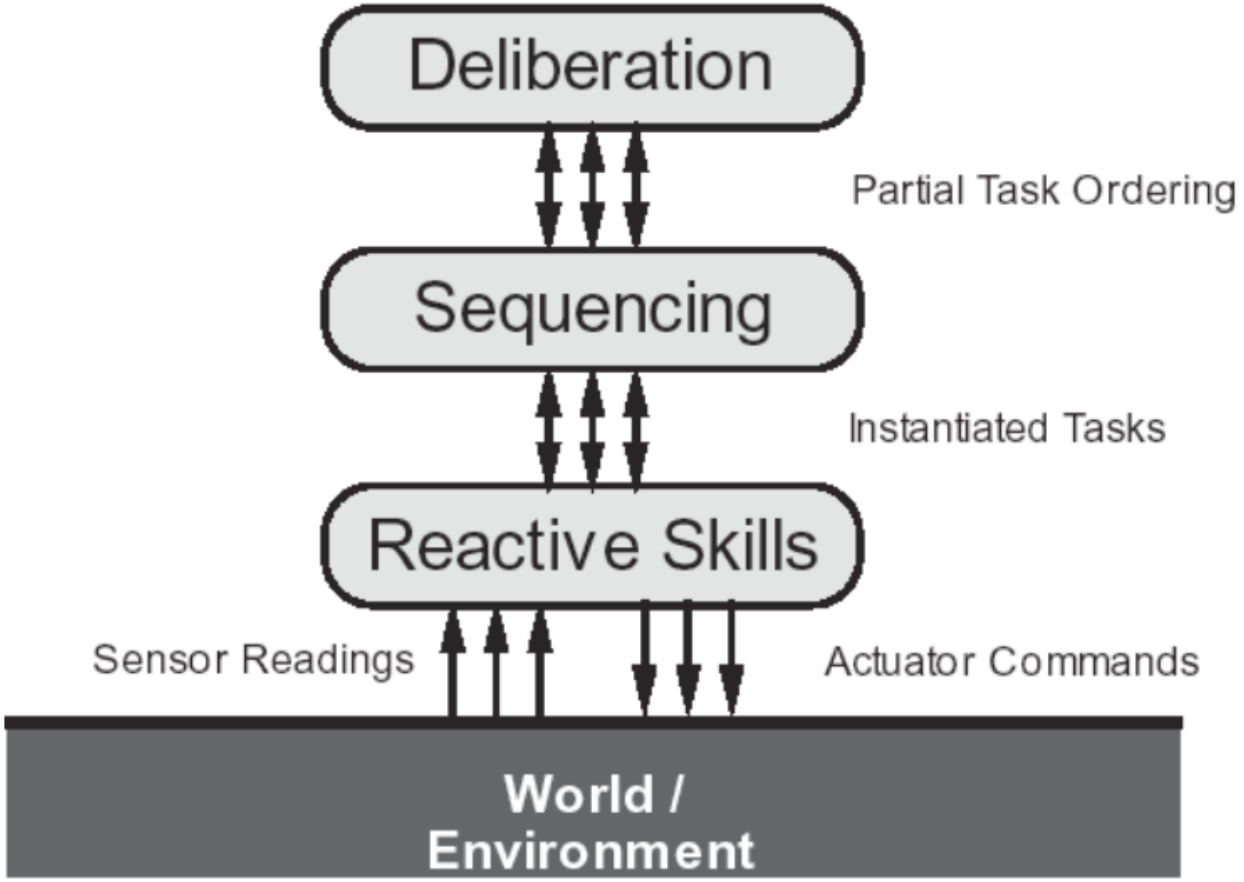
Layered Subsumption Architecture (Rod Brooks, 1986)



Focus on layered reactive systems

Similarities to MOOS (Benjamin, Leonard et al.)

Atlantis: Three Layered Architecture (Jim Firby et al., 1990)



Time-Consuming Deliberations

Reactive Plan Execution (RAPS)

Reactive Feedback Control

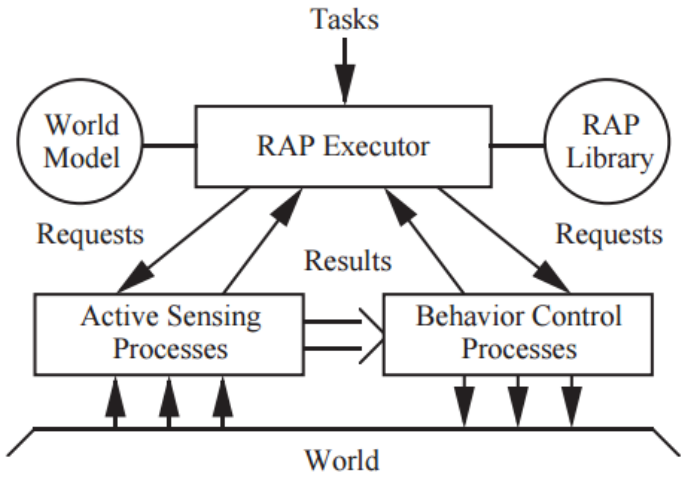
Similarities to 3T(Bonasso et al.) and many others

The RAPs Execution System (James Firby 1989)

```
(Define-rap (attach-at-site ?thesite)
  (succeed (docked ?thesite))
  (method
    (context (ferrous-hull ?thesite))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (t2 (magnetically-attach ?thesite)
            (wait-for (docked ?thesite))))))
  (method
    (context (not (ferrous-hull ?thesite)))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (t2 (grip-attach ?thesite)
            )wiat-for (docked ?thesite)))))
```

Repair Action Procedures (RAP):

- **Goal-directed task-decomposition.**
- **Non-deterministic choice.**
- **Closed-look monitoring on goals.**



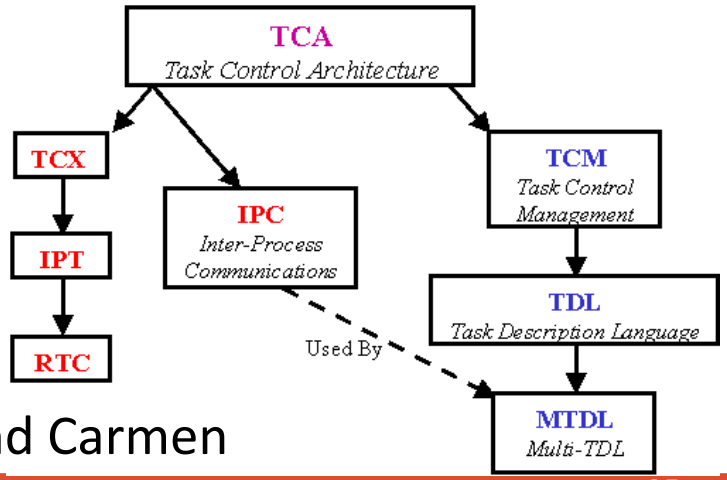
Similarities to Golog (Levesque et al), Alisp (Russel et al , and RMPL (Williams et al)

Task Control Architecture (Reid Simmons, 1994)

high-level robot operating system that balances goal-directed task decomposition and reactive control.

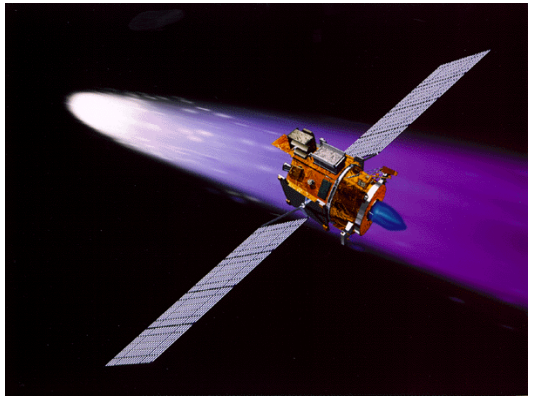
Supports:

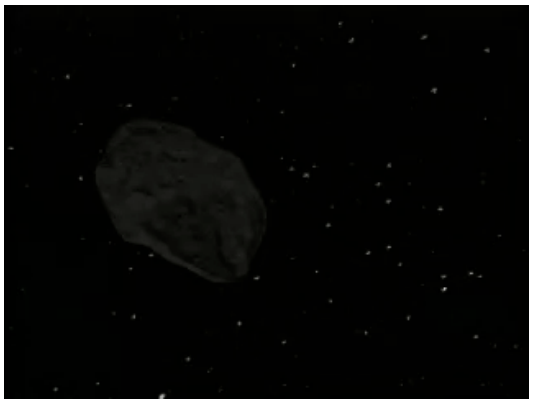
- distributed communications,
- task decomposition,
- resource management
- execution monitoring
- and error recovery.

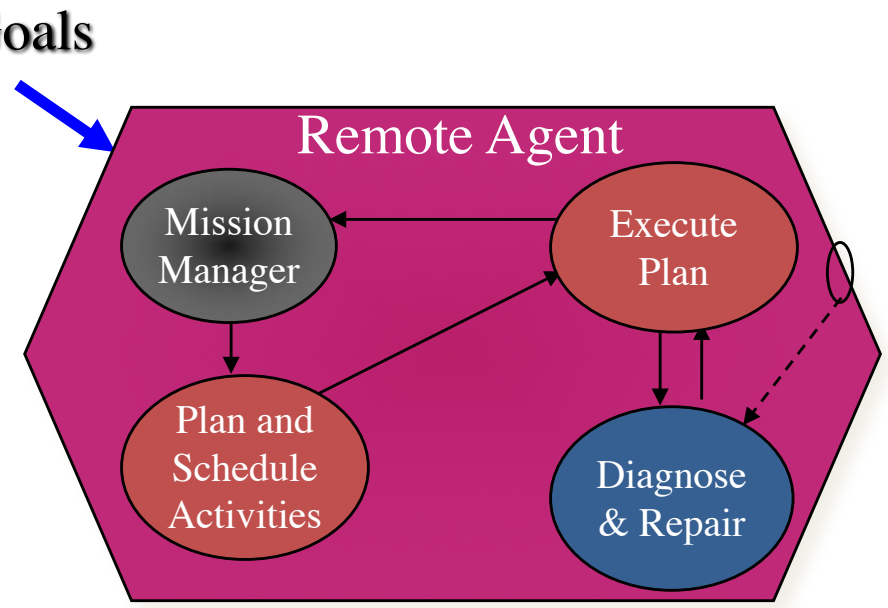


Similarities to Player Stage, Claraty, ROS, MOOS, and Carmen

Remote Agent on Deep Space One, Spring, 1999





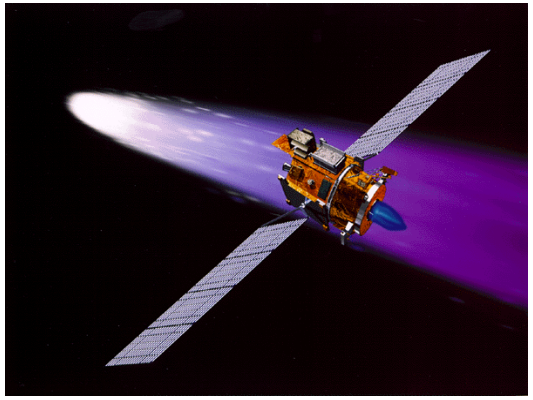


1. Commanded by giving goals
2. Reasons extensively from models
3. Closes loop on goals
4. Diagnoses, repairs and continuously re-plans

