AAAI-17 Tutorial on Planning and Robotics

Michael Cashmore **Daniele Magazzeni**

King's College London

KING'S College LONDON

AAAI-17

4 February 2017

San Francisco – California - USA

Outline of the Tutorial

- **What is AI Planning?**
- **Planning for Persistent Autonomy**
- **ROSPlan I: Planning with ROS**

Coffee (15.30-16.00)

- **ROSPlan II: Planning with Opportunities and HRI**
- **Open challenges**

Artificial Intelligence Planning Group at King's

We focus on planning for real applications:

- Autonomous **Underwater** Vehicles
- **Energy** Technology
- Autonomous **Drones** and UAVs
- Ocean Liners
- Multiple **Battery** System Management
- **Hybrid Vehicles**
- Smart Buildings
- Air Traffic Control and Plane Taxiing
- **Urban Traffic Control**

Plan-based Policies for Efficient Multiple Battery Load Management. JAIR 2012

Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach. AAAI 2016.

Solving Realistic Unit Commitment Problems Using Temporal Planning: Challenges and Solutions. ICAPS 2016

Focus of Our Research

Rich Planning Models

We are pushing the research on planning with complex domains

- **PDDL+ modelling**
- **Planners (UPMurphi, DiNO, SMTPlan+)**
- **Policy learning framework**
- **Planning with external solvers**

Validation

We explore the links between planning and verification

- **Plan validation (VAL)**
- **Plan robustness evaluation**
- **Domain validation**

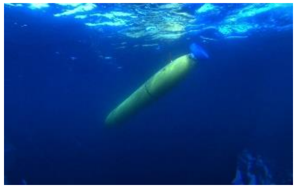
Planning with Robots

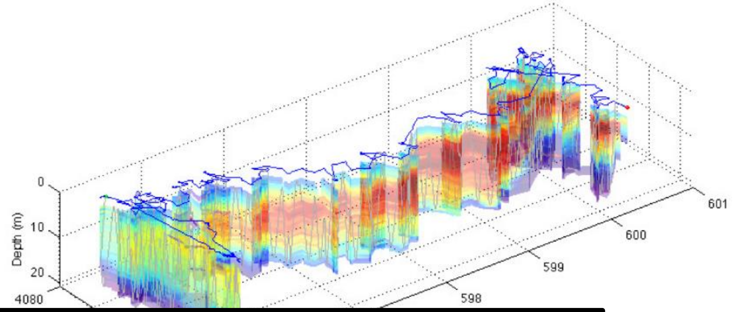
Persistent Autonomy
ROSPlan

Planning with Robots

Planning for Persistent Underwater Autonomy

Policy Learning for Autonomous Feature Tracking

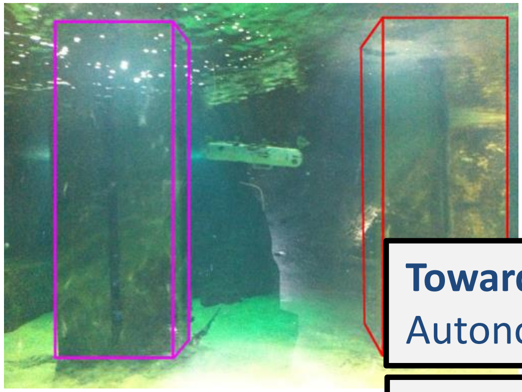


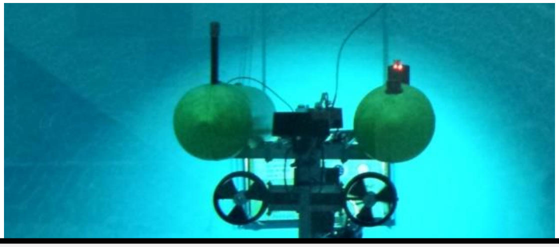


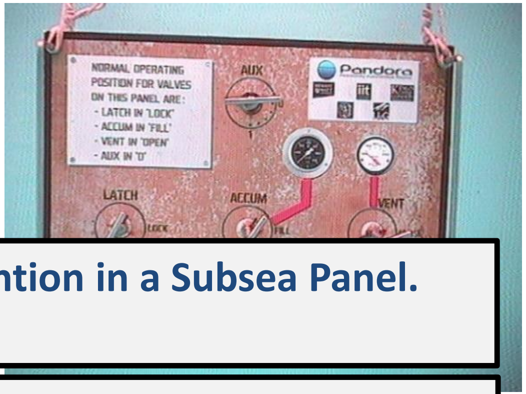
Policy Learning for Autonomous Feature Tracking.
Autonomous Robots (2015)

Autonomous maintenance of submerged oil & gas infrastructures

EU Project PANDORA







Toward Persistent Autonomous Intervention in a Subsea Panel.
Autonomous Robots. (2016)

Opportunistic Planning in Autonomous Underwater Missions.
IEEE Transactions on Automation Science and Engineering. (2017)

Planning with Robots

SQUIRREL
Clearing Clutter Bit by Bit

Robot interacting with children in a toy cleaning scenario

-localisation and navigation in a crowded and changing scene
-iterative task planning in an open world
-engaging with multiple users in a dynamic collaborative task

Short-Term Human Robot Interaction through Conditional Planning and Execution. ICAPS 2017.

Robotics Receptionist at King's College
Multi-Robot Coordination

ERGO

Goal: to deliver an advanced yet flexible **space autonomous software** framework/system suitable for single and/or collaborative space robotic missions

Artificial Intelligence Planning

Artificial Intelligence Planning

Planning is about determining actions *before* doing them, anticipating the things that will need to be done and preparing for them.

If you have a goal to achieve and to do so you need to decide what to do, when to do it and what to use, then that’s *planning.*

Planning is usually done by (teams of) humans: automated planning is for when this job needs to be done fast, frequently, or is too complicated for humans.

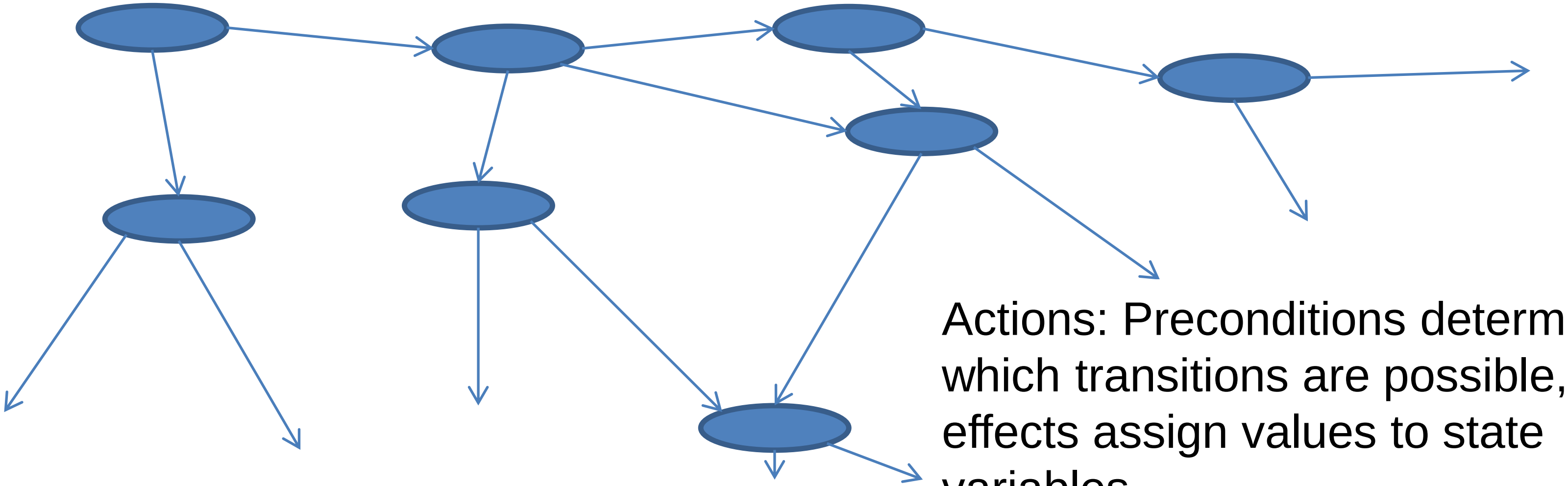
Where there is money to be made, pollution to reduce, productivity to increase, resources to be managed, *planning can do it*.

A *planner* uses a model of an application domain and a description of a specific problem (initial state and goals) and generates a plan. Powerful *heuristics* are used to guide the search in huge state spaces.

AI Planning

- Use a *model* of the world in order to predict and anticipate its behaviour in order to choose actions that will lead to desirable states

- Assume the world can be modelled as a finite collection of *state variables* and that actions cause changes in the values of those variables



Actions: Preconditions determine which transitions are possible, effects assign values to state variables

AI Planning

Given

- An **initial state**: a set of propositions and assignments to numeric variables,
  - E.g. (at rover waypoint1) (= (energy rover) 10)
- A **goal**: a desired set of propositions/assignments,
  - E.g.   (at rover waypoint4) (have-soil-sample waypoint3)
- A **set of actions** each with:
  - Preconditions on execution;
  - Effects that describe how the worl changes upon their execution

```
(:action navigate
 :parameters (?r - rover ?x ?y - waypoint)
 :precondition (and (available ?r)
                    (at ?r ?x)
                    (visible ?x ?y)
                    (>= (energy ?r) 8))
 :effect (and
           (decrease (energy ?r) 8)
           (not (at ?r ?x ))
           (at ?r ?y)))
```

Find

- A sequence of actions (a **plan**) that when applied in the initial state leads to a state that satisfies the goal condition

PDDL Planning

The **Domain** file contains the actions.

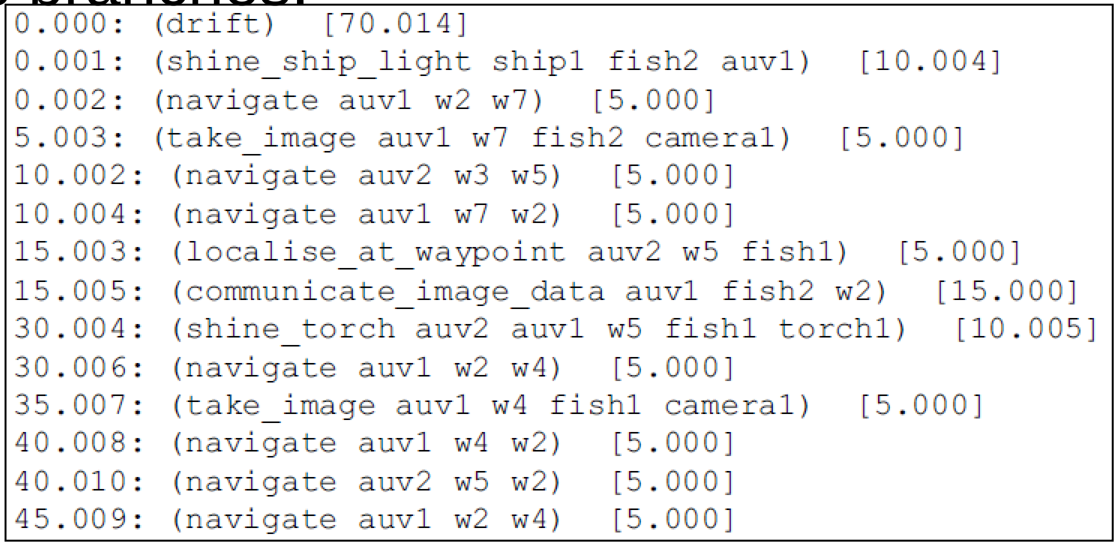
The **Problem** file contains the instance to be solved (i.e., the initial state and the goal state).

Different problems for the same domain.

The planner takes as input the domain D and a problem P,
and produce a plan to solve P.

Planning

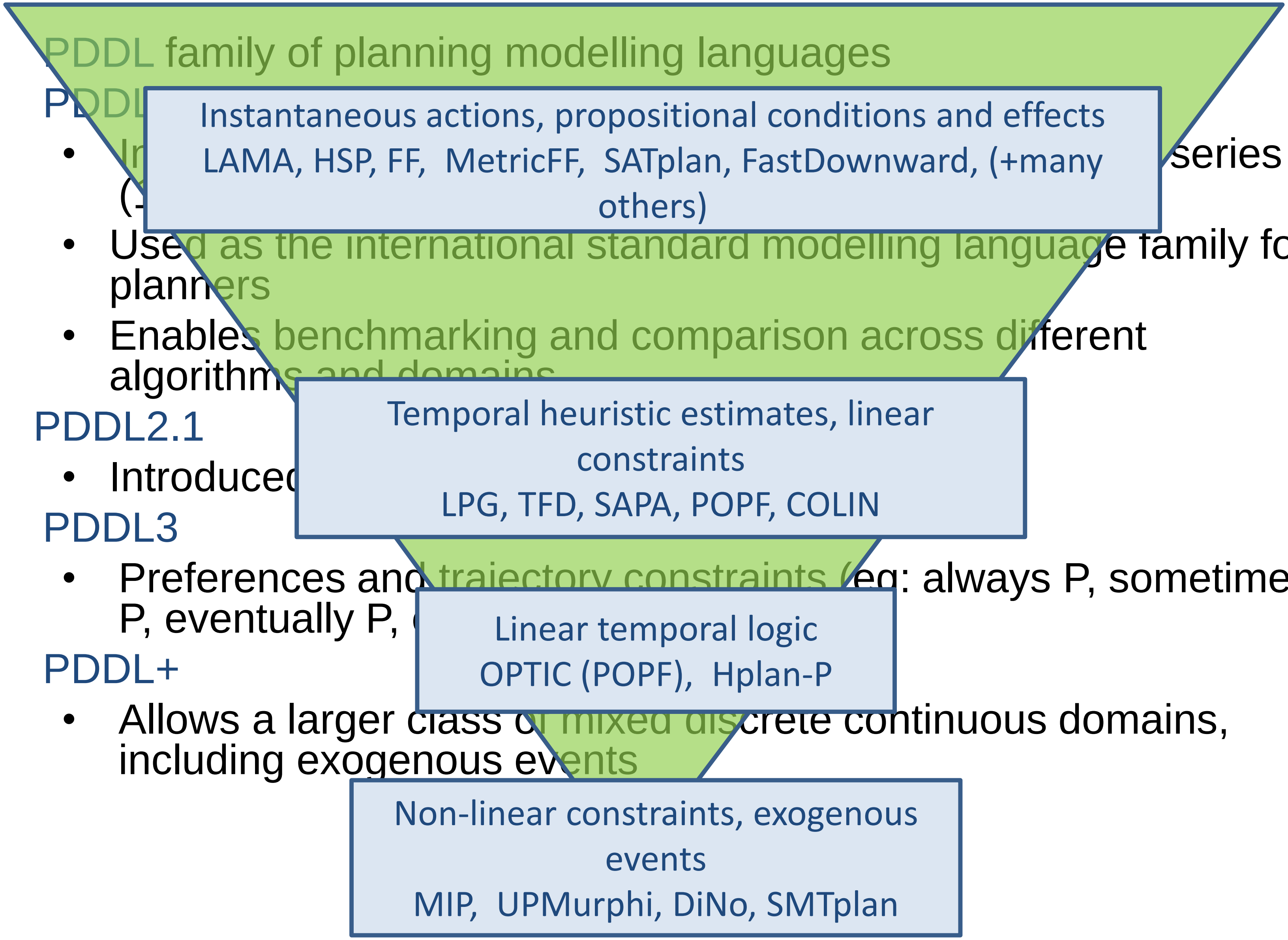
- **Classical** planning: a plan to get to a desirable state that **satisfies some goals**.
- **Optimisation**: minimize/maximise a cost function.
- **Temporal** planning: actions have a duration. *Concurrency, synchronisation, time dependent effects.*
- Planning with **preferences**: hard and soft goals.
- **Conditional** planning**:** actions can perform observations, and the plan contains branches.



```
0.000: (drift)  [70.014]
0.001: (shine_ship_light ship1 fish2 auv1)  [10.004]
0.002: (navigate auv1 w2 w7)  [5.000]
5.003: (take_image auv1 w7 fish2 camera1)  [5.000]
10.002: (navigate auv2 w3 w5)  [5.000]
10.004: (navigate auv1 w7 w2)  [5.000]
15.003: (localise_at_waypoint auv2 w5 fish1)  [5.000]
15.005: (communicate_image_data auv1 fish2 w2)  [15.000]
30.004: (shine_torch auv2 auv1 w5 fish1 torch1)  [10.005]
30.006: (navigate auv1 w2 w4)  [5.000]
35.007: (take_image auv1 w4 fish1 camera1)  [5.000]
40.008: (navigate auv1 w4 w2)  [5.000]
40.010: (navigate auv2 w5 w2)  [5.000]
45.009: (navigate auv1 w2 w4)  [5.000]
```

Planning Problems: Modelling and Solving

- PDDL family of planning modelling languages
- PDDL [illegible]
  - In [illegible] series ([illegible]
  - Used as the international standard modelling language family for planners
  - Enables benchmarking and comparison across different algorithms and domains
- PDDL2.1
  - Introduced [illegible]
- PDDL3
  - Preferences and trajectory constraints (eg: always P, sometimes P, eventually P, [illegible]
- PDDL+
  - Allows a larger class of mixed discrete continuous domains, including exogenous events



Instantaneous actions, propositional conditions and effects
LAMA, HSP, FF, MetricFF, SATplan, FastDownward, (+many others)

Temporal heuristic estimates, linear constraints
LPG, TFD, SAPA, POPF, COLIN

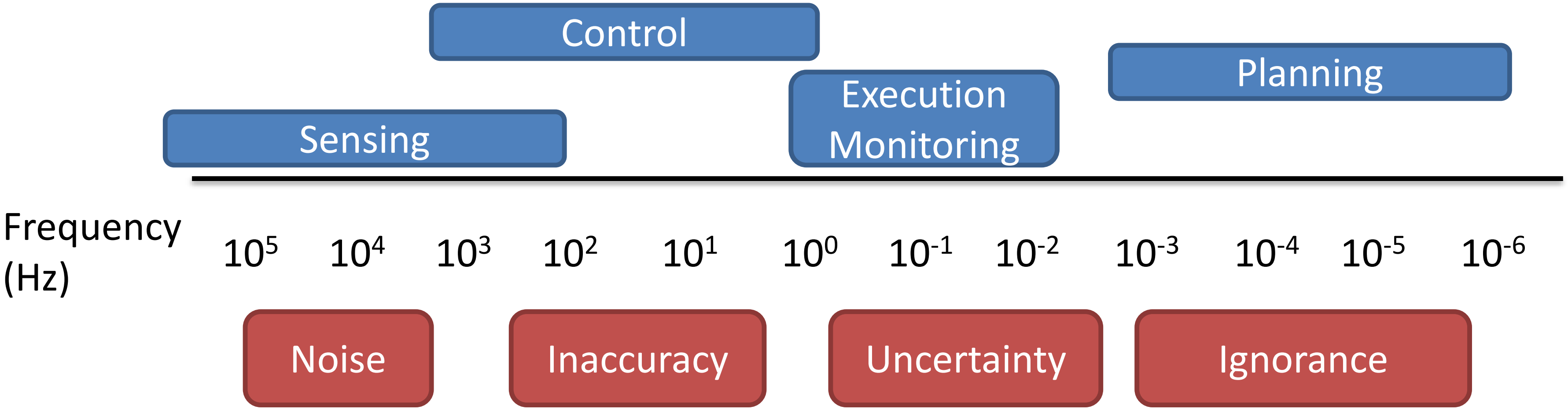
Linear temporal logic
OPTIC (POPF), Hplan-P

Non-linear constraints, exogenous events
MIP, UPMurphi, DiNo, SMTplan

Planning and Control

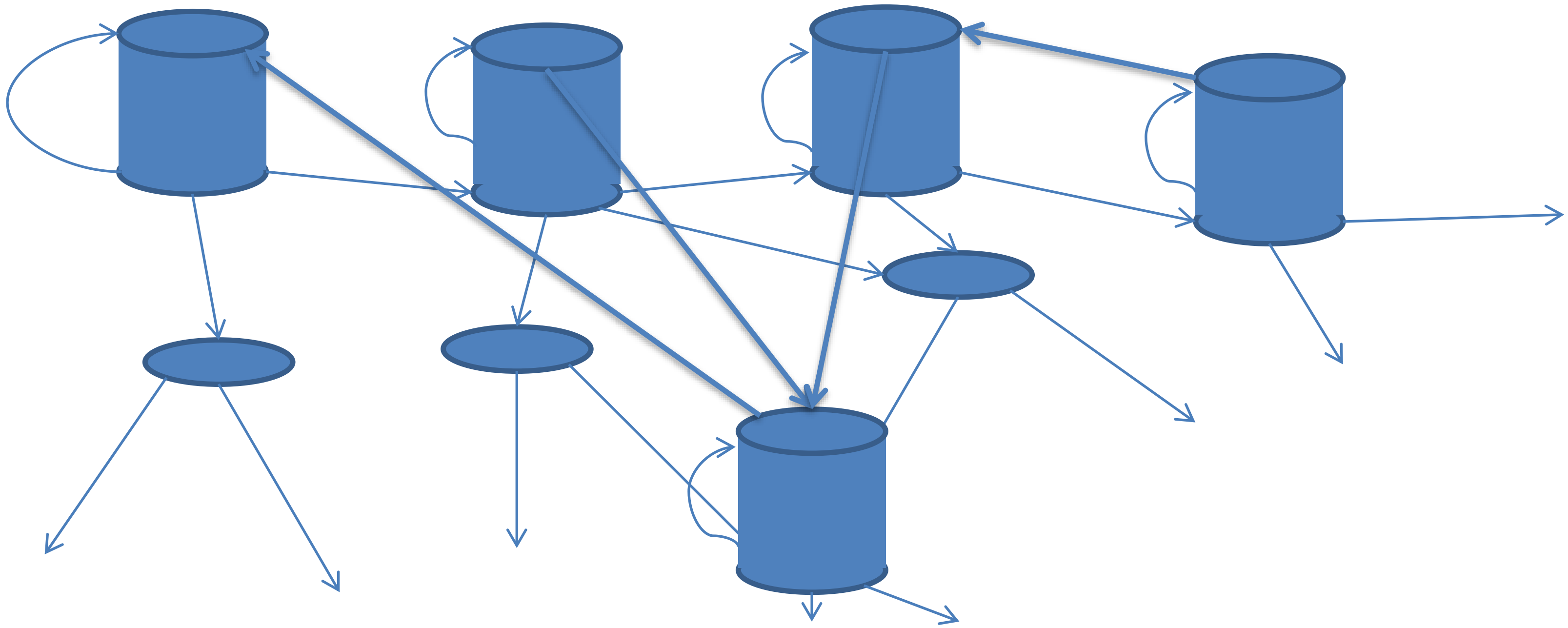
Planning is an AI technology that seeks to select and organise activities in order to achieve specific goals

Plan Dispatch: a controller is responsible for realising each plan action



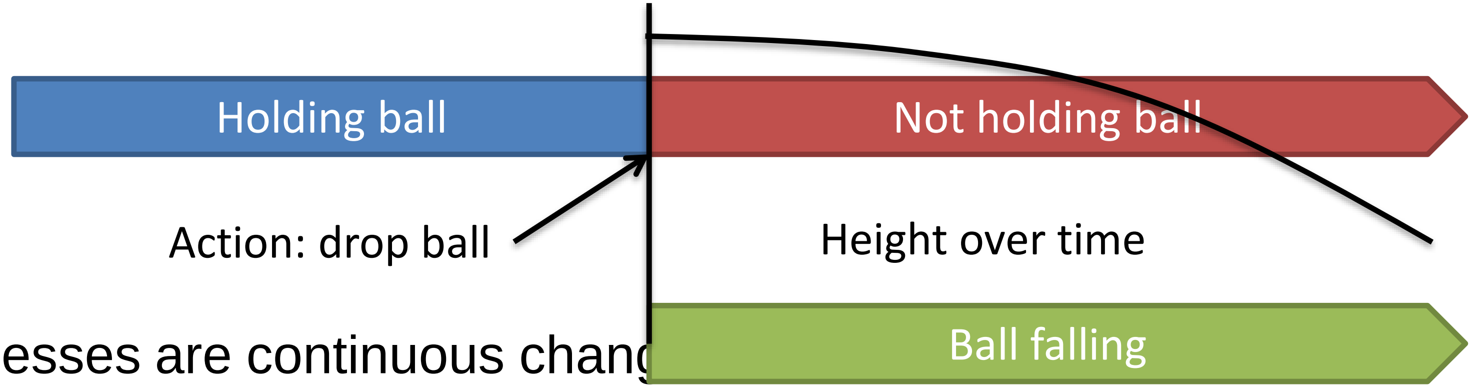
Planning with Time: An Additional Dimension

- Processes mean time spent in states matters



Planning in *Hybrid* Domains

- When actions or events are performed they cause instantaneous changes in the world
  - These are discrete changes to the world state
  - When an action or an event has happened it is over



- Processes are continuous chang
  - Once they start they generate continuous updates in the world state
  - A process will run over time, changing the world at every instant

Modelling Mixed Discrete-Continuous Domains for Planning. (Fox & Long) JAIR 2006

PDDL+: Let it go

- First drop it...

```
(:action release
 :parameters (?b - ball)
 :precondition (and (holding ?b) (= (velocity ?b) 0))
 :effect (and (not (holding ?b))))
```

- Then watch it fall...

```
(:process fall
 :parameters (?b - ball)
 :precondition (and (not (holding ?b)) (>= (height ?b) 0)))
 :effect (and (increase (velocity ?b) (* #t (gravity)))
              (decrease (height ?b) (* #t (velocity ?b)))))
```

- And then?

PDDL+: See it bounce

- Bouncing…

```
(:event bounce
 :parameters (?b - ball)
 :precondition (and  (>= (velocity ?b) 0)
                     (<= (height ?b) 0))
 :effect (and (assign (height ?b) (* -1 (height ?b)))
               (assign (velocity ?b) (* -1 (velocity ?b)))))
```

- Now let’s plan to catch it…

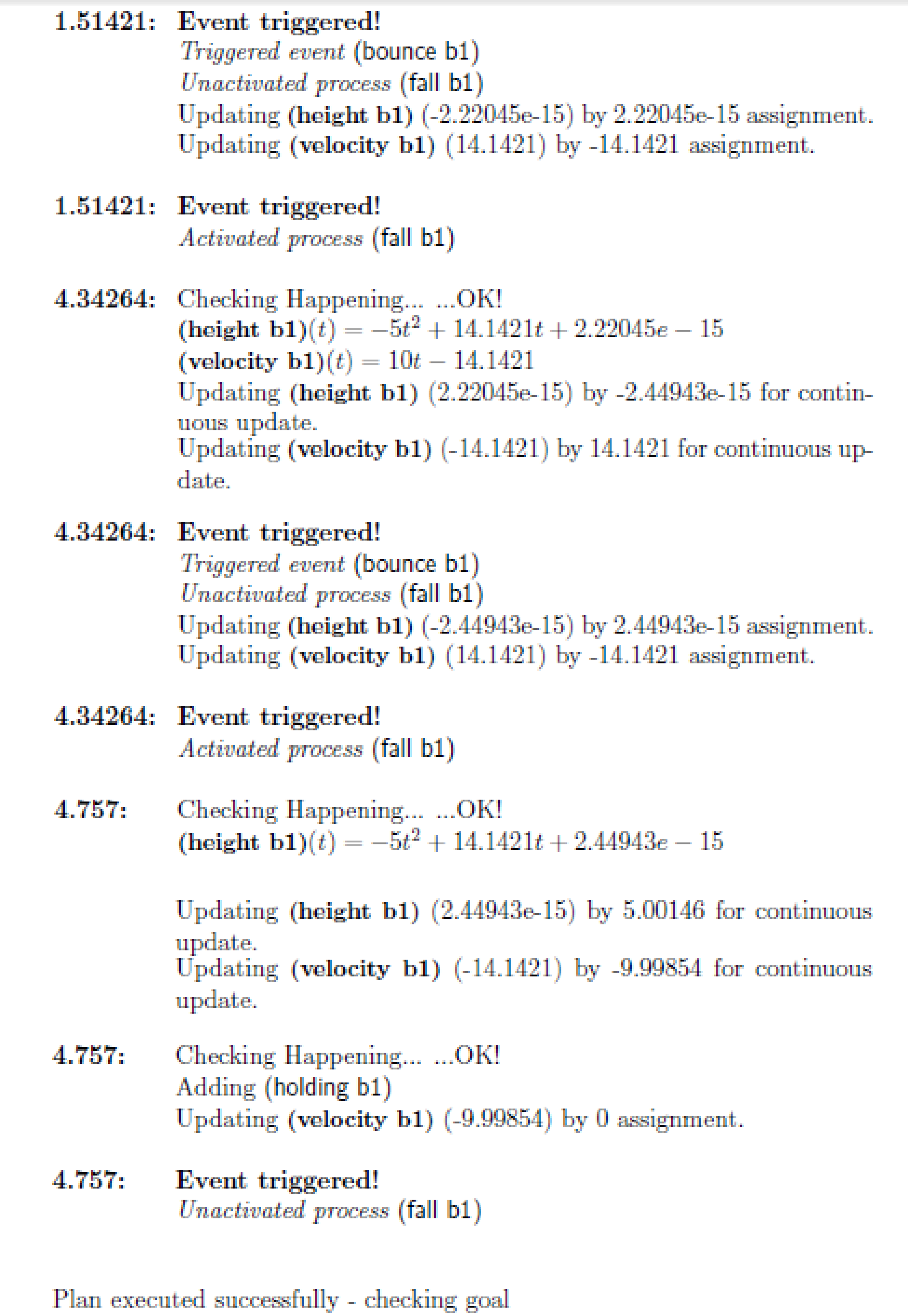
```
(:action catch
 :parameters (?b - ball)
 :precondition (and (>= (height ?b) 5) (<= (height ?b) 5.01))
 :effect (and (holding ?b) (assign (velocity ?b) 0)))
```

A Valid Plan

- Let it bounce, then catch it...

```
0.1:   (release b1)
4.757: (catch b1)
```

- The validator VAL can be used to check plan validity. (https://github.com/KCL-Planning/VAL)



1.51421: Event triggered!
Triggered event (bounce b1)
Unactivated process (fall b1)
Updating **(height b1)** (-2.22045e-15) by 2.22045e-15 assignment.
Updating **(velocity b1)** (14.1421) by -14.1421 assignment.

1.51421: Event triggered!
Activated process (fall b1)

4.34264: Checking Happening... ...OK!
(height b1)$(t) = -5t^2 + 14.1421t + 2.22045e - 15$
(velocity b1)$(t) = 10t - 14.1421$
Updating **(height b1)** (2.22045e-15) by -2.44943e-15 for continuous update.
Updating **(velocity b1)** (-14.1421) by 14.1421 for continuous update.

4.34264: Event triggered!
Triggered event (bounce b1)
Unactivated process (fall b1)
Updating **(height b1)** (-2.44943e-15) by 2.44943e-15 assignment.
Updating **(velocity b1)** (14.1421) by -14.1421 assignment.

4.34264: Event triggered!
Activated process (fall b1)

4.757: Checking Happening... ...OK!
(height b1)$(t) = -5t^2 + 14.1421t + 2.44943e - 15$

Updating **(height b1)** (2.44943e-15) by 5.00146 for continuous update.
Updating **(velocity b1)** (-14.1421) by -9.99854 for continuous update.

4.757: Checking Happening... ...OK!
Adding (holding b1)
Updating **(velocity b1)** (-9.99854) by 0 assignment.

4.757: Event triggered!
Unactivated process (fall b1)

Plan executed successfully - checking goal

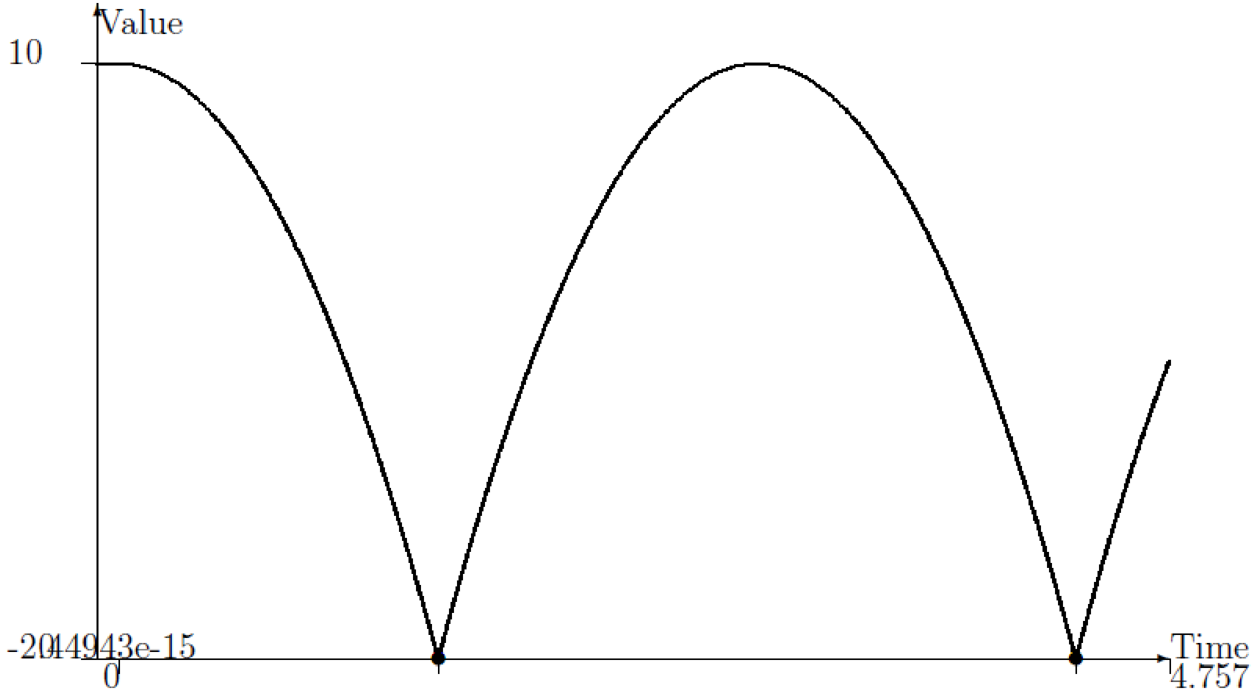


Figure 2.1: Graph of (height b1).

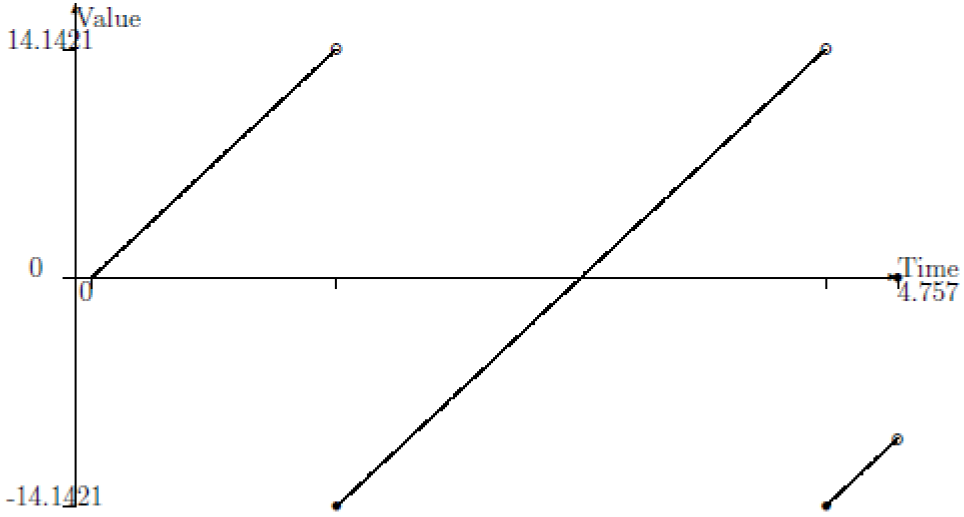


Figure 2.2: Graph of (velocity b1).