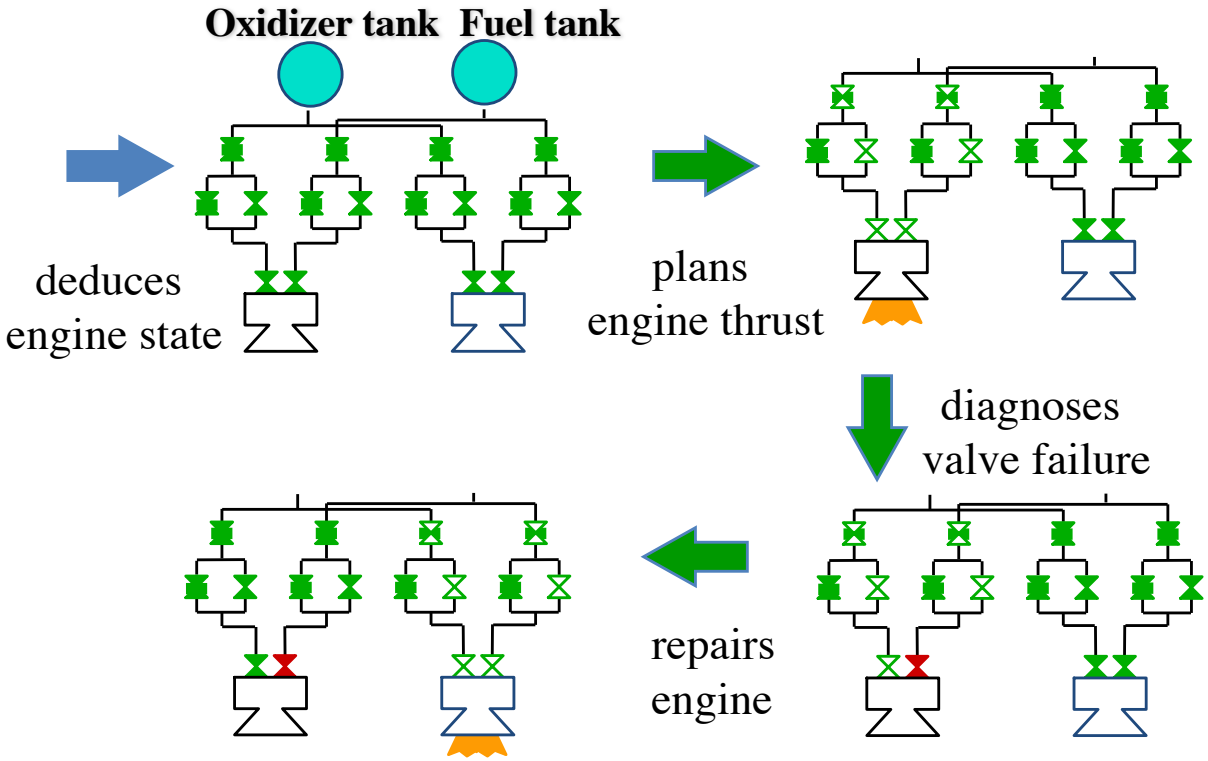
Muscettola, Chien; Pell, Gat; Williams, Nayak, et. al.

Similarities to IxTet, IDEA, Casper, MDS, TREX, Titan, Enterprise

Remote Agent Planners (HSTS and Livingstone)

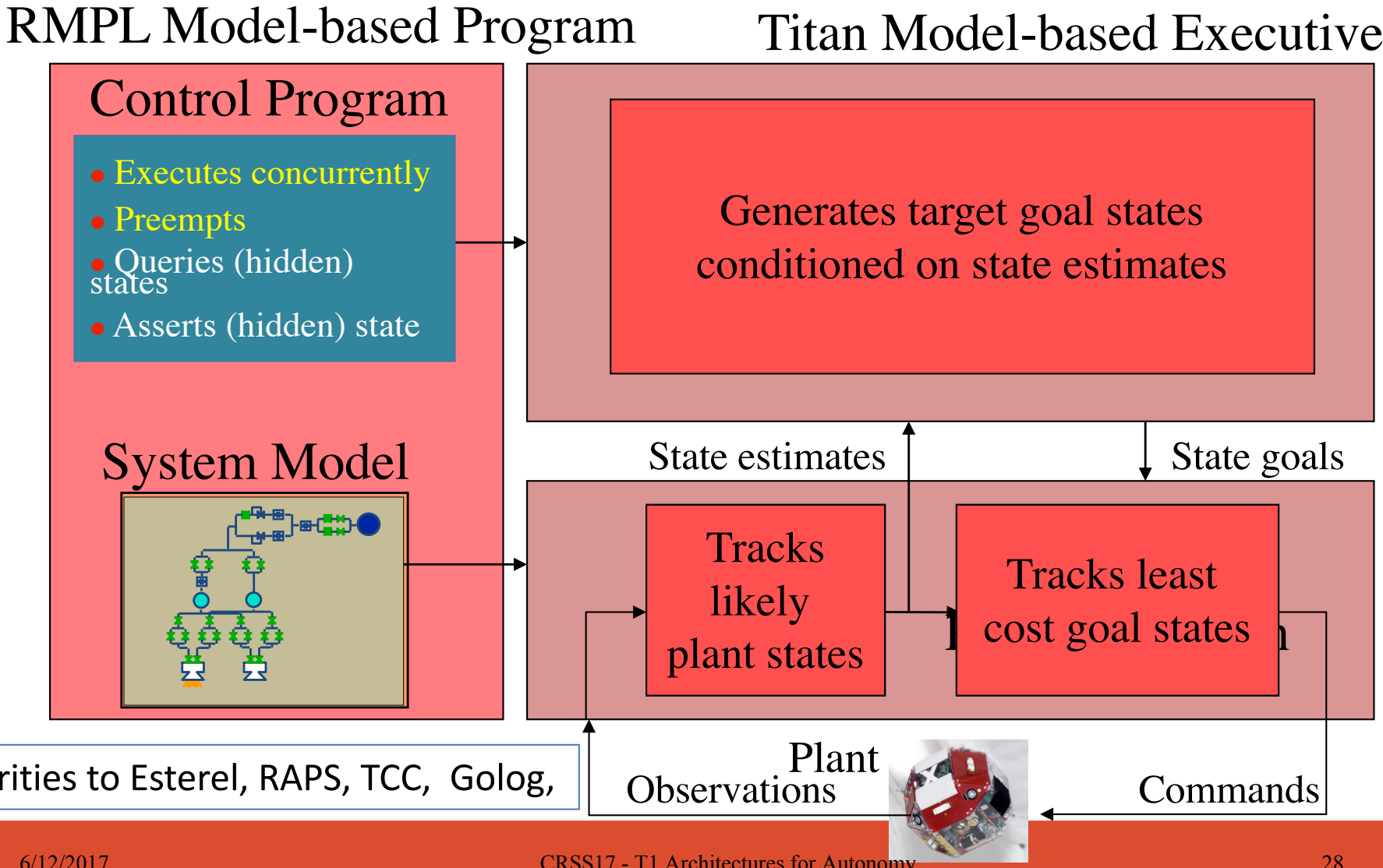


1. Plans concurrent processes (timelines).
2. Plans for time and resources.
3. Plans to control devices with indirect effects.
4. Plans and deduces at reactive times scales.



Programs on Hidden State
(Williams et al., 2003)

Planners fully absorbed into execution language.



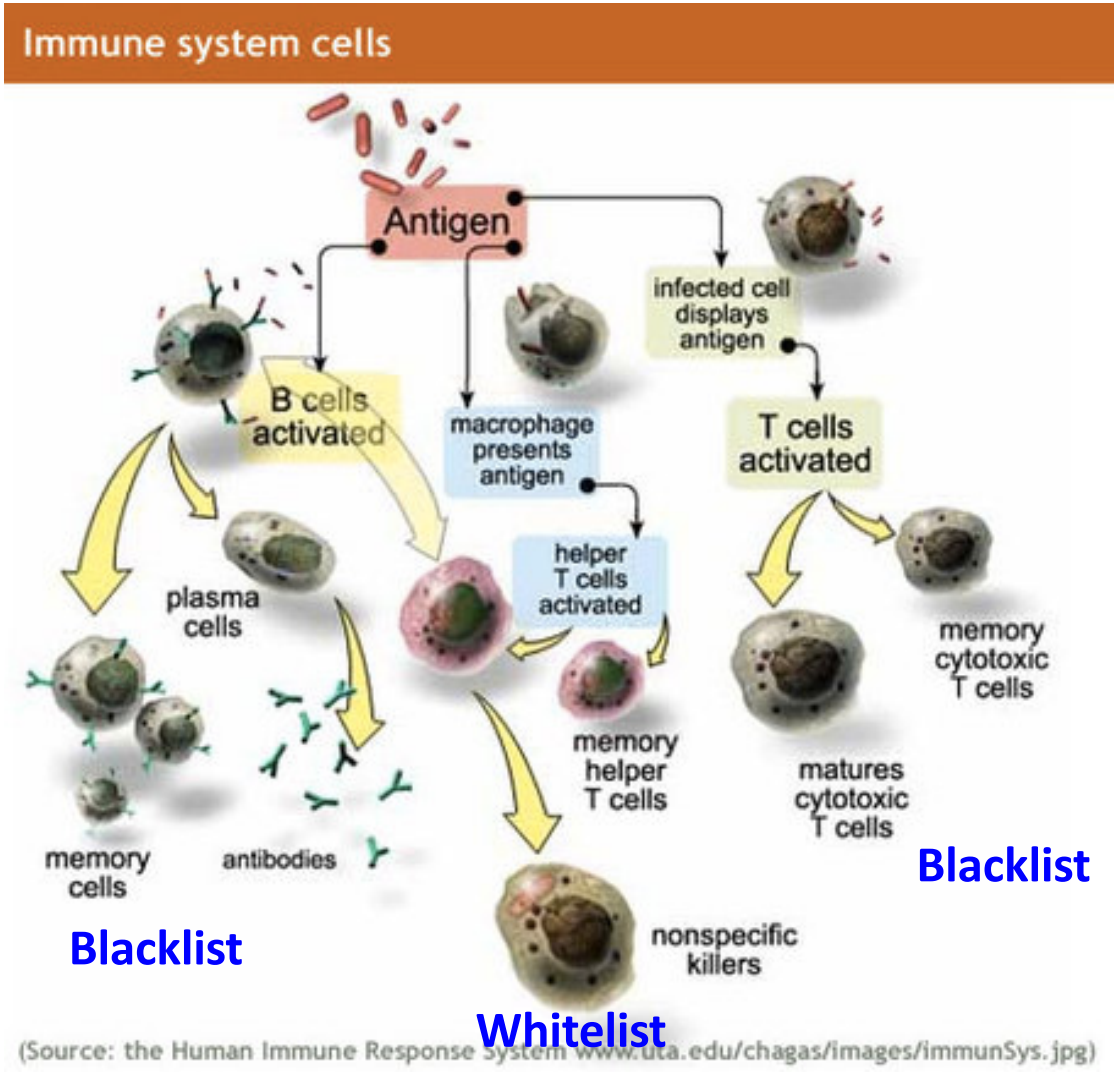
Similarities to Esterel, RAPS, TCC, Golog,

And Many More . . .