Some PDDL+ Planners

- **UPMurphi** (https://github.com/gdellapenna/UPMurphi) [ICAPS'09]

  Based on Discretise and Validate

  (Baseline for adding new heuristics:
  multiple battery management [JAIR'12] or urban traffic control [AAAI'16])

- **DiNo** (http://kcl-planning.github.io/DiNo/) [IJCAI'16]

  Extend UPMurphi with TRPG heuristic for hybrid domains

- **SMTPlan** (http://kcl-planning.github.io/SMTPlan/) [ICAPS'16]

  Based on STM encoding of PDDL+ domains

One more PDDL+ example

Vertical Take-Off Domain

The aircraft takes off vertically and needs to reach a location where stable fixed-wind flight can be achieved.

The aircraft has fans/rotors which generate lift and which can be tilted by 90 degrees to achieve the right velocity both vertically and horizontally.

V-22 Osprey

Vertical Take-Off

```
(:action start_engines
:parameters ()
:precondition (and (not (ascending)) (not (crashed)) (= (altitude) 0) )
:effect (ascending))
```

```
(:process ascent
:parameters ()
:precondition (and (not (crashed)) (ascending) )
:effect (and (increase (altitude) (* #t (- (* (v_
                  (* (angle) 0.0174533) ) 2) ) ) (g)
                  (increase (distance) (* #t (* (v_f
                  (- 40500 (* (angle) (- 180 (angle
```

Timed Initial Fluents

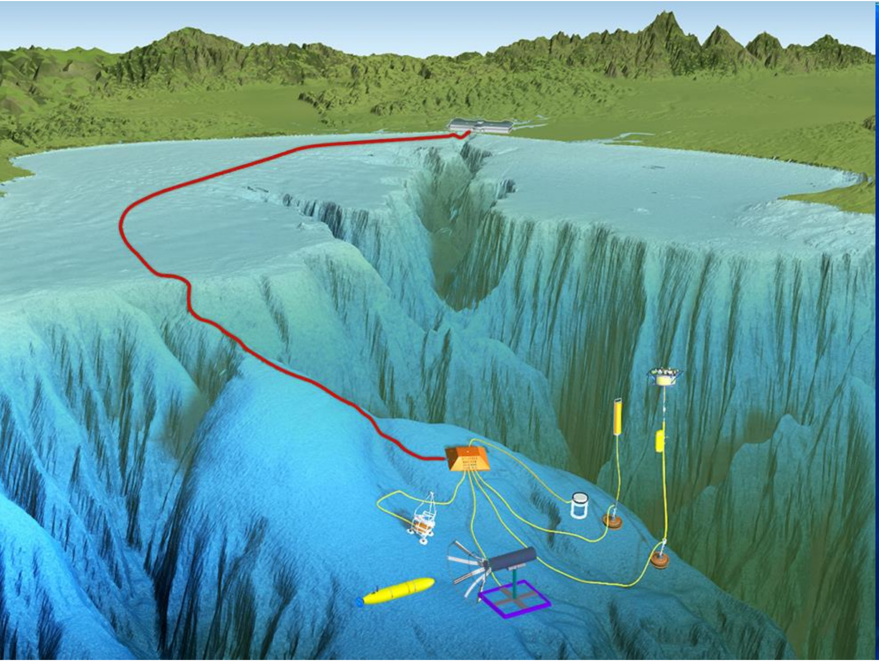
```
(at 5.0 (= (wind_x) 1.3))
(at 5.0 (= (wind_y) 0.2))
(at 9.0 (= (wind_x) -0.5))
(at 9.0 (= (wind_y) 0.3))
.. …
```

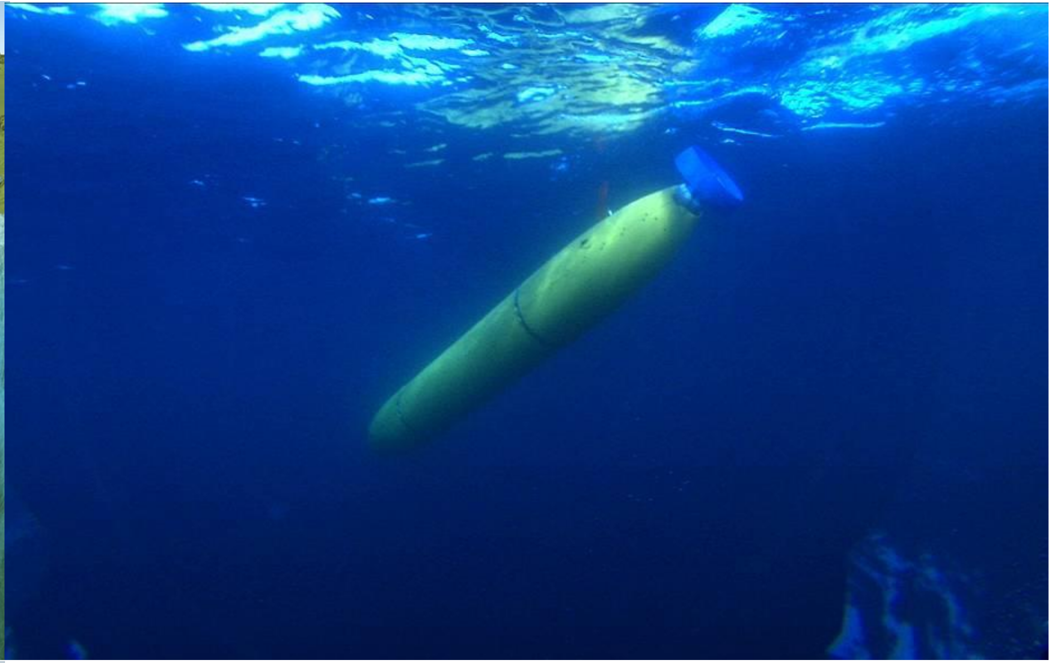
```
(:durative-action increase_angle
:parameters ()
:duration (<= ?duration (- 90 (angle)) )
:condition (and (over all (ascending)) (over all (<= (angle) 90)) (over all (>= (angle) 0)) )
:effect (and (increase (angle) (* #t 1)) ))
```

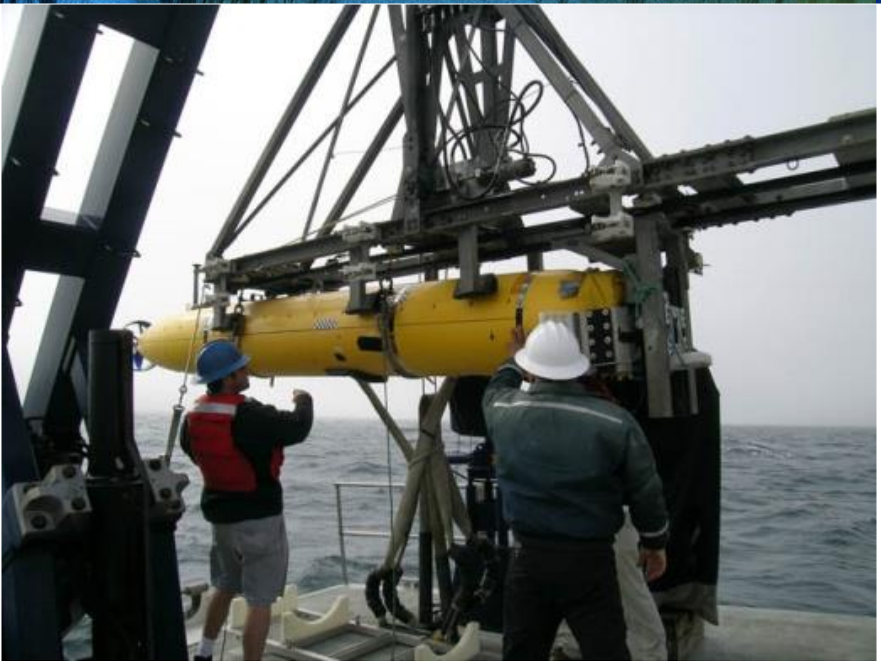
```
(:event crash
:parameters ()
:precondition (and (< (altitude) 0))
:effect ((crashed))
)
```

```
(:process wind
 :parameters ()
 :precondition (and (not (crashed)) (ascending) )
 :effect (and (increase (altitude) (* #t (wind_y) 1)
                   (increase (distance) (* #t (wind_x) 1)))
```

Planning for Persistent Autonomy







Planning on the sea

EU-FP7 PANDORA Persistent Autonomy for AUVs

EU-FP7 PANDORA

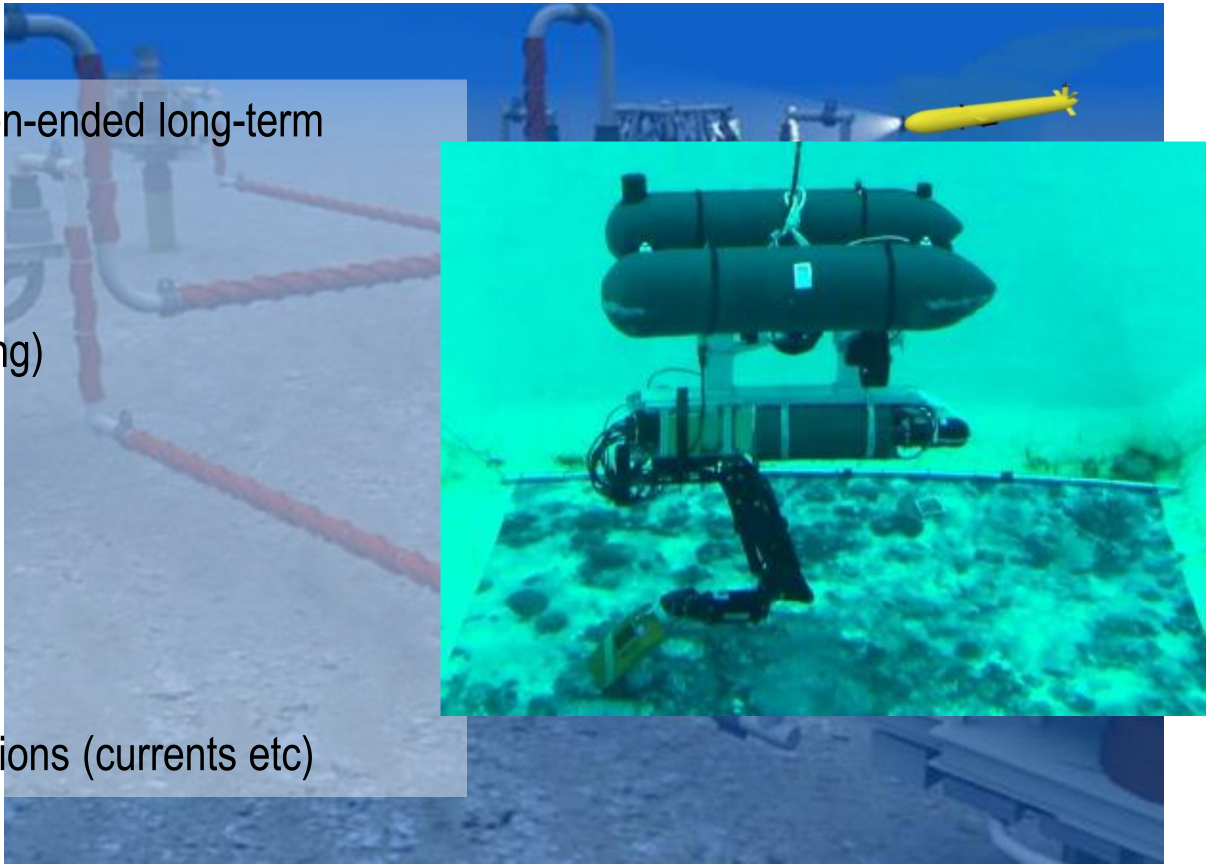
Persistently autonomous inspection and maintenance of an underwater installation, such as an offshore oil and gas field

Tasks performed as part of an open-ended long-term maintenance programme:

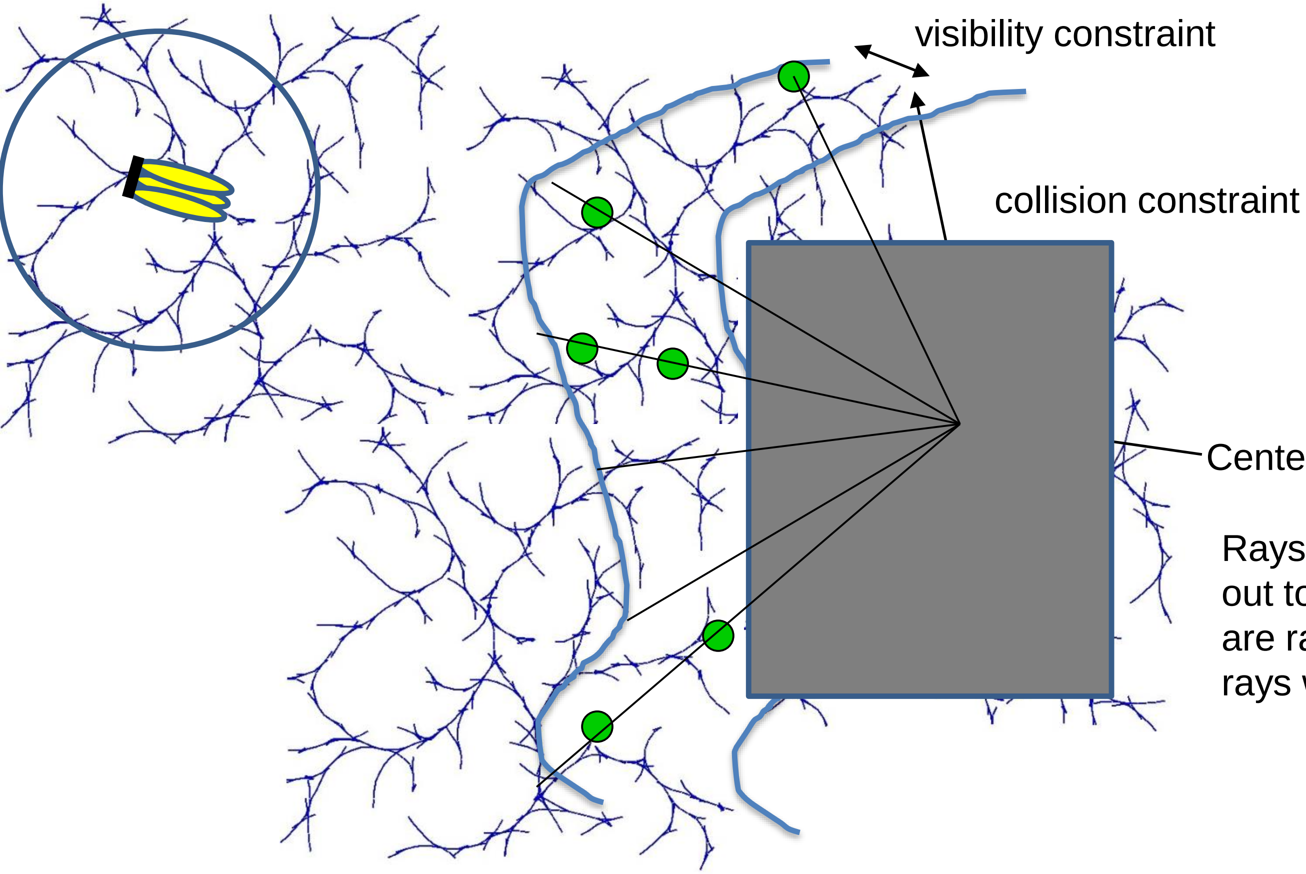
- Observation and inspection
- State-changing (eg valve turning)
- Maintenance and cleaning

Subject to:

- energy constraints
- inspection outcomes
- external requests
- changing environmental conditions (currents etc)



Refining the initial model

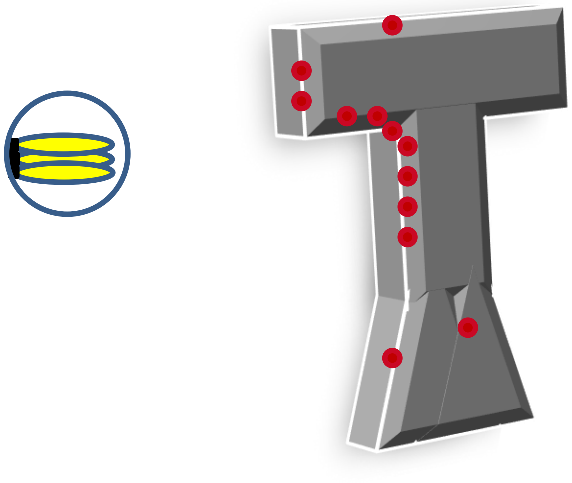


Rays are generated from the CoM out to the visibility band. Waypoints are randomly generated on the rays within the band