- PRS (Georgeff, Lansky)
- Propel (Leveinson)
- IDEA (Pell, Muscettola)
- Casper (Chien, Rabideau et al)
- MDS (Rasmussen et al)
- Claraty (Volpe et al)
- IxTet
- TRex
- Moos (Benjamin et al)
- Carmen (Thrun et al)
- Player/Stage (Gerkey et al)
- RosPlan (Cashmore et al)

For autonomy to receive broad adoption, managing risk is key.

What Helps to Make Humans Robust



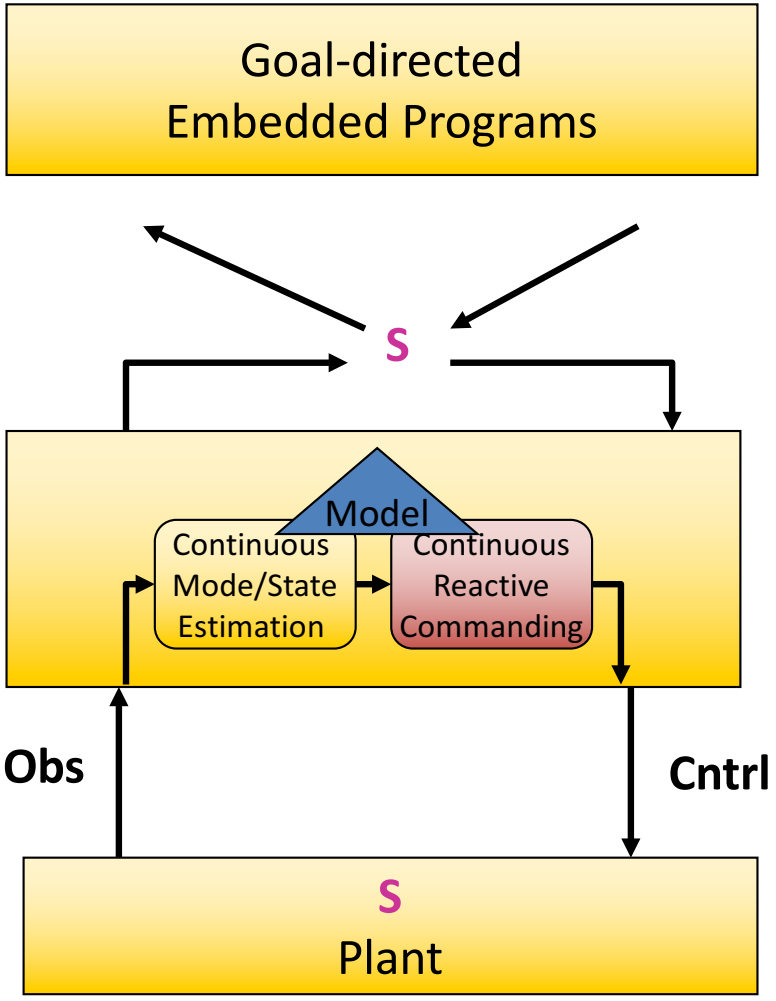
- Immune system
- Physical barriers
- Inhospitable environments
- Tolerance
- Death

Complements of Jake Beal, BBN

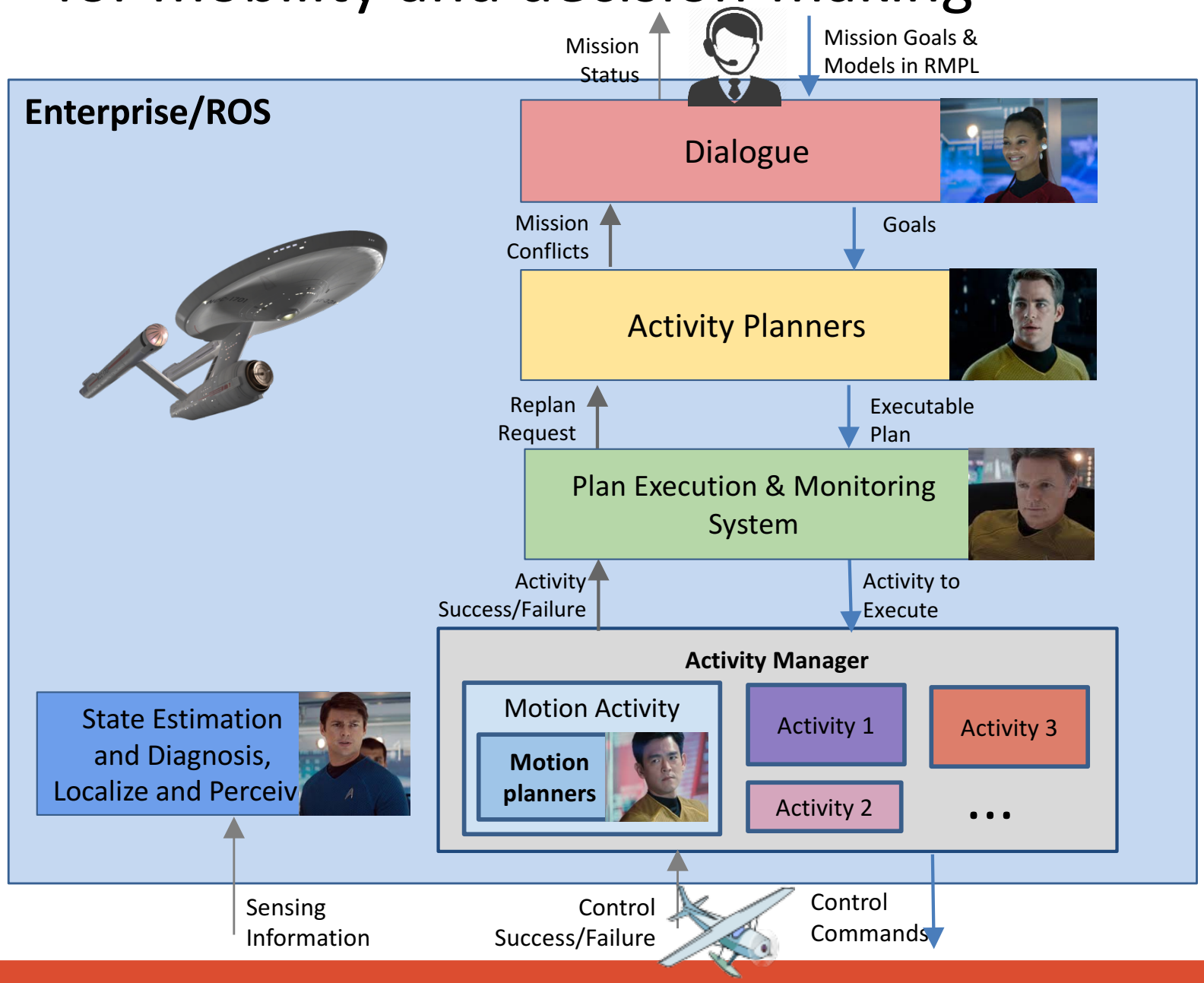
Goal-directed Systems Offer Layers of Defense

Architectures that achieve robustness through decision layers that are:

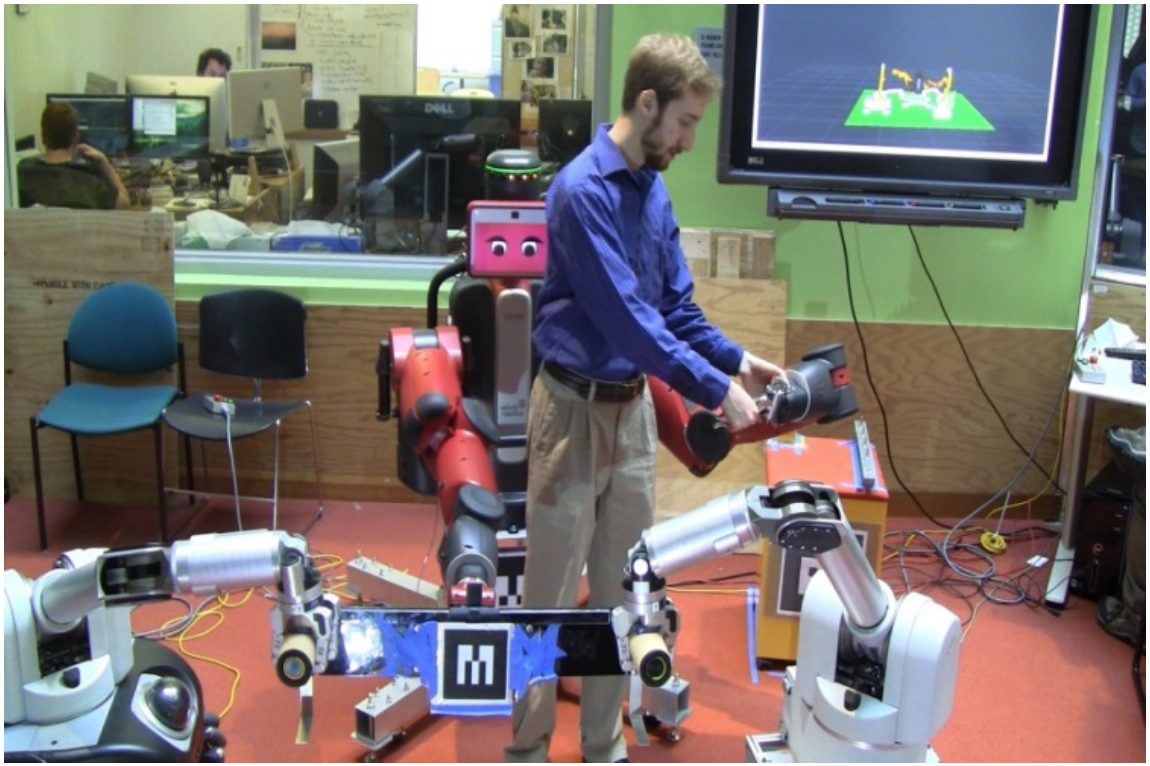
- Suspicious (M)
  - Monitor states and goals.
- Adapt to disturbance (M)
  - Adjust timing
  - Select contingencies
- State Aware (W)
  - Plan to achieve goals states.
- Precise (T,R)
  - Achieve continuous goal states.
- Manage Risk (F)
  - Executes plans in uncertain environments with bounded risk.
- Collaborate (~M)
  - Execute tasks, revise goals and plan with humans.



Enterprise/ROS: goal-directed model-based executive for mobility and decision-making



Components:
Planning as State Space Search



- Planning as HFS
- LPA*
- PRM
- RRT
- Belief update

Tutorials: T4, T5, T7, T8, T13, T14, . . .

Components:
Planning as Constraint Programming

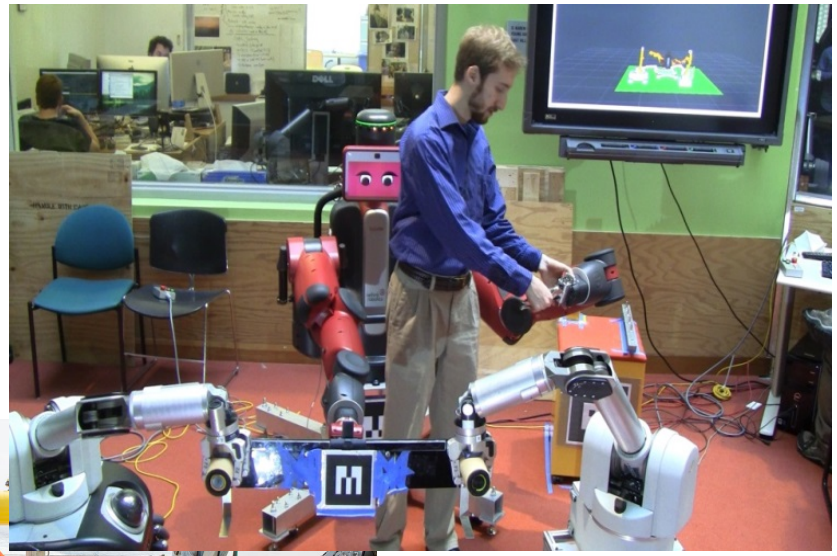
Constraint Types:

- Finite Domain
- Logical
- Temporal
- Global

- Linear
- Convex
- Mixed-integer

Tutorials: T2, T3, T6, T9, T10, T12, T15, . . .

Components: Perception

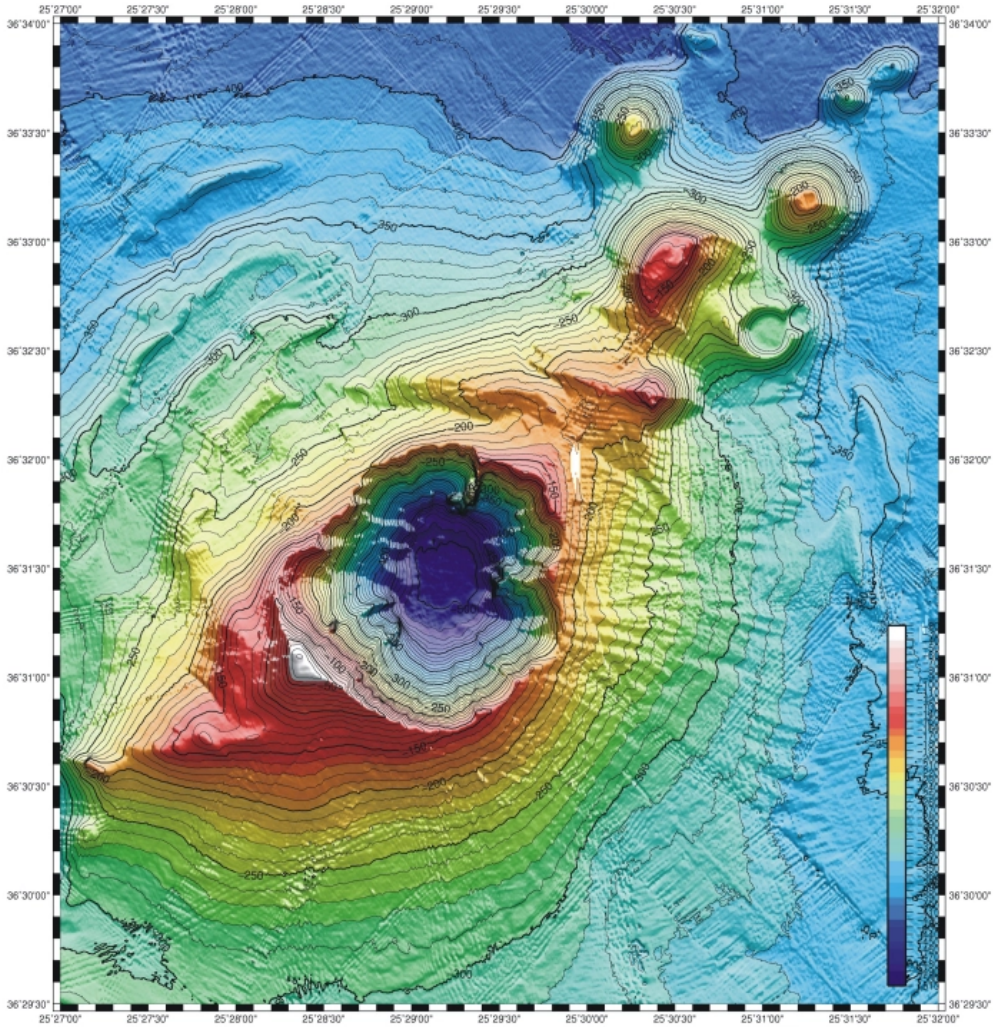


- Goal-monitoring
- Mode estimation
- Graphical models
- SLAM

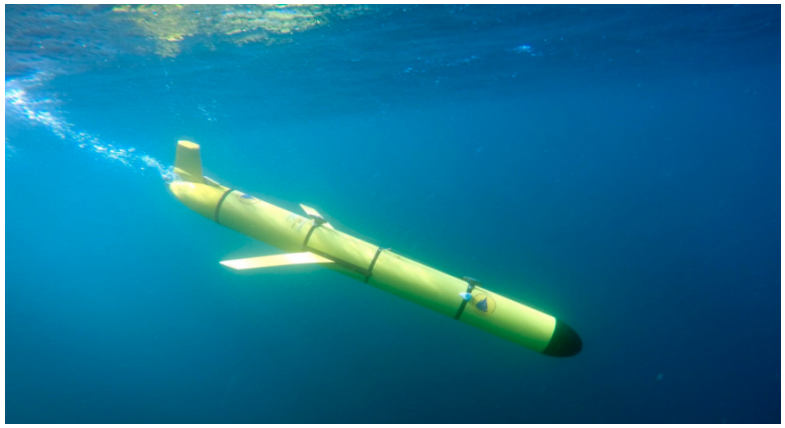
Tutorials: T2, T11, T14

A Challenge that Requires Everything!

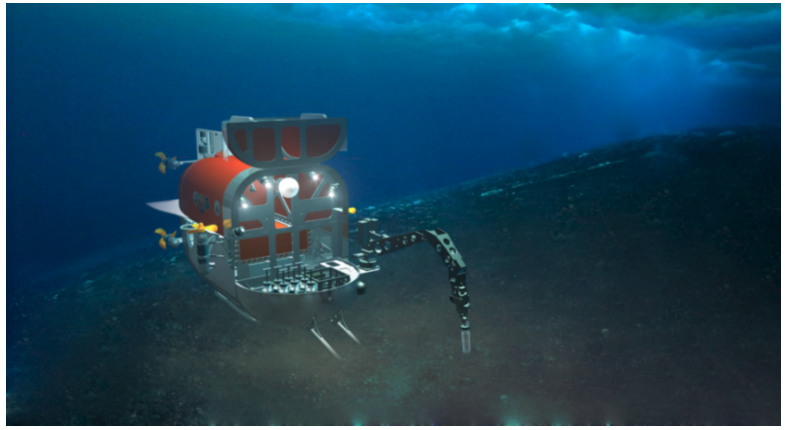
2019: Europa Analog Mission Demonstration



Reconnaissance



Sample return



Mission: Look for evidence of “extreme” life at Kolumbo Deep-Sea Volcano near Santorini, Greece

Sponsor: NASA PSTAR Program

Team: WHOI, MIT, ACFR, U. Michigan

Outline

- Summer School in a Nutshell
- Architectures for Autonomy
- **Programming Cognitive Robots**
  - Programs that make choices
  - Programs that monitor
  - Programs on state

Non-deterministic Languages

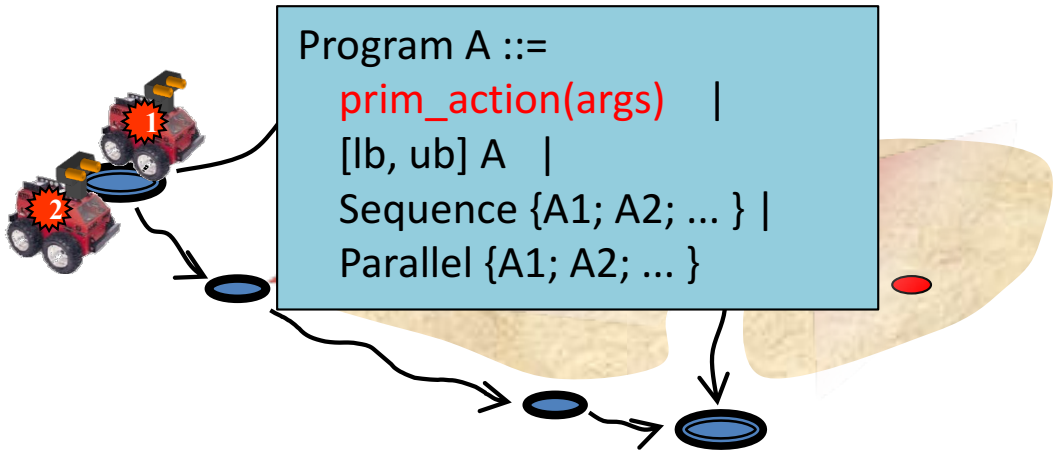
```
(Define-rap (attach-at-site ?thesite)
  (succeed (docked ?thesite))
  (method
    (context (ferrous-hull ?thesite))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (t2 (magnetically-attach ?thesite)
            (wait-for (docked ?thesite))))))
  (method
    (context (not (ferrous-hull ?thesite)))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (t2 (grip-attach ?thesite)
            )wiat-for (docked ?thesite)))))))
```

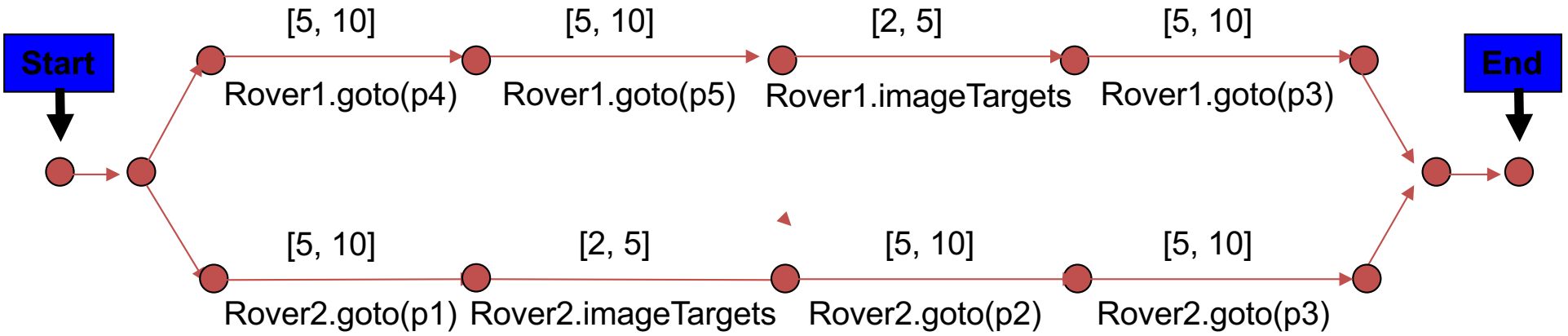
- **RAPS (Firby)**
- **ESL (Gat)**
- **ALISP (Russel).**
- **Golog (Levesque)**
- **RMPL (Williams)**
- . . .