New waypoints are generated, requiring model revision, and replanning

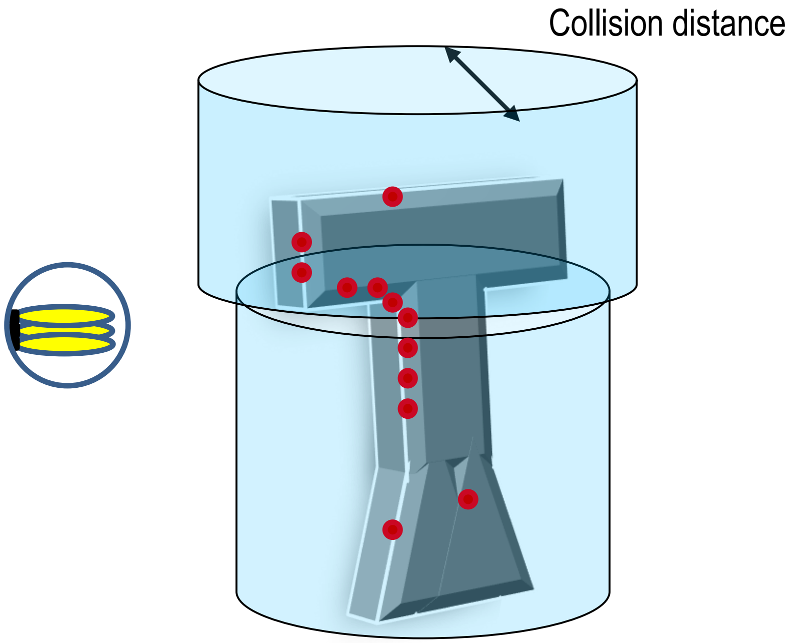
Inspection Task

The PRM selects waypoints from a biased distribution that places more points at good viewing distances from inspection points



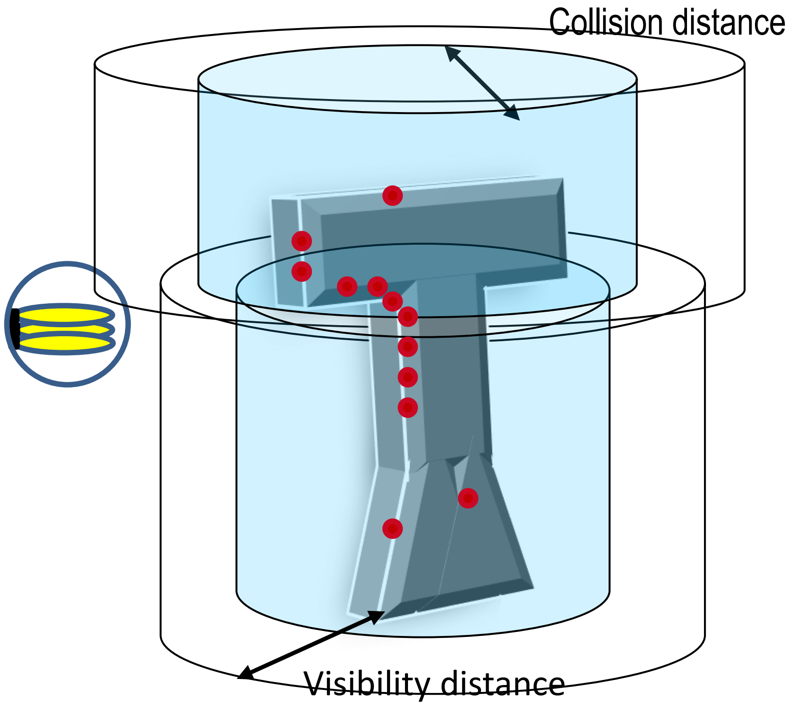
Inspection Task

The PRM selects waypoints from a biased distribution that places more points at good viewing distances from inspection points



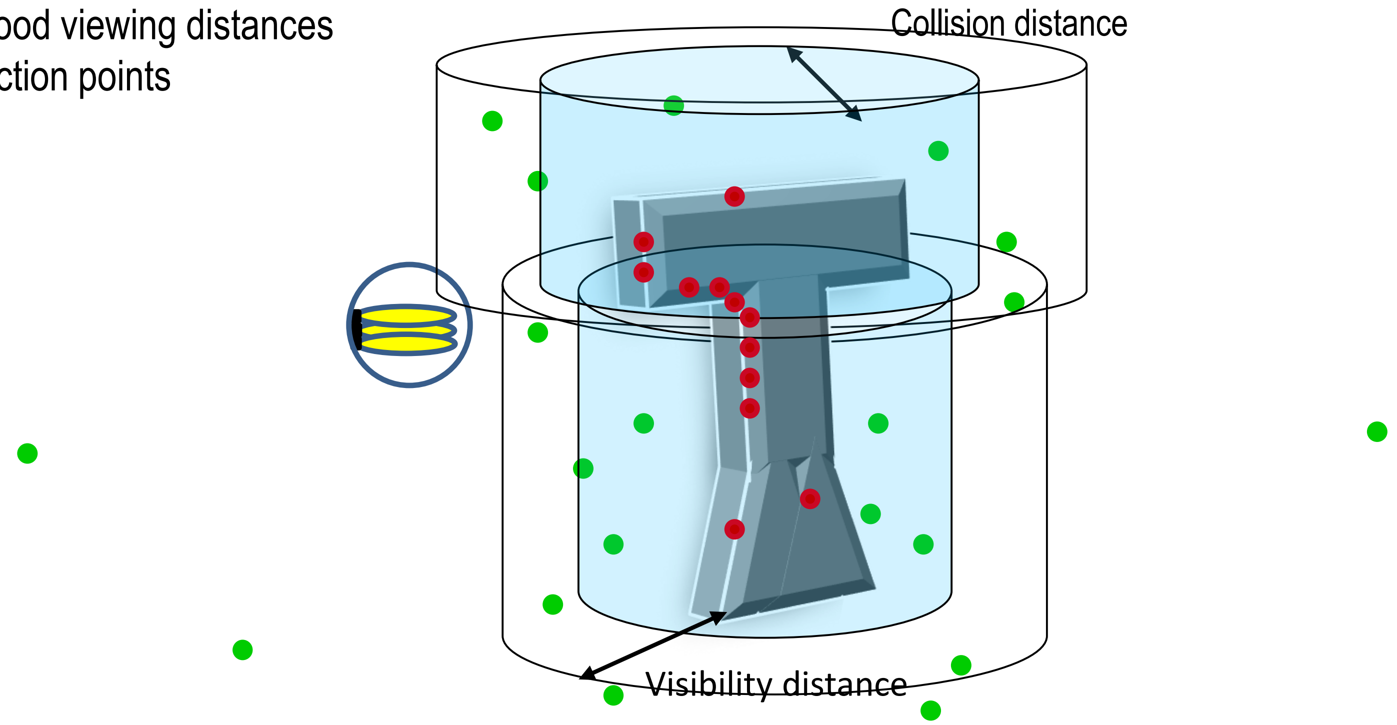
Inspection Task

The PRM selects waypoints from a biased distribution that places more points at good viewing distances from inspection points



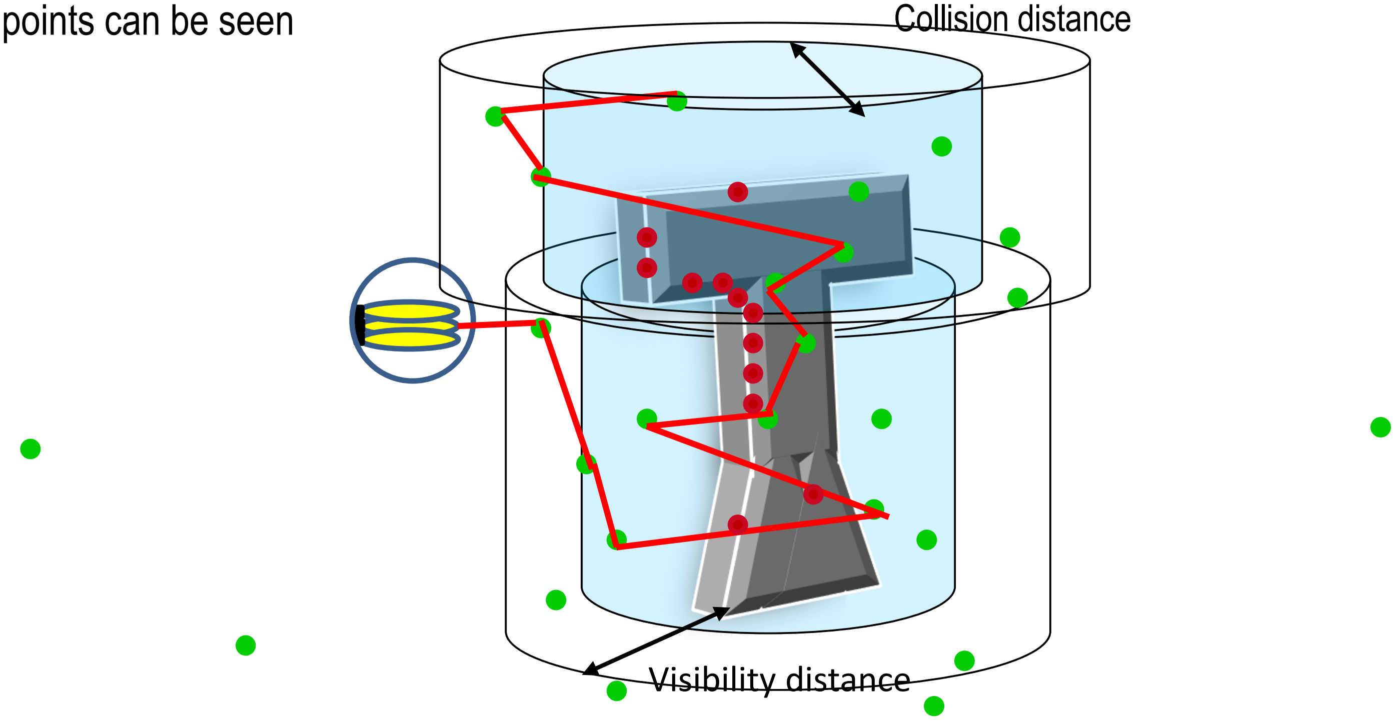
Inspection Task

The PRM selects waypoints from a biased distribution that places more points at good viewing distances from inspection points



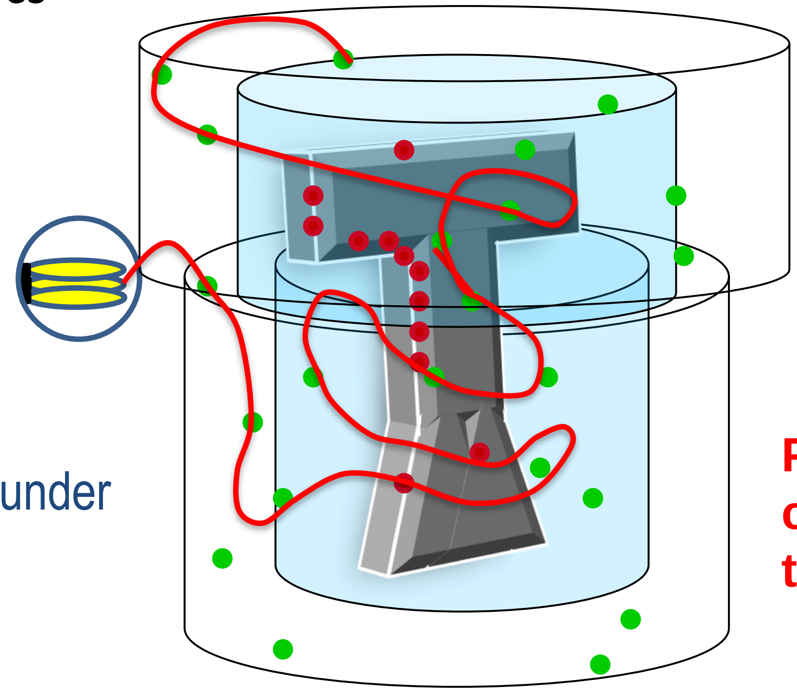
Inspection Task

A path is planned between waypoints from which the inspection points can be seen



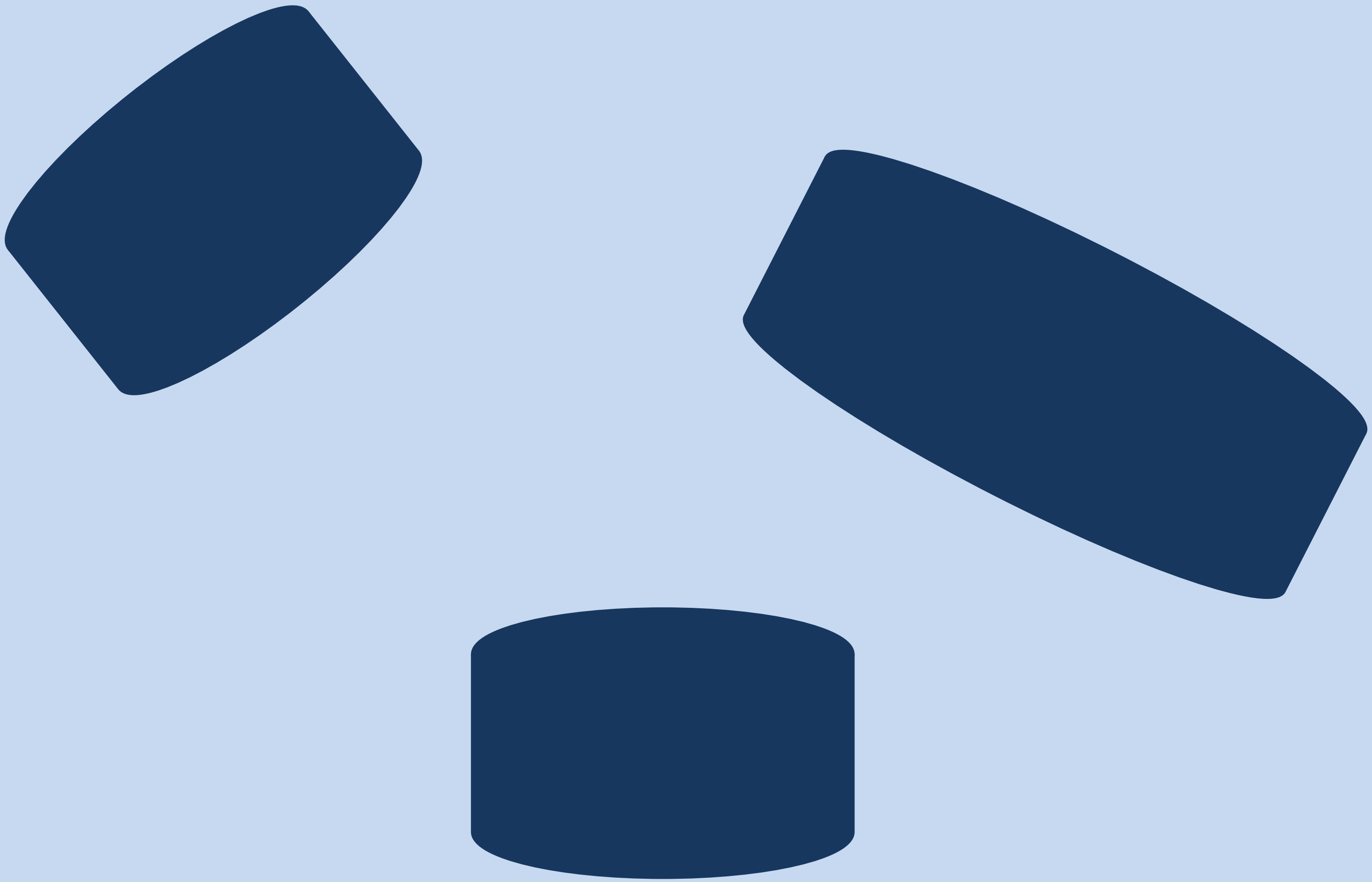
Inspection Task

- We want to achieve inspection task within limited time and energy budget, so efficient paths are important
- Path cost is determined not only by length, but by momentum at start and end and kinematics