Robust Program and Plan Execution

RMPL

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

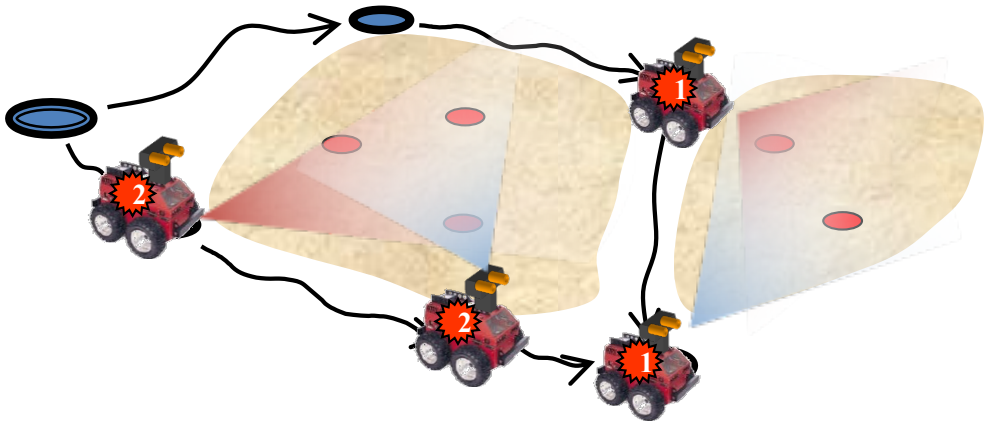


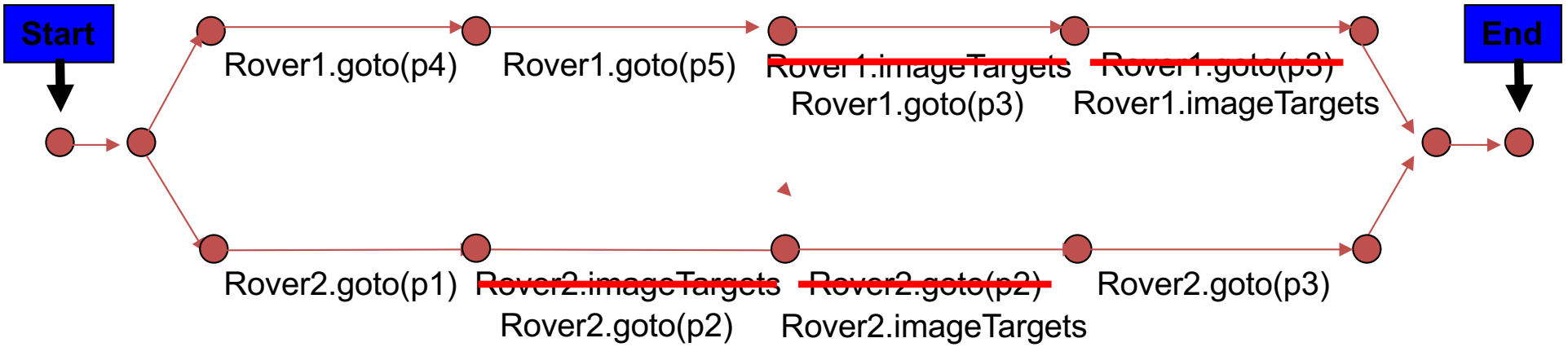


Dynamically schedules

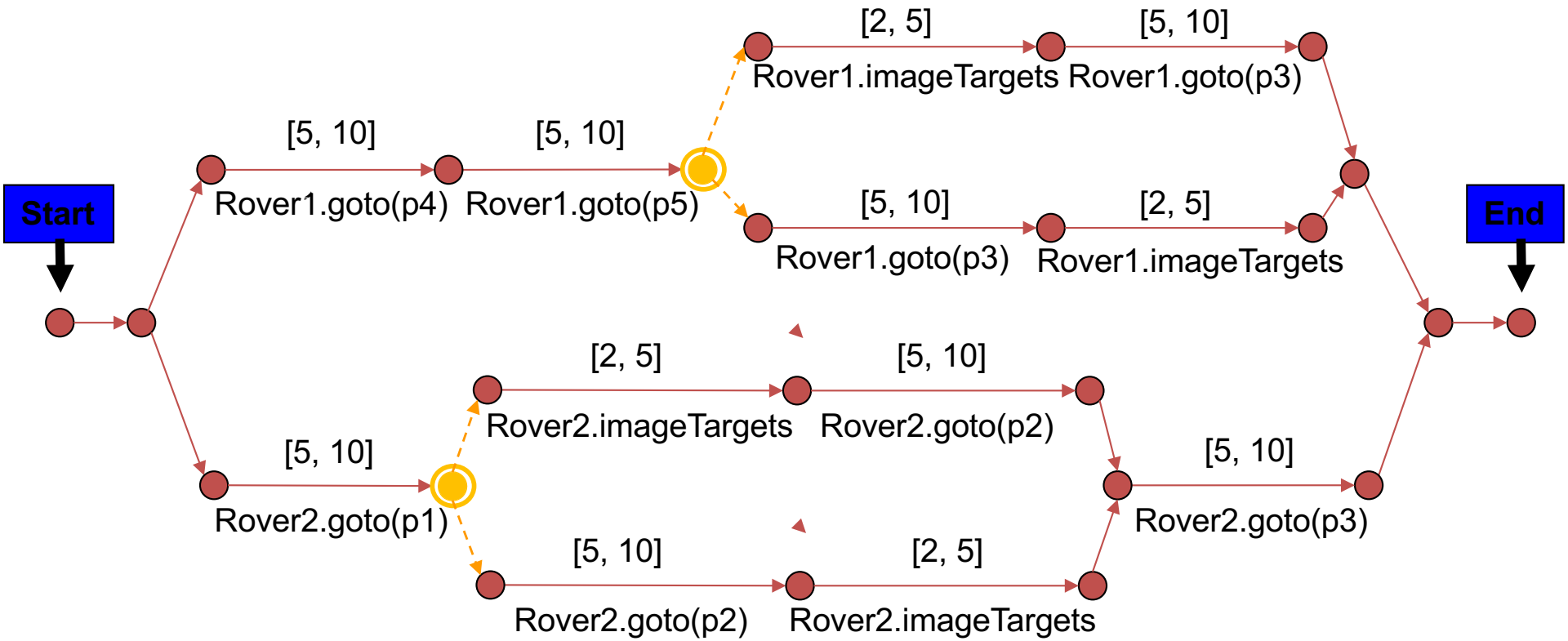
What if we want more flexibility in our plan?

A Different Choice





A Plan with **Choice**



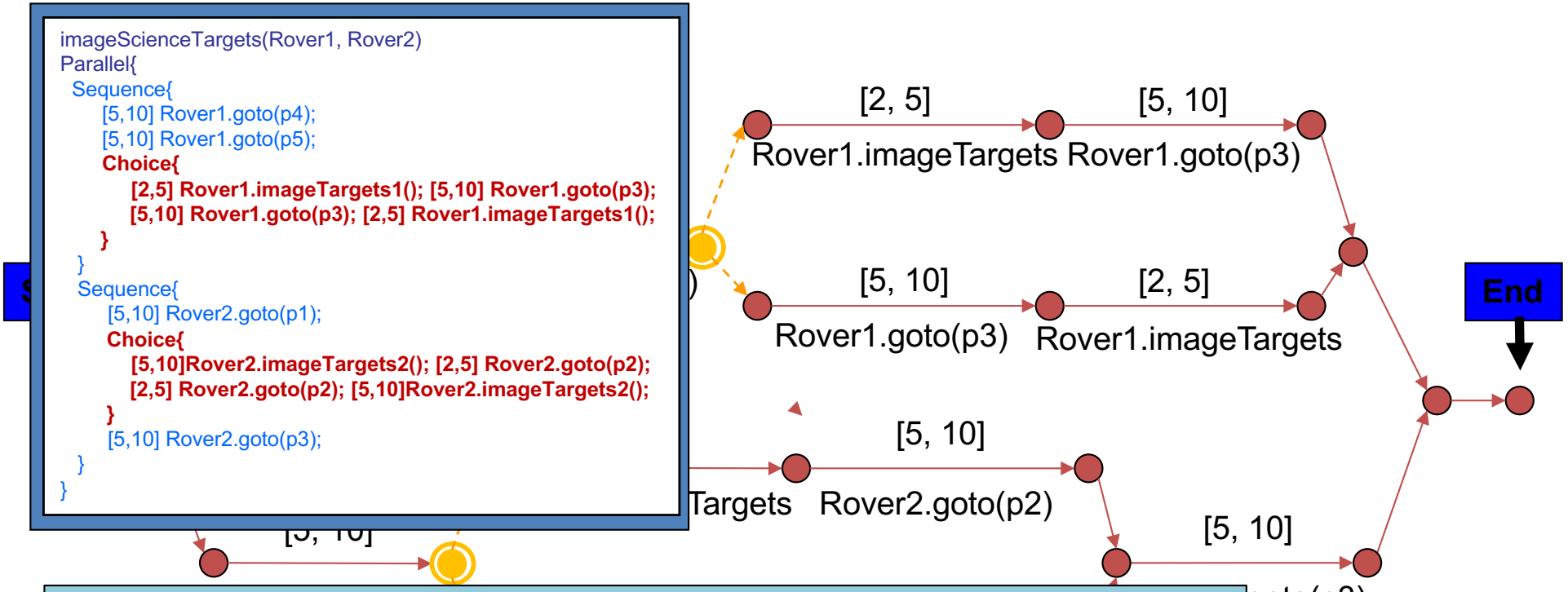
Problem make highest utility choices that are schedulable

Assume for this plan, that edges without explicit temporal constraints are [0,0].