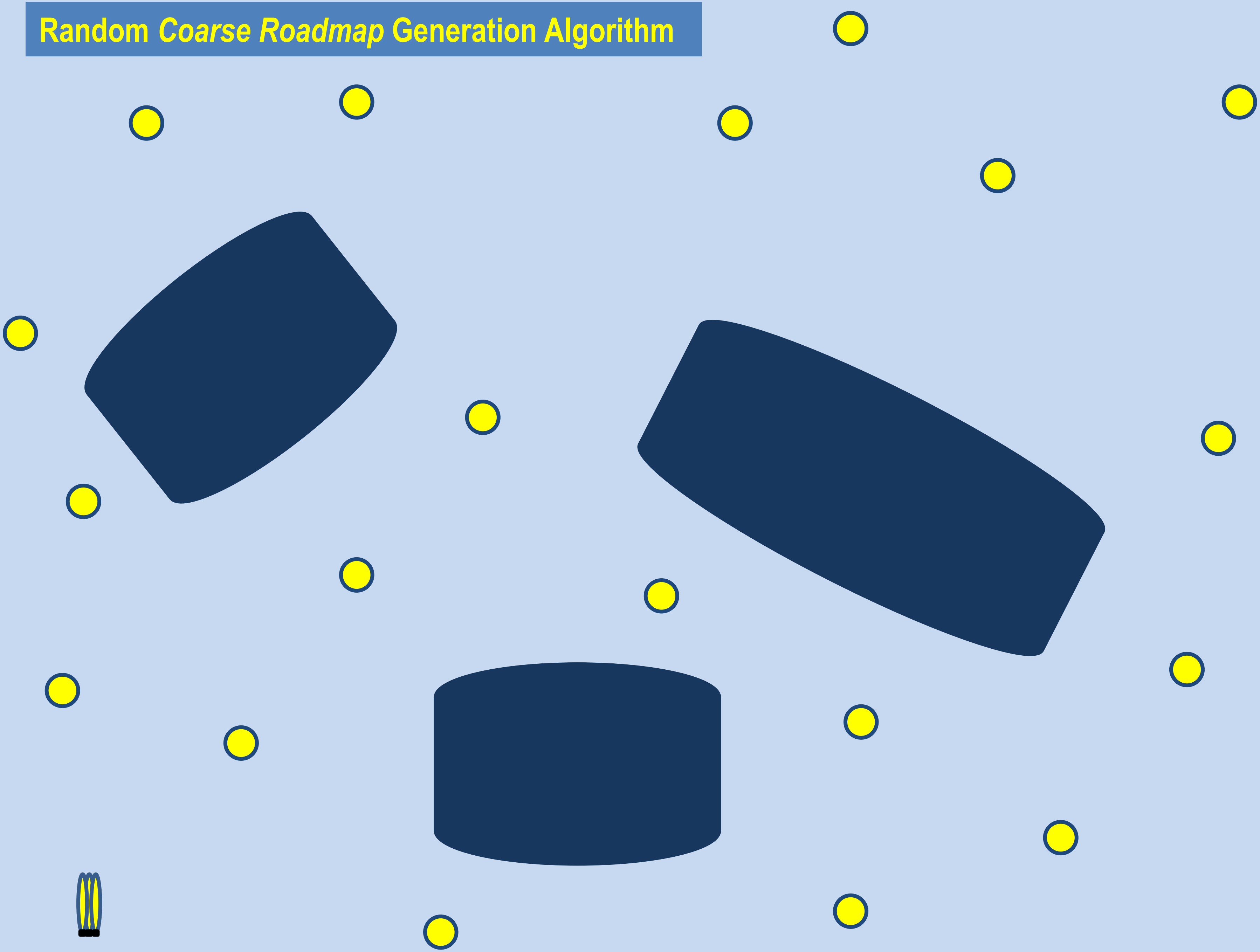


Execution of planned path under kinematic and dynamic constraints

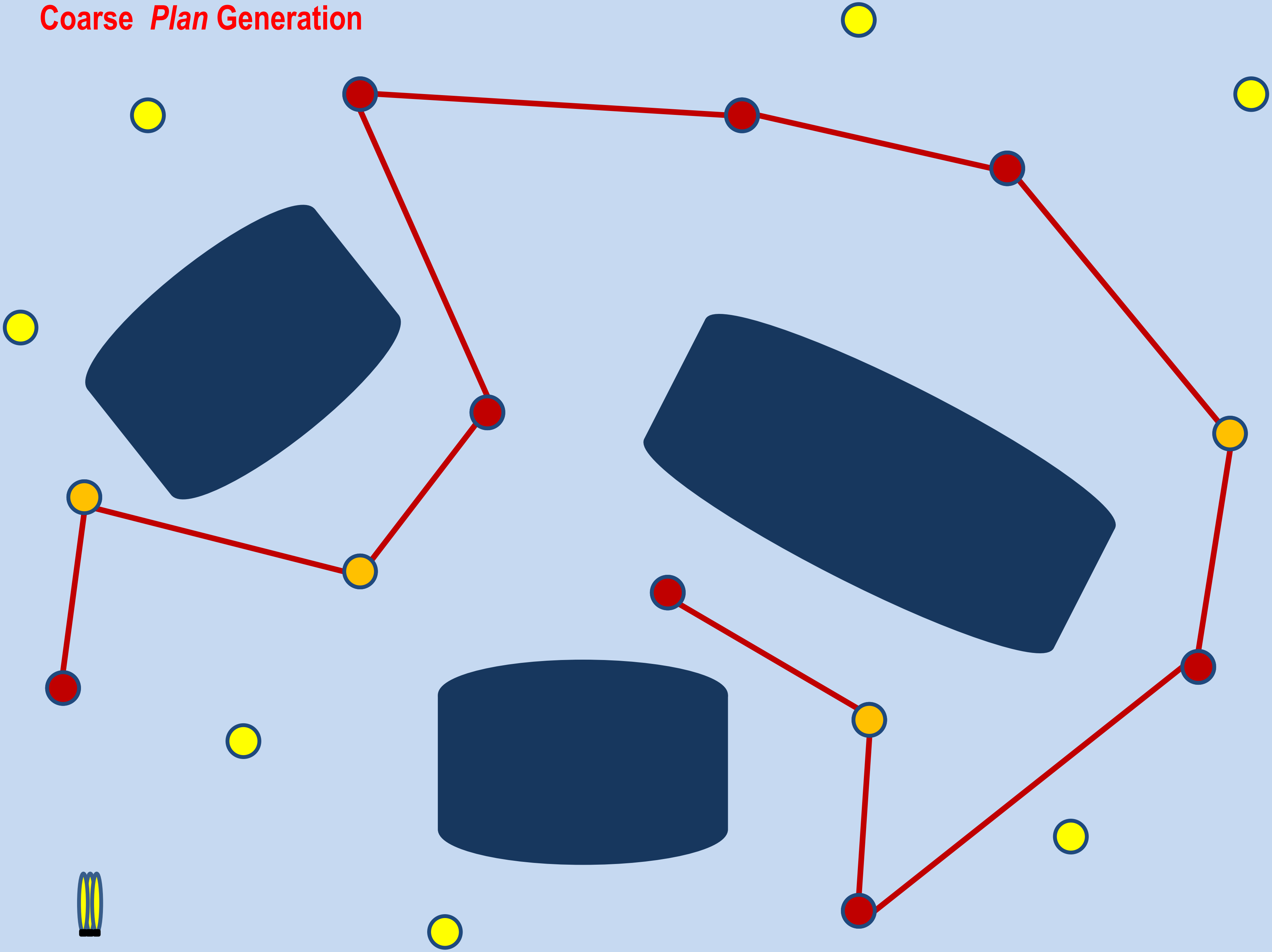
Plans here involve coordinated tasks and time windows



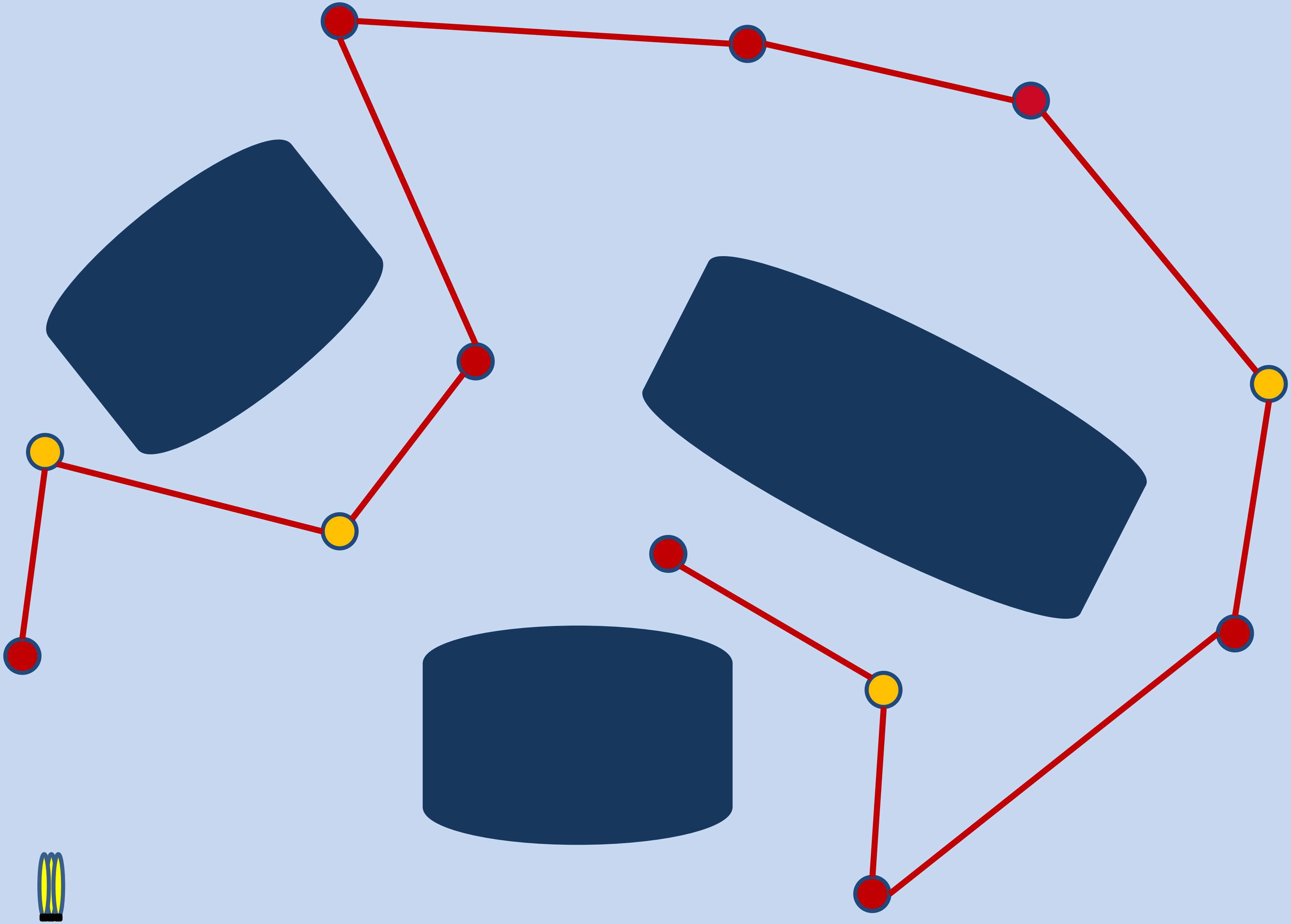
Random *Coarse Roadmap* Generation Algorithm



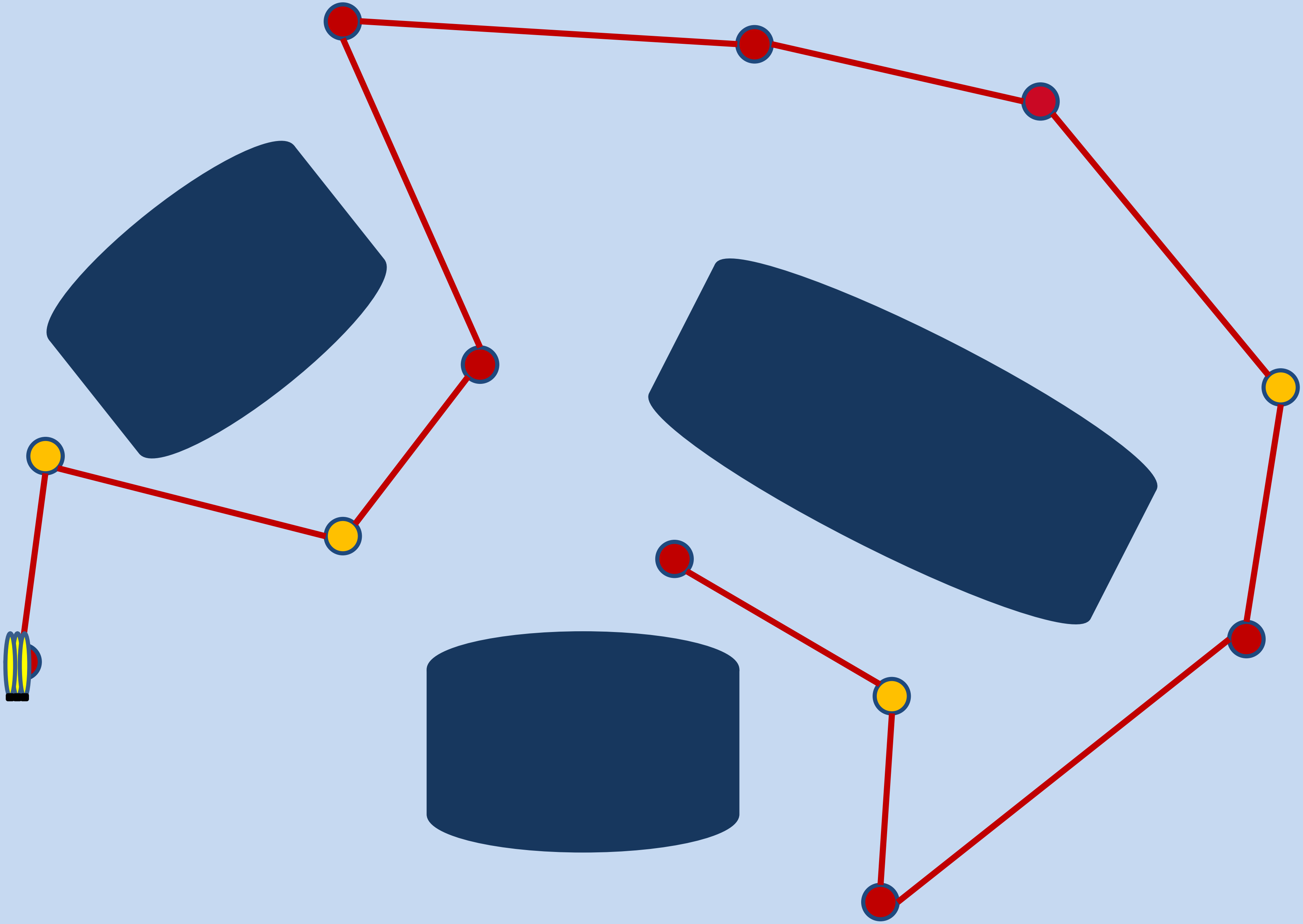
Coarse *Plan* Generation



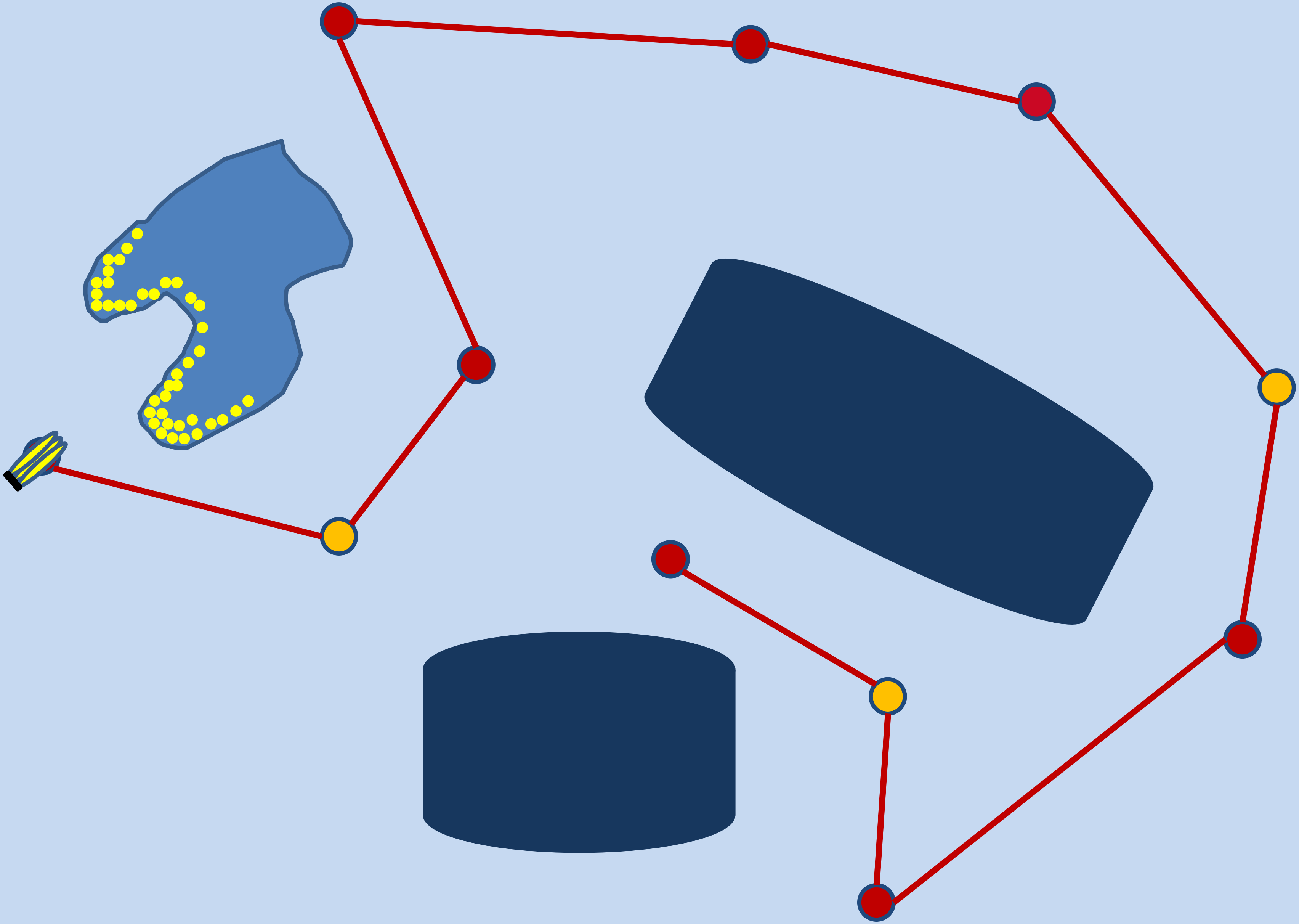
Coarse *Plan* Generation



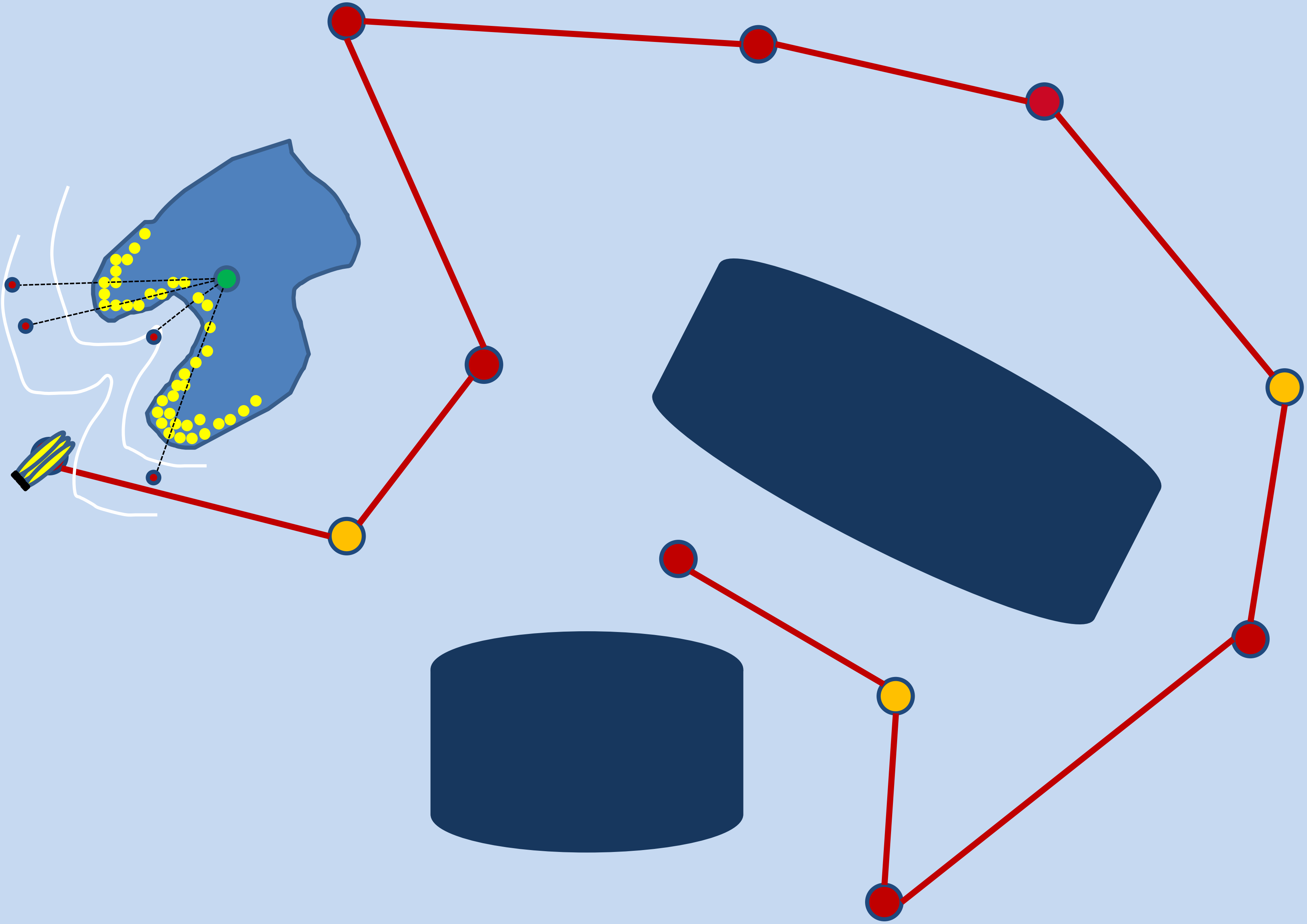
Visibility Points Discovery and REPLANNING



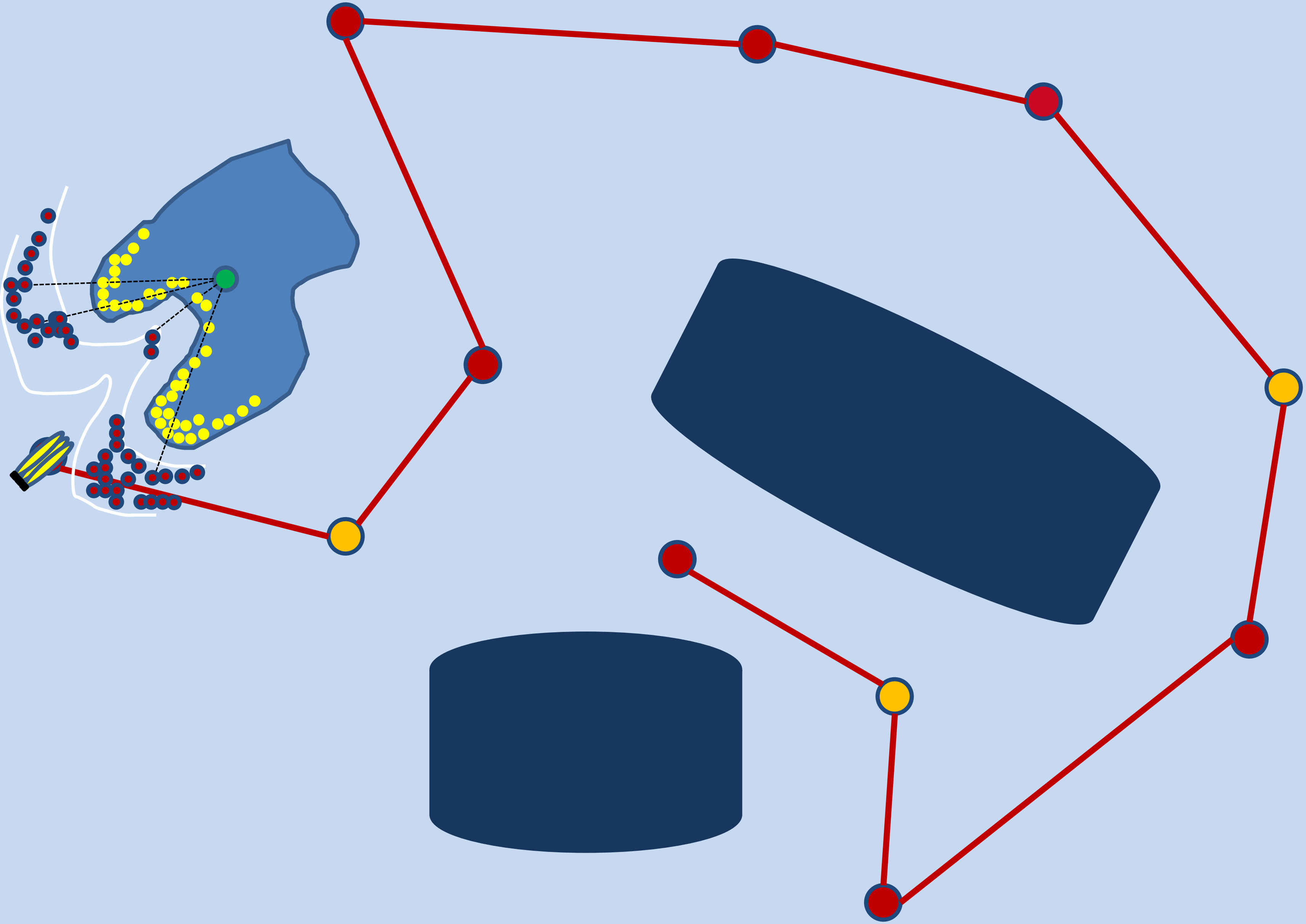
Visibility Points Discovery and REPLANNING



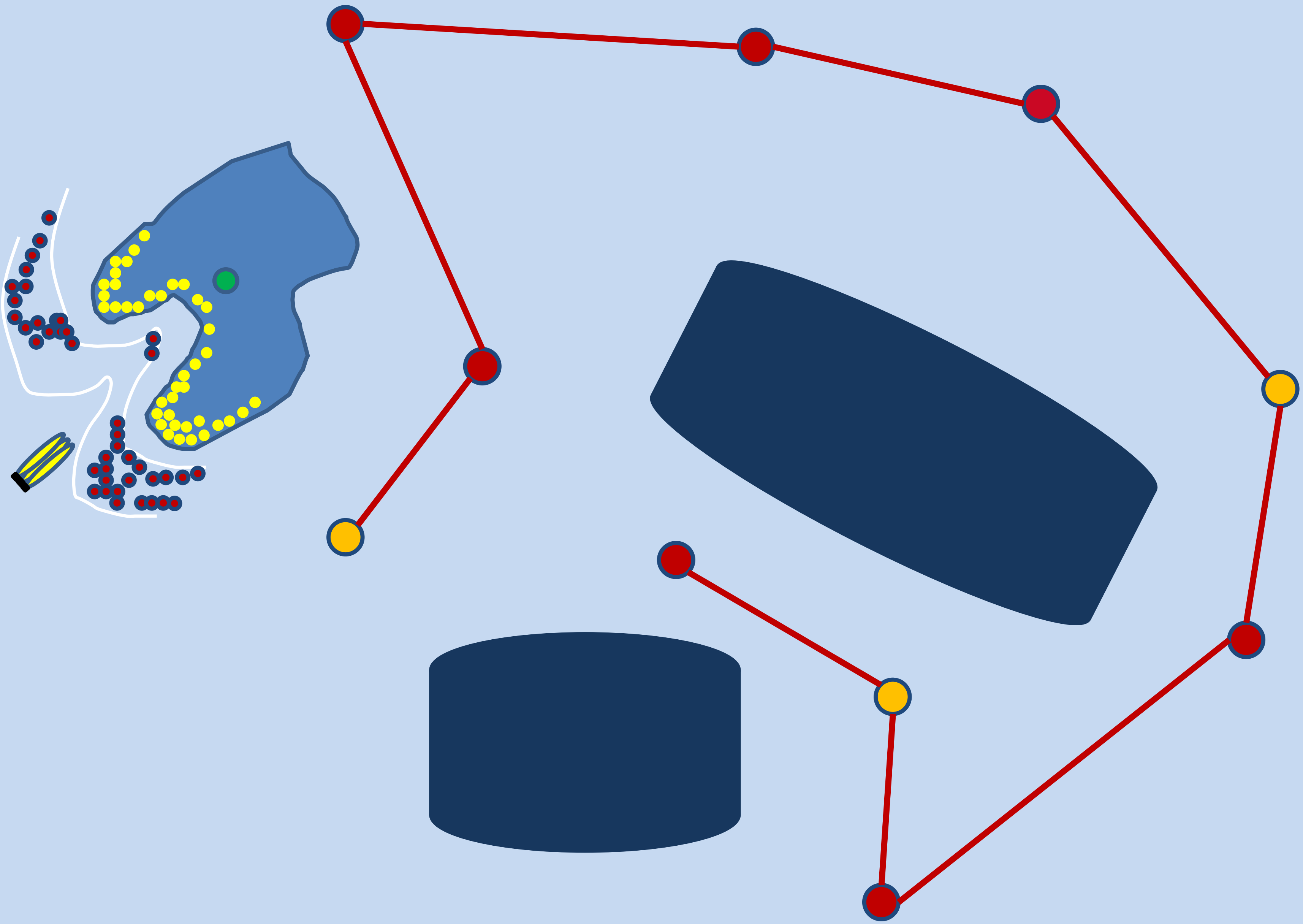
Visibility Points Discovery and REPLANNING



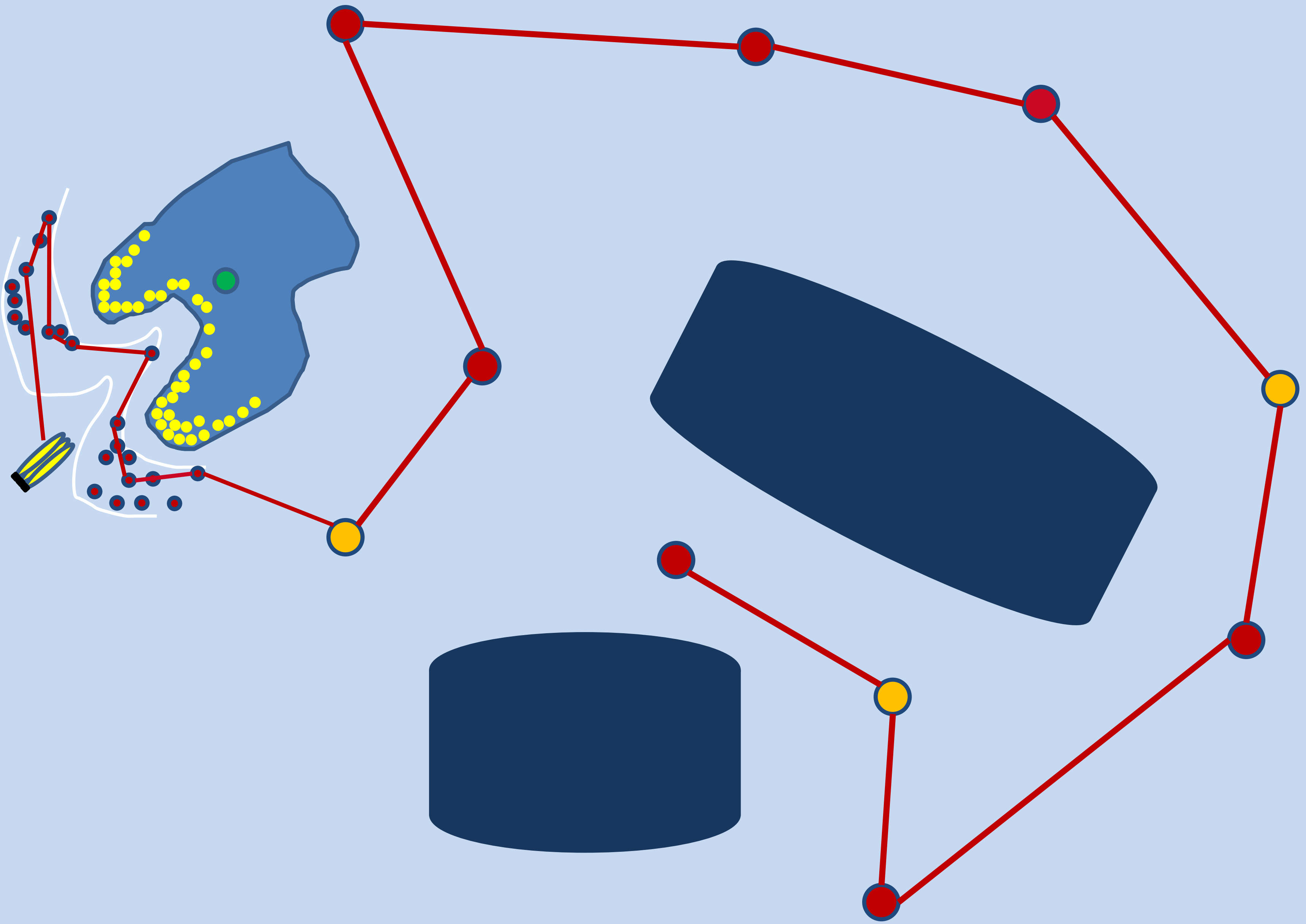
Visibility Points Discovery and REPLANNING



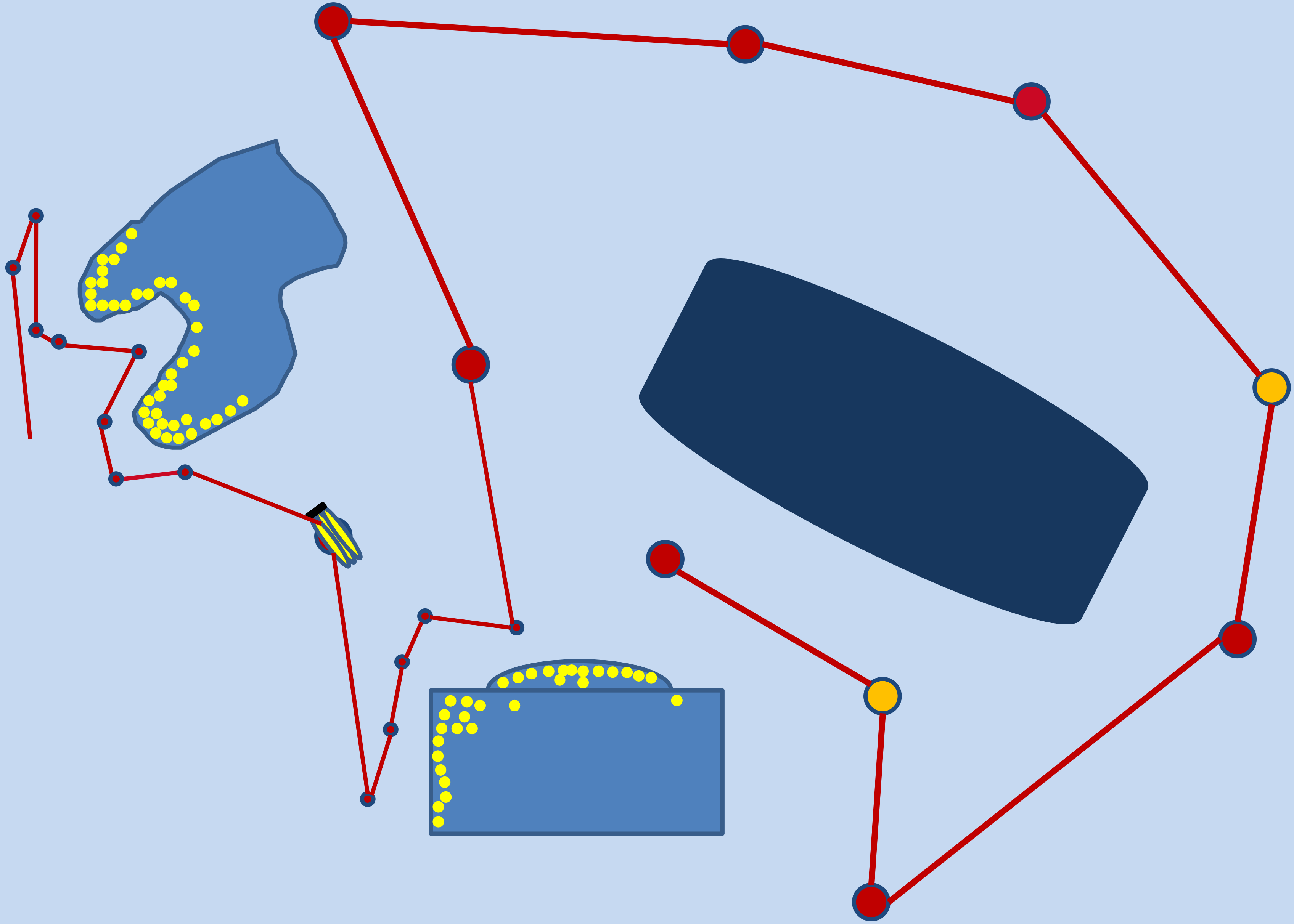
Visibility Points Discovery and REPLANNING