Programs with (Nondeterministic) Choice

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);
  }
}
```



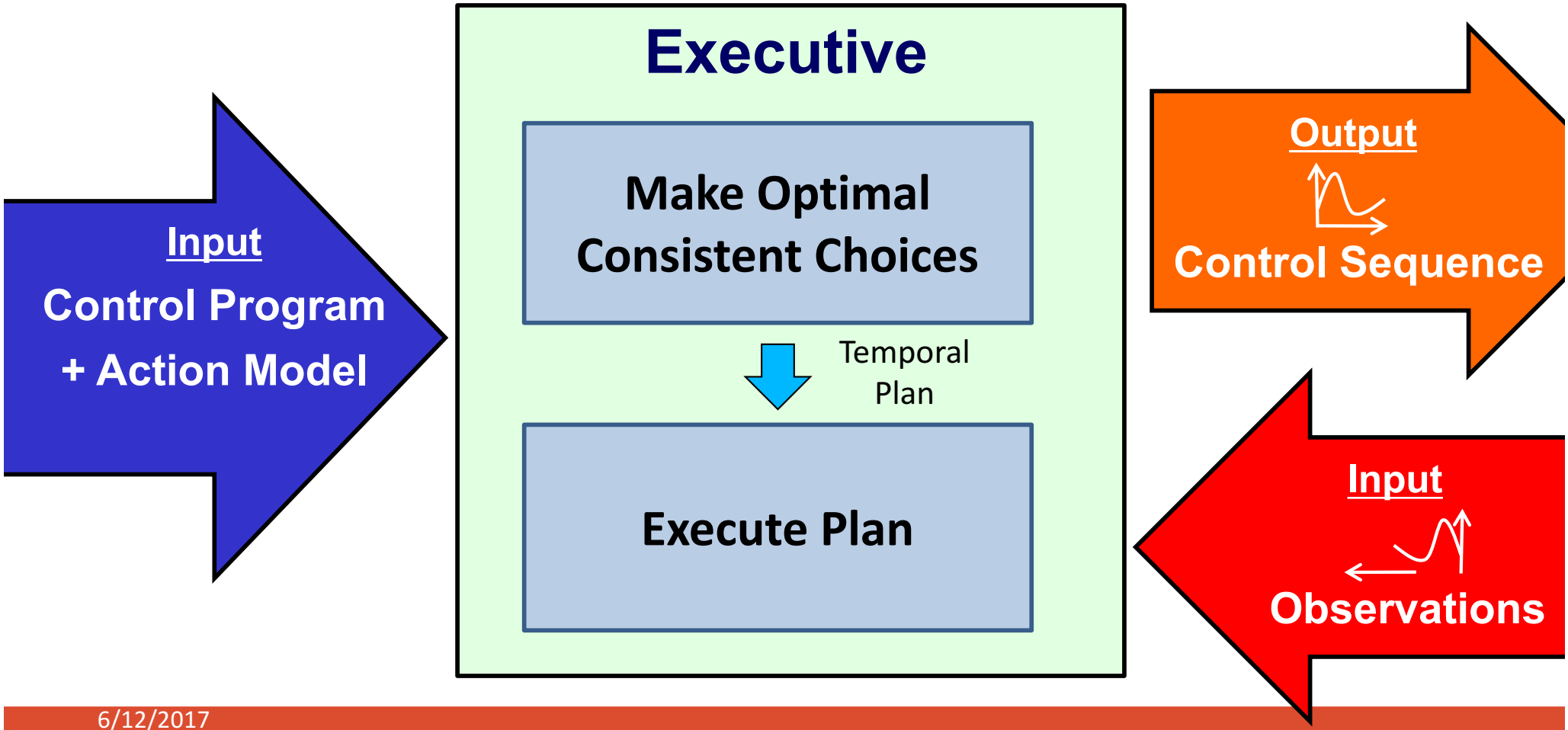
Program A ::=
- prim_action(args) |
- [lb, ub] A |
- Sequence {A1; A2; … } |
- Parallel {A1; A2; … } |
- Choose { [with reward R1] A1; [with reward] R2 A2; . . .}

Non-deterministic programs allow us to represent…

- When:
  Alternative orderings of actions
- With What:
  Alternative resource assignments
- Who:
  Alternative task assignments
- How:
  Alternative methods for completing a task

Executing Programs with Choice

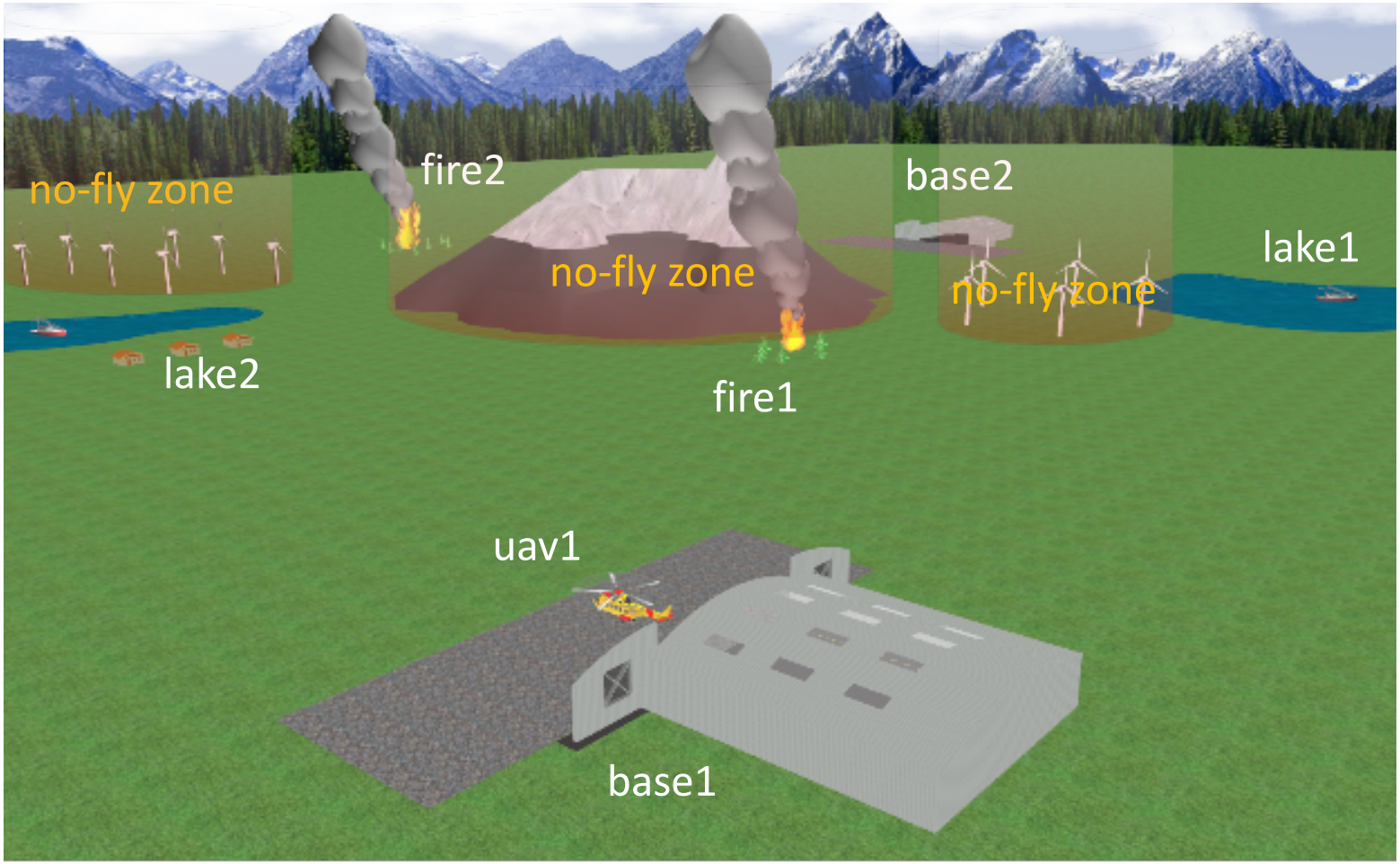
- **Plan**: Make optimal, consistent program choices.
- **Execute** : Dispatch and monitor resulting plan.



Outline

- Summer School in a Nutshell
- Architectures for Autonomy
- **Programming Cognitive Robots**
  - Programs that make choices
  - Programs on state
  - Programs that monitor

Programs on State - Firefighting Scenario



Firefighting in RMPL: Traditional Imperative program

```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
      ...

 method run() {
  sequence{
    uav1.takeoff();
    uav1.fly_base1_to_lake1();
    uav1.load_water(lake1);
    uav1.fly_lake1_to_fire1();
    uav1.drop_water_high_altitute(fire1);
    uav1.fly_fire1_to_lake1();
    uav1.load_water(lake1);
    uav1.fly_lake1_to_fire1();
    uav1.drop_water_low_altitute(fire1);
    uav1.fly_fire1_to_lake2();
    uav1.load_water(lake2);
    uav1.fly_lake2_to_fire2();
    uav1.drop_water_high_altitute(fire2);
    uav1.fly_fire2_to_lake2();
    uav1.load_water(lake2);
    uav1.fly_lake2_to_fire2();
    uav1.drop_water_low_altitute(fire2);
    uav1.fly_fire2_to_base1();
    uav1.land();
  }
}}
```

Firefighting in RMPL: A Program on State

```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
     ...

 method run() {
    sequence{
      (fire1 == out);
      (fire2 == out);
      (uav1.flying == no &&
       uav1.location == base_1_location);
    }
 }

}
```

Firefighting in RMPL: Setup & Initial Conditions

```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
     ...

 method run() {
    sequence{
      (fire1 == out);
      (fire2 == out);
      (uav1.flying == no &&
       uav1.location == base_1_location);
    }
 }

}
```

```
Main (){
     uav1 = new UAV();
     uav1.location= base_1_location;
     uav1.flying = no;
     uav1.loaded = no;

     lake1 = new Lake();
     lake1.location = lake_1_location;

     lake2 = new Lake();
     lake2.location = lake_2_location;

     fire1 = new Fire();
     fire1.location = fire_1_location;
     fire1 = high;