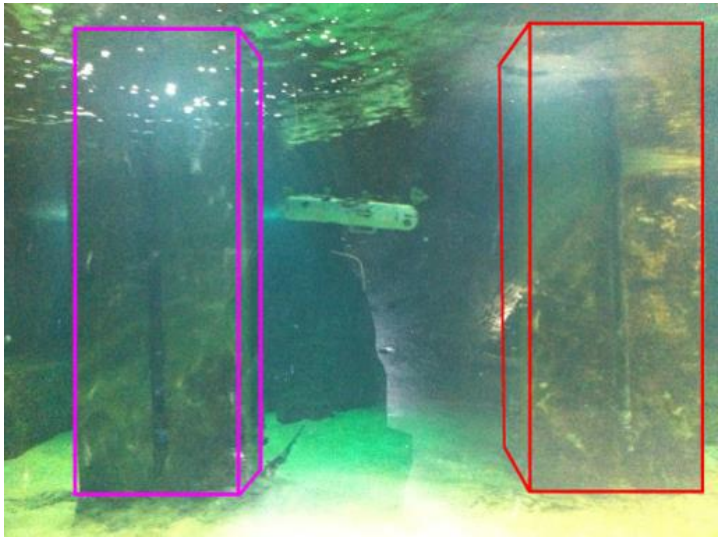
Visibility Points Discovery and REPLANNING

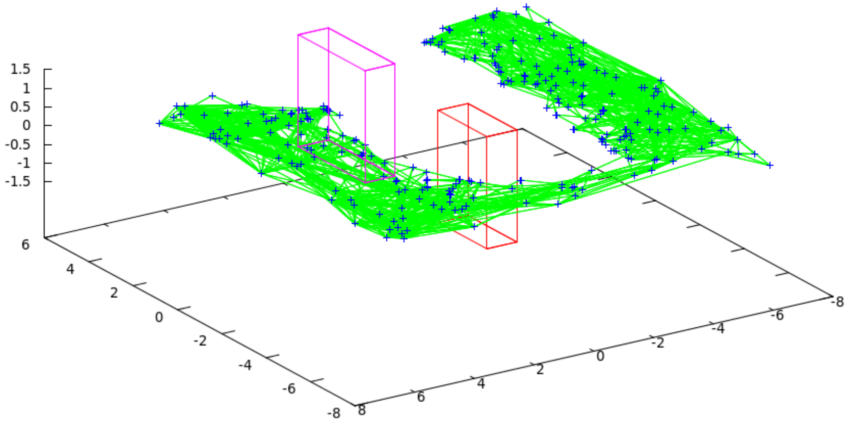


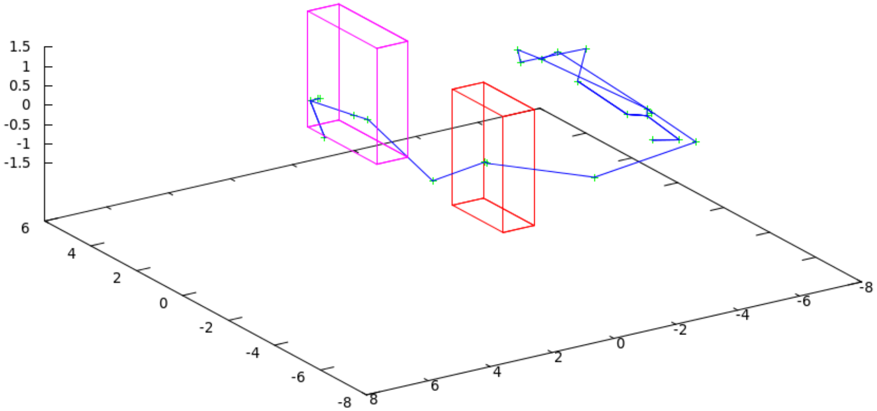
Visibility Points Discovery and REPLANNING



Physical Tests at Fort William (Scotland)







Temporal Planning Model

DOMAIN

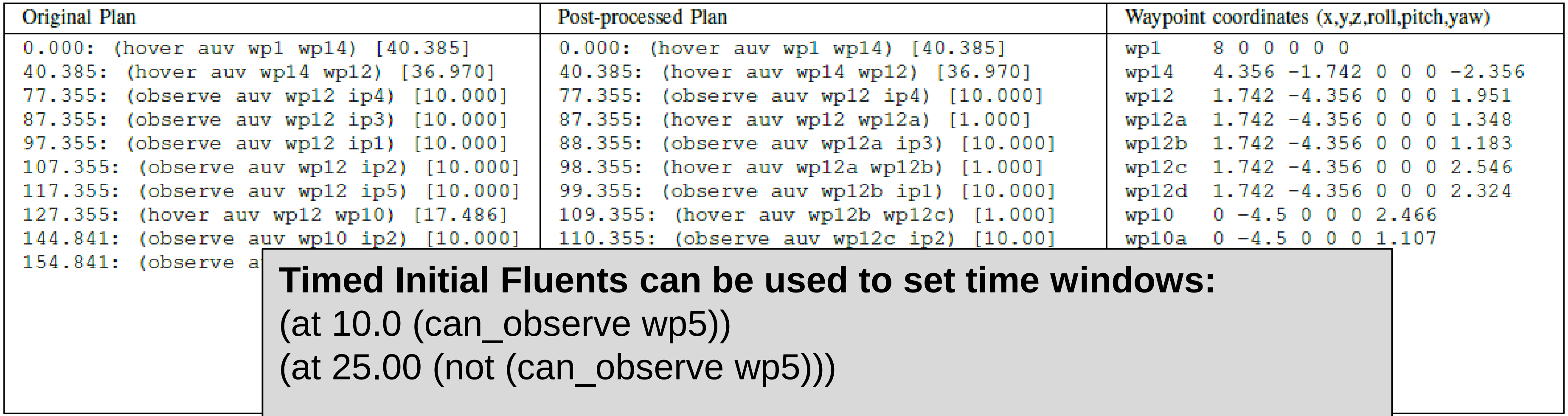
```
(:durative-action do_hover
 :parameters (?v - vehicle ?from ?to - waypoint)
 :duration ( = ?duration (* (distance ?from ?to)
                            (invtime ?v)))
 :condition (and (at start (at ?v ?from))
                 (at start (connected ?from ?to)))
 :effect (and (at start (not (at ?v ?from)))
              (at end (at ?v ?to))))

(:durative-action observe
 :parameters (?v - vehicle ?wp - waypoint
              ?ip - inspectionpoint)
 :duration ( = ?duration (obstime))
 :condition (and (at start (at ?v ?wp))
                 (at start (cansee ?v ?ip ?wp)))
 :effect (and (at start (not (cansee ?v ?ip ?wp)))
              (at end (increase (observed ?ip)
                      (obs ?ip ?wp)))))
```

PROBLEM

```
(define (problem inspection-task-p1)
  (:objects auv - vehicle
            wp1 wp2 wp3 ... - waypoint
            ip1 ip2 ip3 ... - inspectionpoint)
  (:init
    (at auv wp1)
    (= (mission-time) 0)
    (= (observed ip1) 0)
    (connected wp1 wp2)  (connected wp2 wp1)
    (= (distance wp1 wp2) 7.16958)
    (= (distance wp2 wp1) 7.16958)
    (connected wp1 wp9)  (connected wp9 wp1)
    (= (distance wp1 wp9) 3.21484)
    (= (distance wp9 wp1) 3.21484)
    ...
    (cansee auv ip4 wp12)
    (= (obs ip4 wp12) 0.445331)
    ...
  )
  (:goal (and (>= (observed ip1) 1)
         ...
  ))
  (:metric minimize (total-time)))
```

Temporal Planning Model

Plans are found using **POPF** temporal planner



Original Plan	Post-processed Plan	Waypoint coordinates (x,y,z,roll,pitch,yaw)
`0.000: (hover auv wp1 wp14) [40.385]`	`0.000: (hover auv wp1 wp14) [40.385]`	`wp1    8 0 0 0 0 0`
`40.385: (hover auv wp14 wp12) [36.970]`	`40.385: (hover auv wp14 wp12) [36.970]`	`wp14   4.356 -1.742 0 0 0 -2.356`
`77.355: (observe auv wp12 ip4) [10.000]`	`77.355: (observe auv wp12 ip4) [10.000]`	`wp12   1.742 -4.356 0 0 0 1.951`
`87.355: (observe auv wp12 ip3) [10.000]`	`87.355: (hover auv wp12 wp12a) [1.000]`	`wp12a  1.742 -4.356 0 0 0 1.348`
`97.355: (observe auv wp12 ip1) [10.000]`	`88.355: (observe auv wp12a ip3) [10.000]`	`wp12b  1.742 -4.356 0 0 0 1.183`
`107.355: (observe auv wp12 ip2) [10.000]`	`98.355: (hover auv wp12a wp12b) [1.000]`	`wp12c  1.742 -4.356 0 0 0 2.546`
`117.355: (observe auv wp12 ip5) [10.000]`	`99.355: (observe auv wp12b ip1) [10.000]`	`wp12d  1.742 -4.356 0 0 0 2.324`
`127.355: (hover auv wp12 wp10) [17.486]`	`109.355: (hover auv wp12b wp12c) [1.000]`	`wp10   0 -4.5 0 0 0 2.466`
`144.841: (observe auv wp10 ip2) [10.000]`	`110.355: (observe auv wp12c ip2) [10.00]`	`wp10a  0 -4.5 0 0 0 1.107`
`154.841: (observe a`		

Timed Initial Fluents can be used to set time windows:

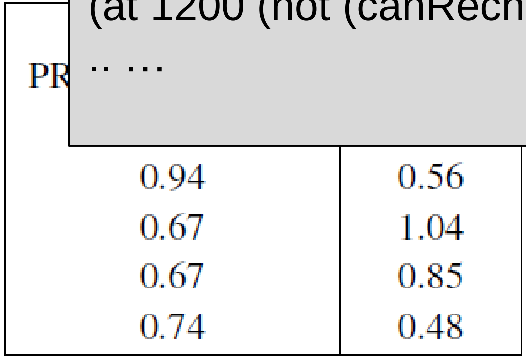
(at 10.0 (can_observe wp5))

(at 25.00 (not (can_observe wp5)))

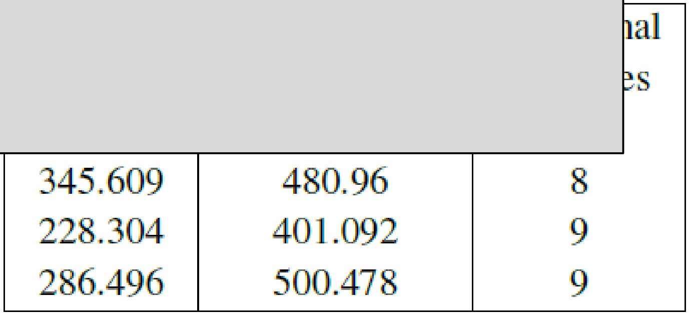
(at 658 (canRecharge dock3))

(at 1200 (not (canRecharge dock3)))

.. …



PR	
0.94	0.56
0.67	1.04
0.67	0.85
0.74	0.48



		nal es
345.609	480.96	8
228.304	401.092	9
286.496	500.478	9

ROSPlan: Planning in the Robot Operating System

ROS Basics

ROS offers a message passing interface that provides inter-process communication.

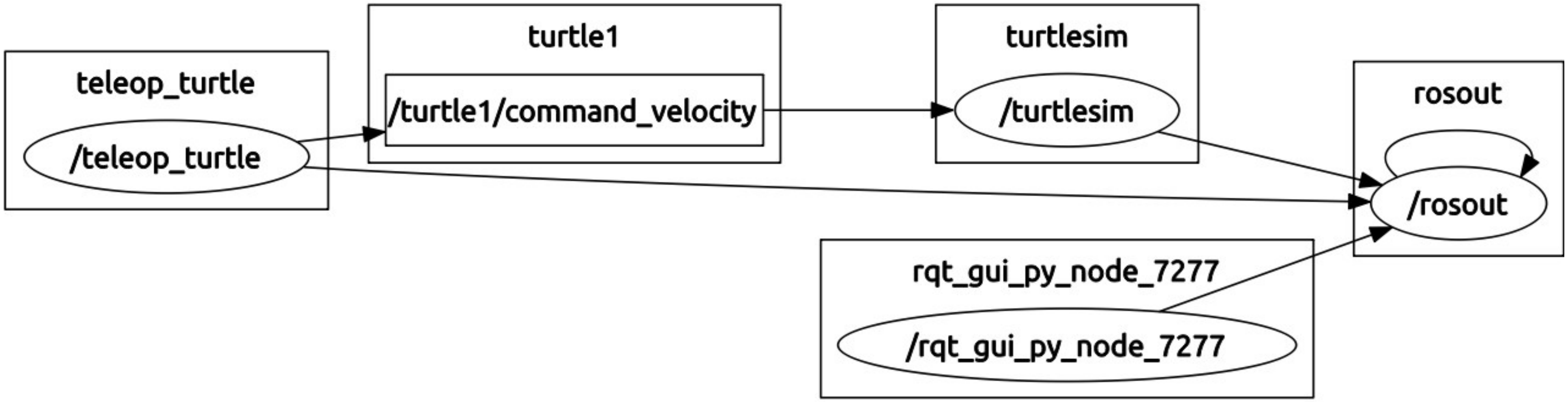
A ROS system is composed of nodes, which pass messages, usually in two forms:
1. ROS messages are published on topics and are many-to-many.
2. ROS services are used for synchronous request/response.

ROS

ROS Basics

ROS offers a message passing interface that provides inter-process communication.

A ROS system is composed of nodes, which pass messages, usually in two forms:
1. ROS messages are published on topics and are many-to-many.
2. ROS services are used for synchronous request/response.



ROS Basics

ROS offers a message passing interface that provides inter-process communication.

A ROS system is composed of nodes, which pass messages, usually in two forms:
1. ROS messages are published on topics and are many-to-many.
2. ROS services are used for synchronous request/response.

```
<launch>
  <include file="$(find turtlebot_navigation)/launch/includes/velocity_smoother.launch.xml"/>
  <include file="$(find turtlebot_navigation)/launch/includes/safety_controller.launch.xml"/>

  <arg name="global_frame_id" default="map"/>
  <arg name="odom_topic" default="odom" />
  <arg name="laser_topic" default="scan" />


  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find turtlebot_navigation)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find turtlebot_navigation)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find turtlebot_navigation)/param/local_costmap_params.yaml" command="load" />
    <remap from="cmd_vel" to="navigation_velocity_smoother/raw_cmd_vel"/>
    <remap from="odom" to="$(arg odom_topic)"/>
    <remap from="scan" to="$(arg laser_topic)"/>
  </node>
</launch>
```

ROS Basics

ROS offers a message passing interface that provides inter-process communication.

A ROS system is composed of nodes, which pass messages, usually in two forms:
1. ROS messages are published on topics and are many-to-many.
2. ROS services are used for synchronous request/response.

The actionlib package standardizes the interface for pre-emptable tasks.
For example:
- navigation,
- performing a laser scan
- detecting the handle of a door...

Aside from numerous tools, Actionlib provides standard messages for sending task:
- goals
- feedback
- result

ROS Basics

Aside from numerous tools, Actionlib provides standard messages for sending task:

- goals
- feedback
- result

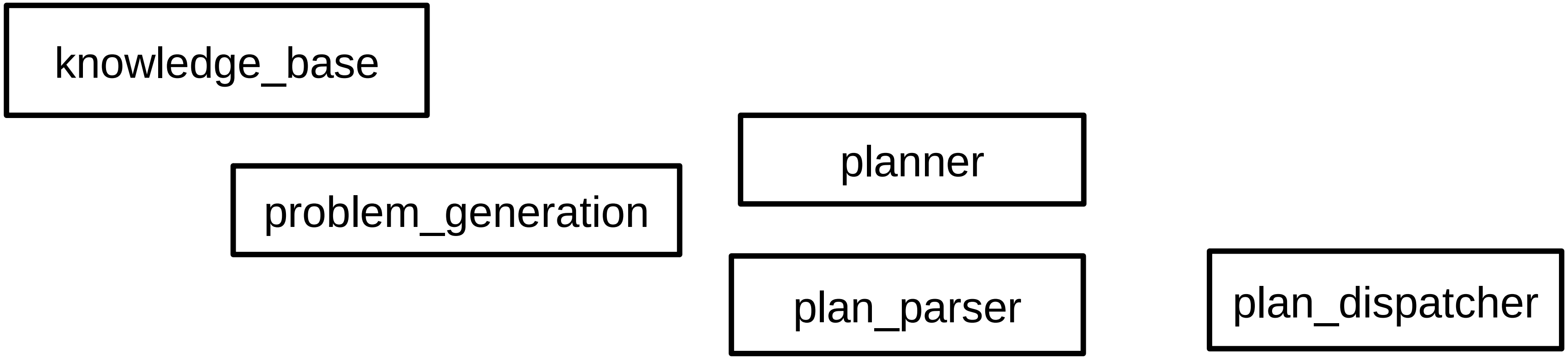
move_base/MoveBaseGoal

```
geometry_msgs/PoseStamped target_pose
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
```

ROSPlan Basics

The ROSPlan package provides a standard interface for PDDL planners in ROS.