     fire2 = new Fire();
     fire2.location = fire_2_location;
     fire2 = high;
}
```

Firefighting in RMPL: Model of Actions

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

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

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

  primitive method load_water(Lake lakespot)
    ((flying == yes) && (loaded == no) && (lakespot.location == location)) => loaded == yes;

  primitive method drop_water_high_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes) && (firespot.location == location) && (firespot == high))
     => ((loaded == no) && (firespot == medium));

  primitive method drop_water_low_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes) && (firespot.location == location) && (firespot == medium))
     => ((loaded == no) && (firespot == out));

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

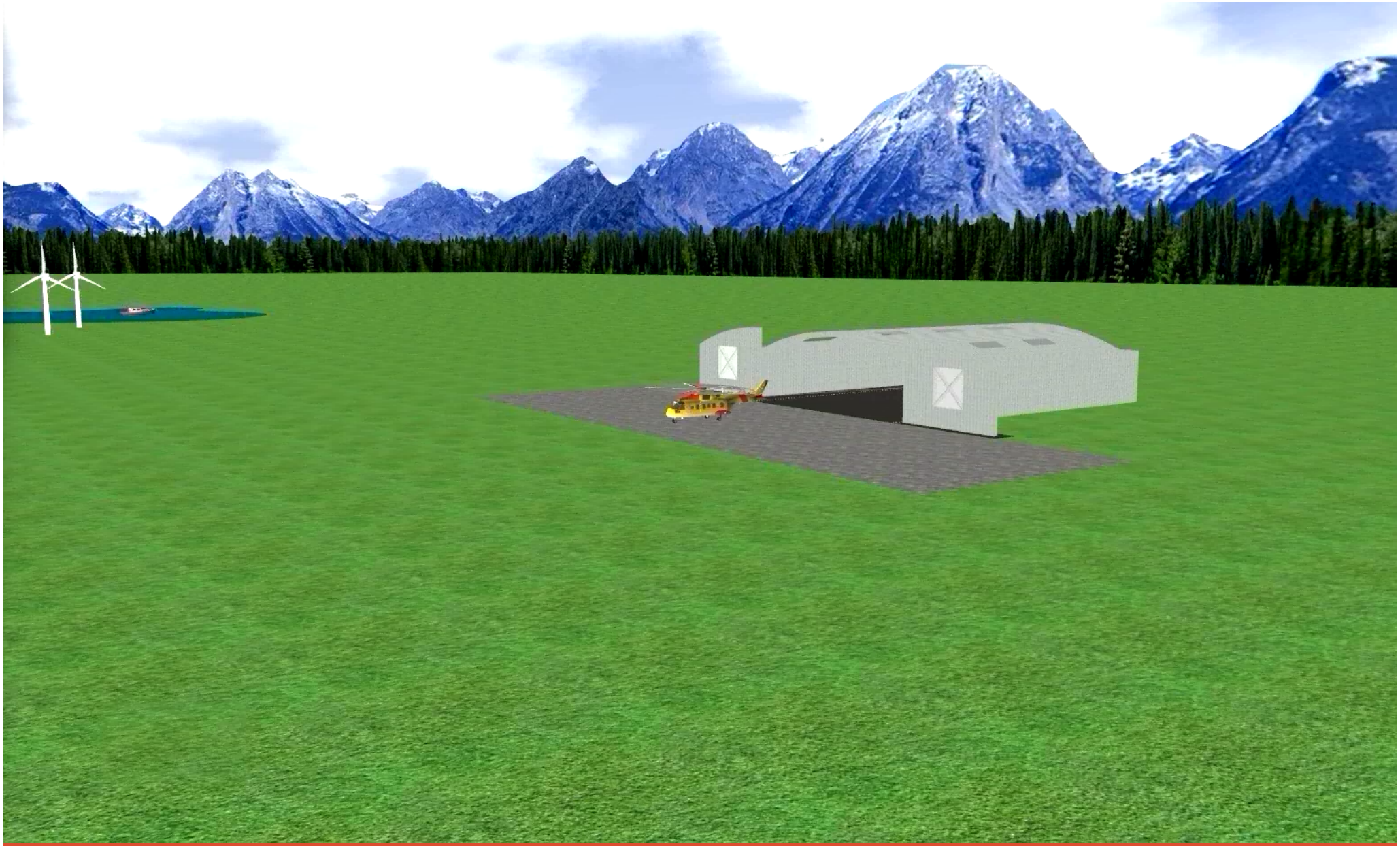
```
class Lake {
   Roadmap location;
}

class Fire{
   initial value high;
   value medium;
   value out;
   Roadmap location;
}
```

Firefighting in RMPL: Model of Actions

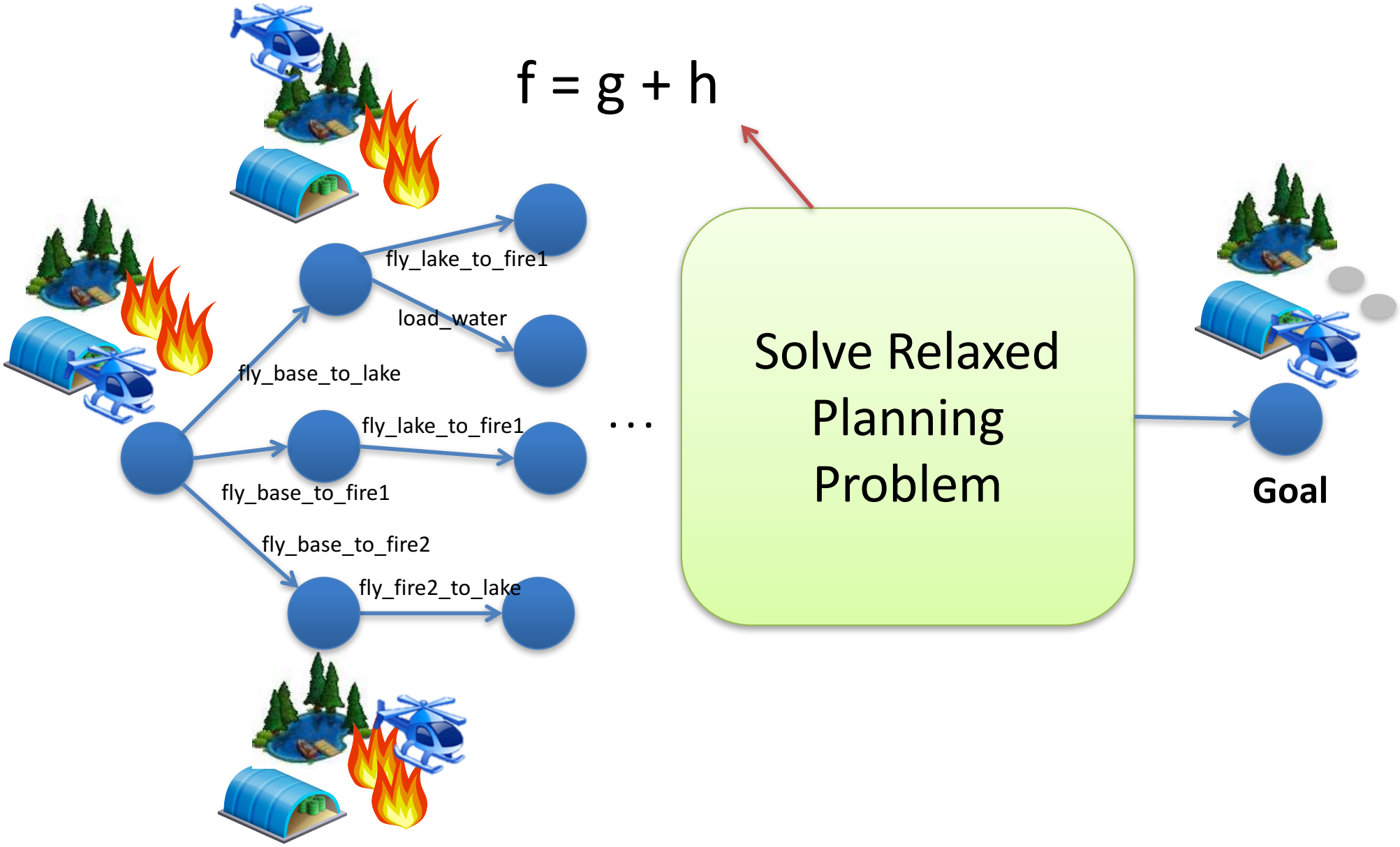
```
class UAV {
  Roadmap location;
  Boolean flying;
  Boolean loaded;

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

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

  primitive method load_water(Lake lakespot)
    ((flying == yes) && (loaded == no) && (lakespot.location == location)) => loaded == yes;

  primitive method drop_water_high_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes) && (firespot.location == location) && (firespot == high))
     => ((loaded == no) && (firespot == medium));

  primitive method drop_water_low_altitude(Fire firespot)
    ((flying == yes) && (loaded == yes) && (firespot.location == location) && (firespot == medium))
     => ((loaded == no) && (firespot == out));

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

```
class Lake {
   Roadmap location;
}

class Fire{
   initial value high;
   value medium;
   value out;
   Roadmap location;
}
```

Simulation Testing: Firefighting Scenario



Decision-making algorithms: Activity Planning (W), Path Planning (T)



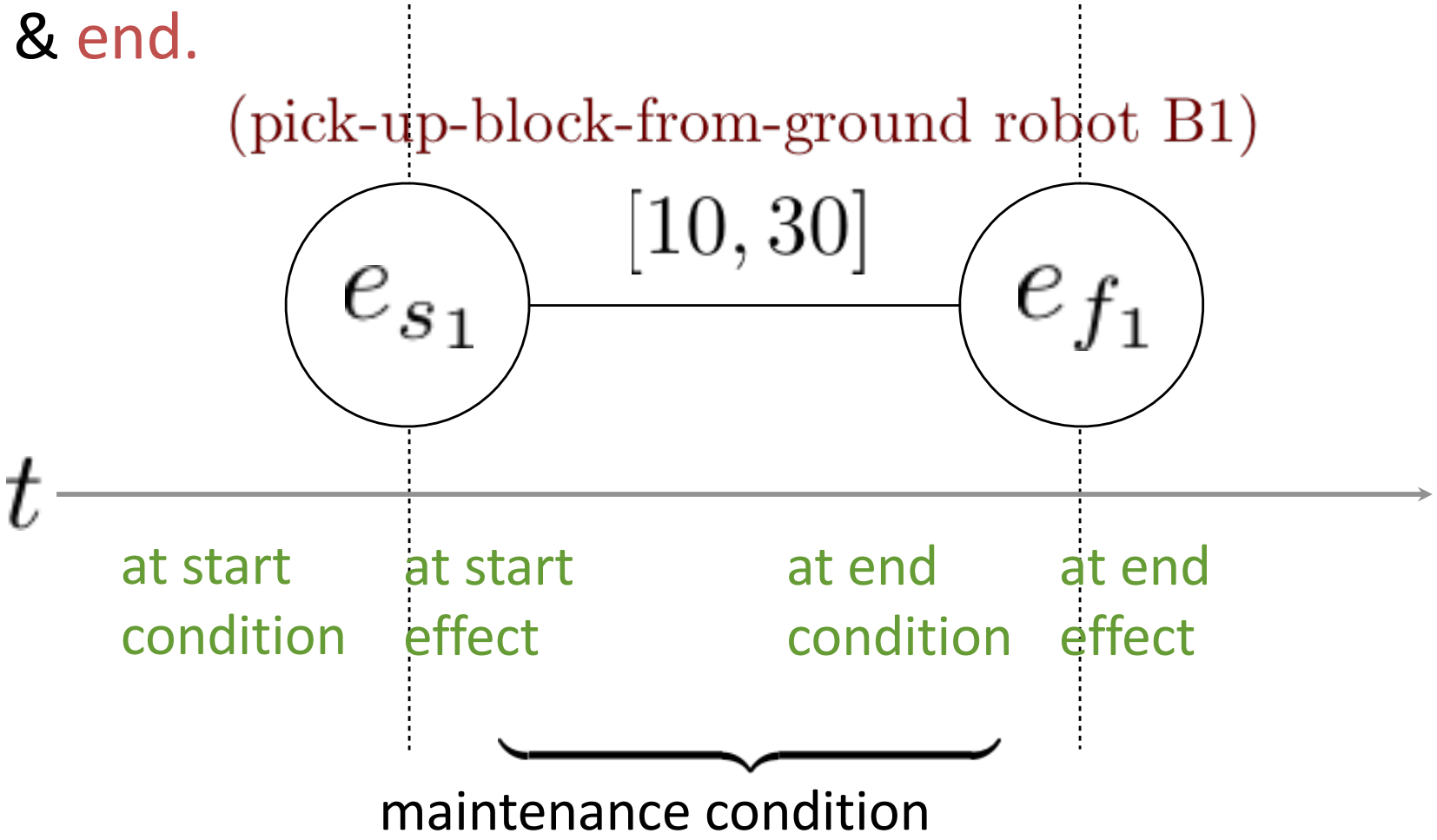
Action Models are Instances of PDDL (Plan Domain Description Language)

```
(:durative-action pick-up-block

    :parameters (?r - robot ?t ?b - block)

    :duration  (and (>= ?duration 10) (<= ?duration 30))

    :condition (and (at start (clear-above ?t))
                    (at start (empty-gripper ?r))
                    (over all (can-reach ?r ?t))
                    (at start (on ?t ?b)))

    :effect    (and (at end (not (empty-gripper ?r)))
                    (at end (not (on ?t ?b)))
                    (at end (holding ?r ?t))
                    (at end (clear-above ?b))
                    (at end (not (clear-above ?t))))
)
```

Primitive Action Model

Temporal actions modeled

- with conditions & effects,
- at start & end.



Outline

- Summer School in a Nutshell
- Architectures for Autonomy
- **Programming Cognitive Robots**
  - Programs that make choices
  - Programs on state
  - **Programs that monitor**

In an Uncertain World
Task Executives need to be Suspicious

"something unexpected happens"
= not modeled in plan!

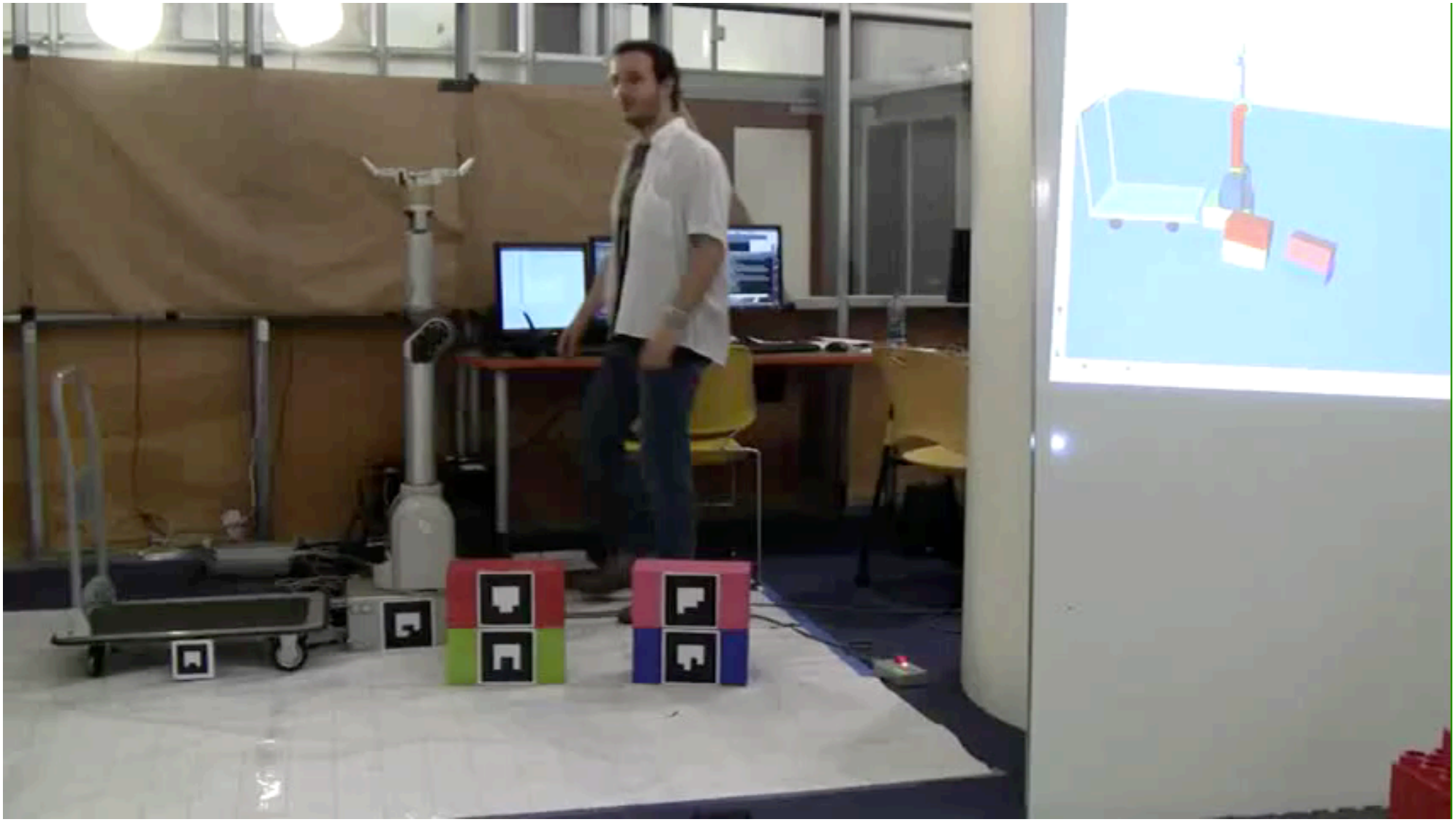
How should we react?

Languages that support Self-Monitoring

```
(Define-rap (attach-at-site ?thesite)
  (succeed (docked ?thesite))
  (method
    (context (ferrous-hull ?thesite))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (t2 (magnetically-attach ?thesite)
            (wait-for (docked ?thesite)))))))
  (method
    (context (not (ferrous-hull ?thesite)))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (t2 (grip-attach ?thesite)
            (wait-for (docked ?thesite))))))
```

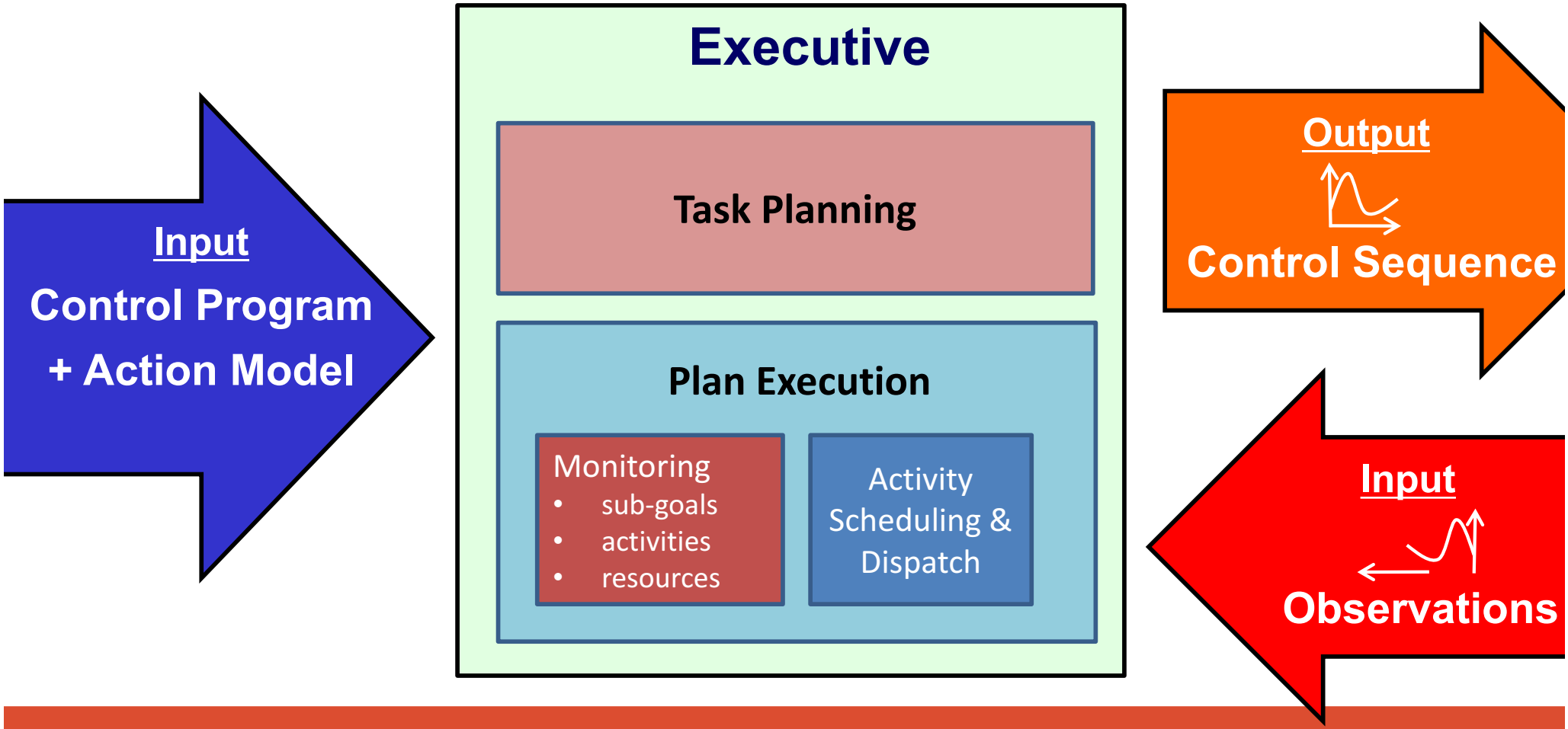
- **RAPS (Firby)**
- **ESL (Gat)**
- **RMPL (Williams)**
- **. . .**

Self-Monitoring Systems

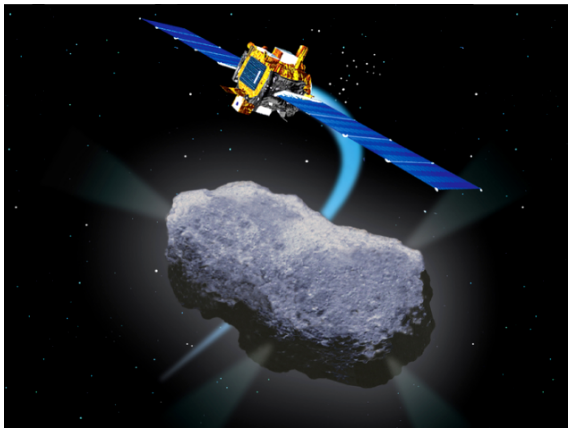


A Suspicious Task Executive Closes the Loop on Goals

- **Monitors**: detects as soon as sub-goals go wrong.
- **Adapts**: fixes the problem.



Summer School is about Creating Goal-directed Systems



1. Decision-making components for generating, refining, executing and monitoring plans on line (15 Tutorials).
2. Architectures and principles for creating goal-directed systems from decision-making components
   (3T, Remote Agent, Claraty, MDS, Idea, Enterprise/ROS, ROSplan).
3. Dynamic languages, elevated to the goal-level through partial specification and operation on state
   (PRS, RAPS, TCC, Golog, RMPL).

QUESTIONS?