The purpose of the ROSPlan package is to integrate planners within a ROS system without having to write an architecture from scratch.



knowledge_base

problem_generation

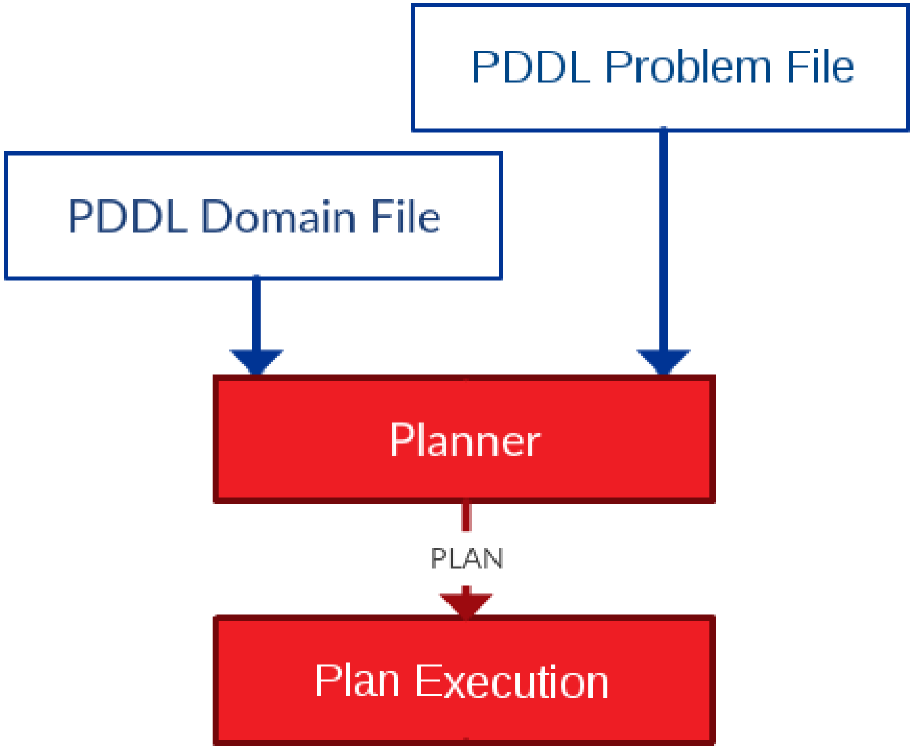
planner

plan_parser

plan_dispatcher

Plan Execution 1: Very simple Dispatch

The most basic structure.
- The plan is generated.
- The plan is executed.

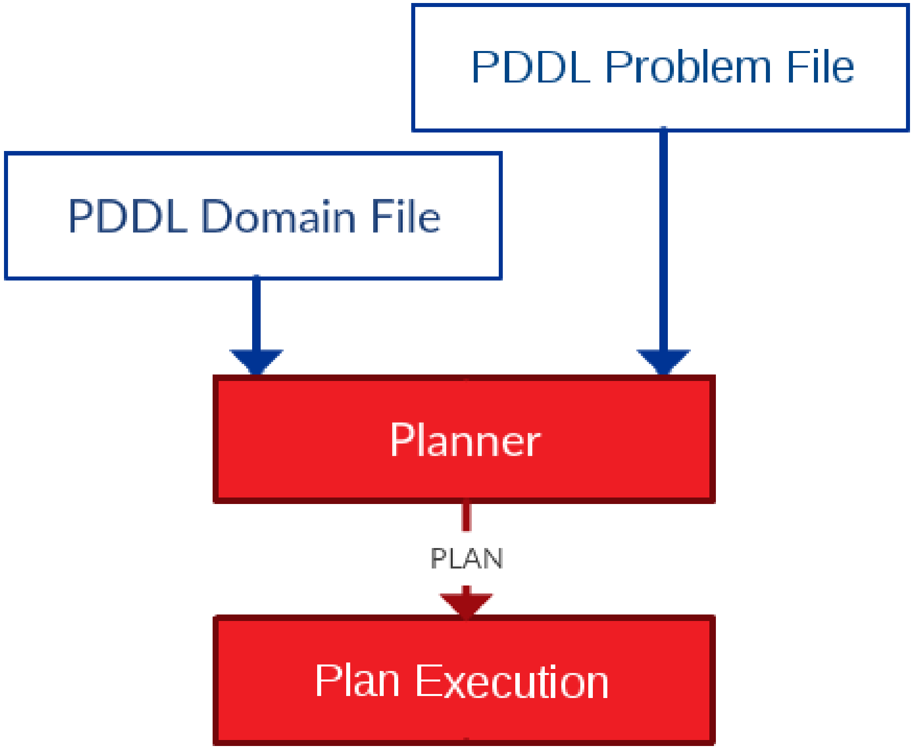


Plan Execution 1: Very simple Dispatch

The most basic structure.
- The plan is generated.
- The plan is executed.

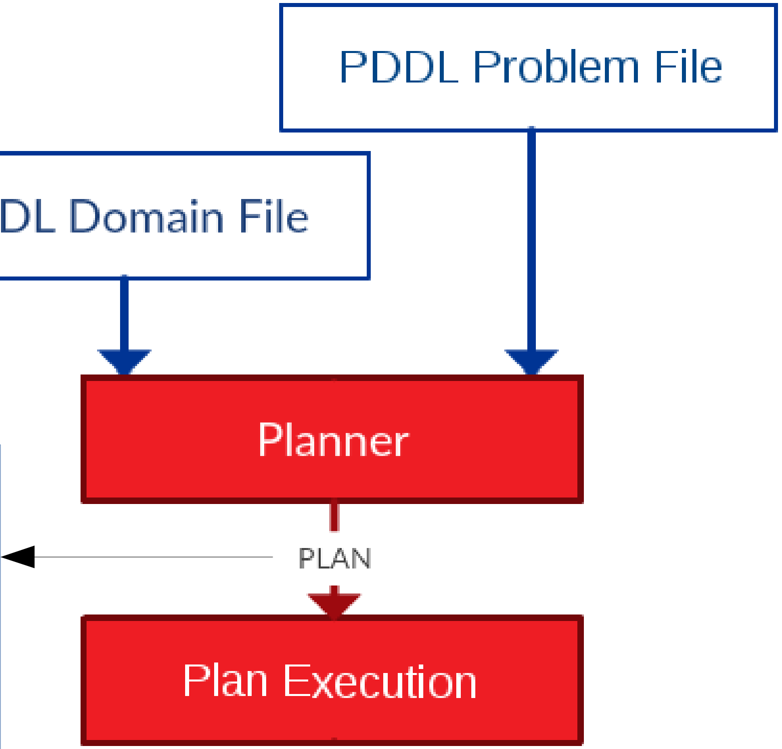
The red boxes are included in ROSPlan. They correspond to ROS nodes.

The domain and problem file can be supplied
- in launch parameters
- as ROS service parameters



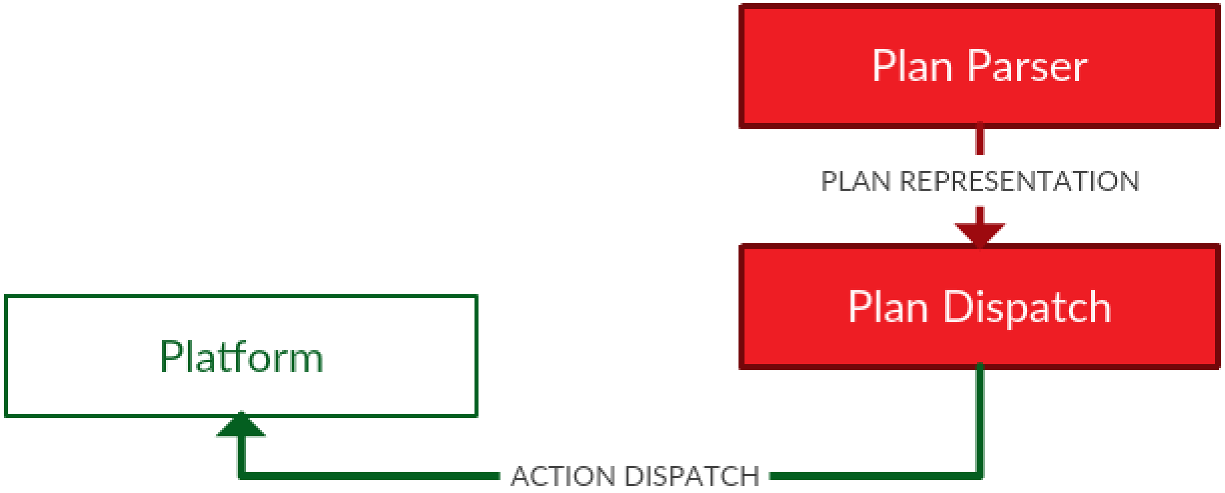
Plan Execution 1: Very simple Dispatch

```
rosplan_dispatch_msgs/CompletePlan
ActionDispatch[] plan
  int32 action_id
  string name
  diagnostic_msgs/KeyValue[] parameters
    string key
    string value
  float32 duration
  float32 dispatch_time
```



Dispatch Loop without feedback

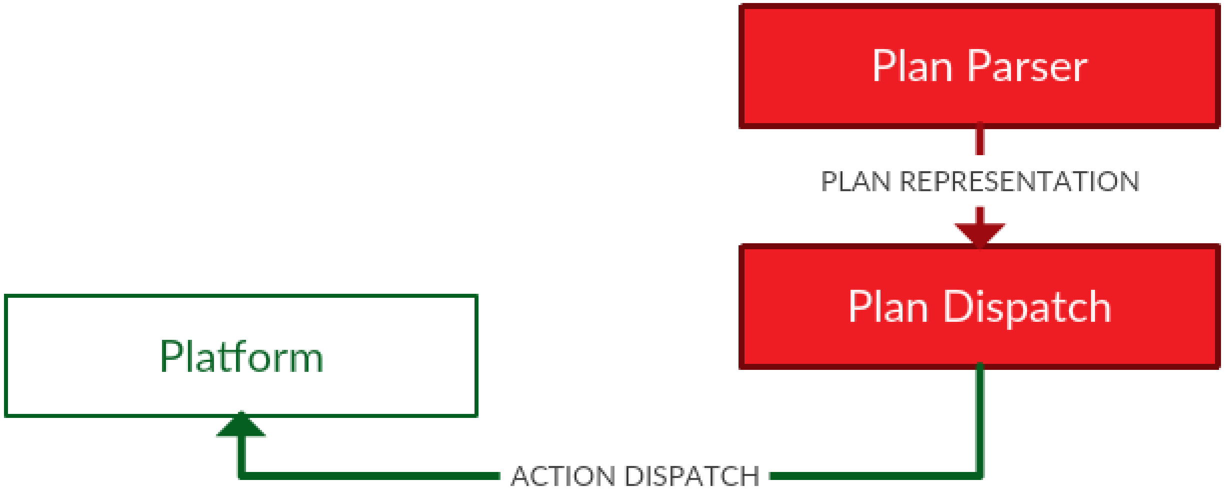
How does the “Plan Execution” ROS node work? There are multiple variants:
- simple sequential execution
- timed execution
- Petri-Net plans
- Conditional Contingent Temporal Constraint Network.
- etc.



Dispatch Loop without feedback

How does the “Plan Execution” ROS node work? There are multiple variants:
- simple sequential execution

1. Take the next action from the plan.
2. Send the action to control.
3. Wait for the action to complete.
4. GOTO 1.

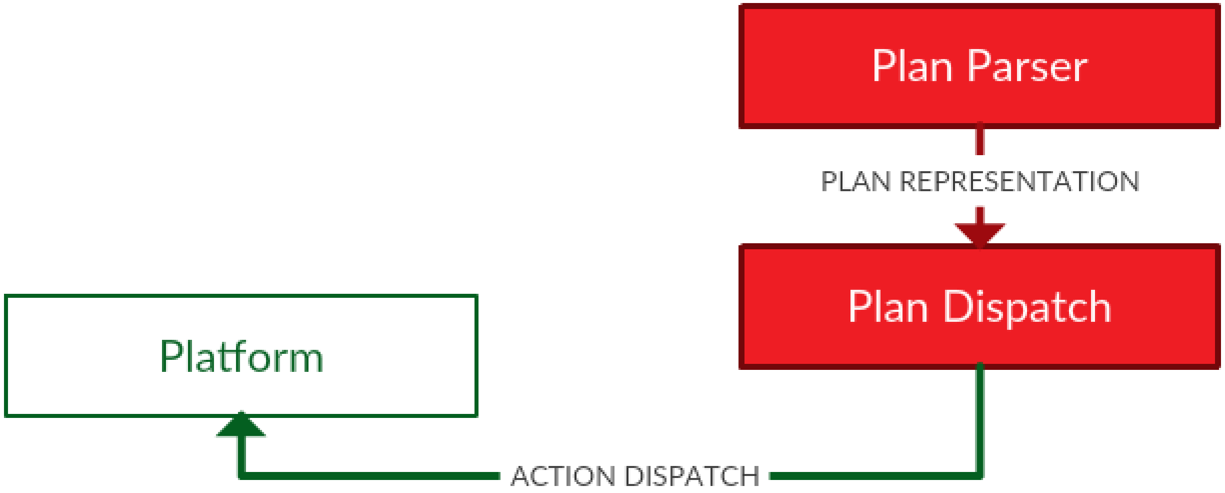


Dispatch Loop without feedback

How does the “Plan Execution” ROS node work? There are multiple variants:
- simple sequential execution

1. Take the next action from the plan.
2. Send the action to control.
3. Wait for the action to complete.
4. GOTO 1.

An action in the plan is stored as a ROS message *ActionDispatch,* which corresponds to a PDDL action.

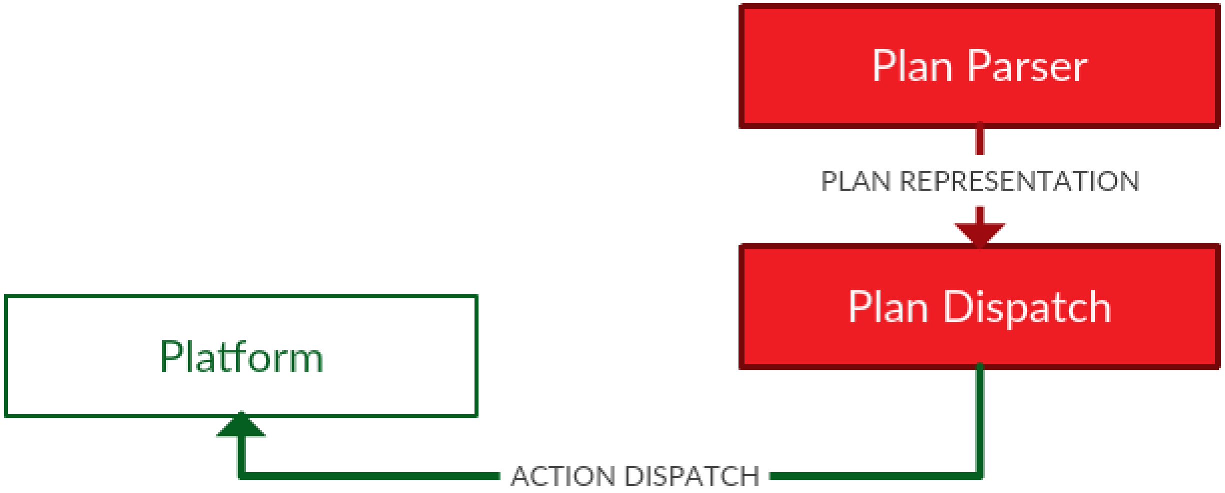


Dispatch Loop without feedback

How does the “Plan Execution” ROS node work? There are multiple variants:
- simple sequential execution

1. Take the next action from the plan.
2. Send the action to control.
3. Wait for the action to complete.
4. GOTO 1.

The *ActionDispatch* message is received by a listening interface node, and becomes a goal for control.

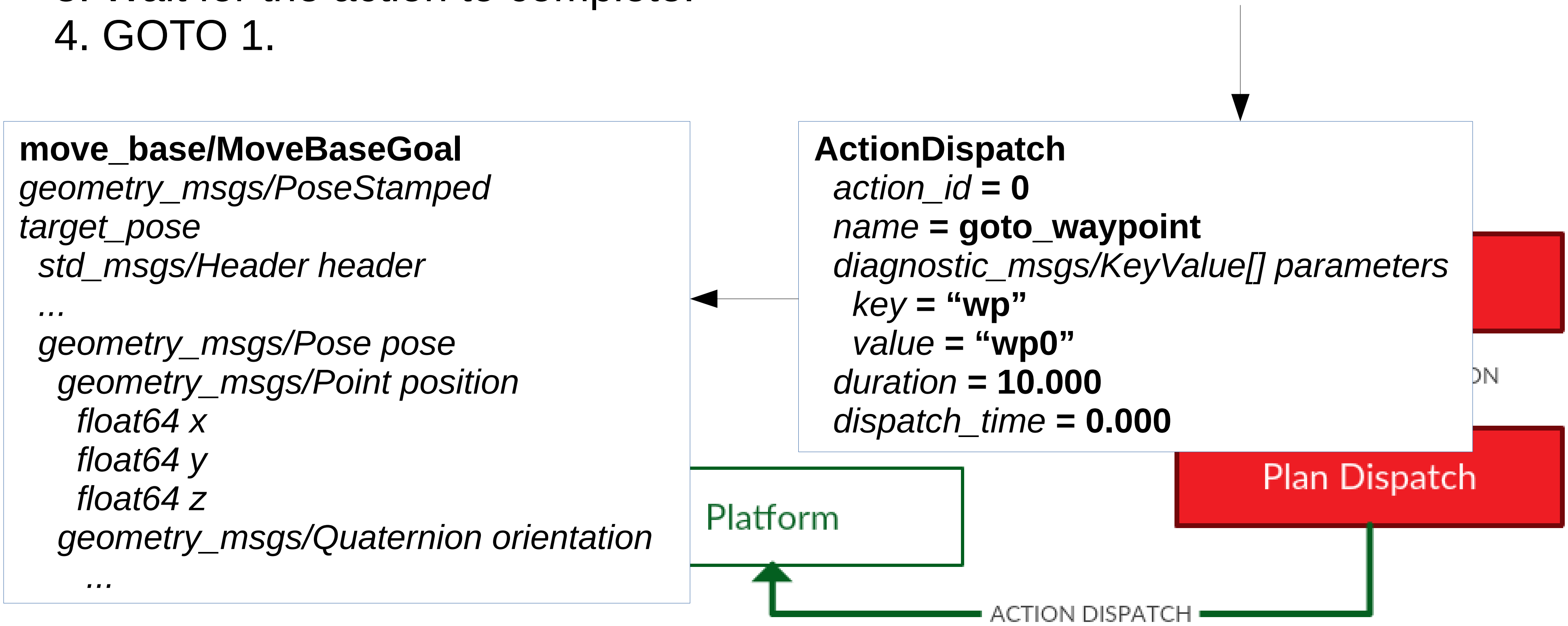


Dispatch Loop without feedback

How does the "Plan Execution" ROS node work? There are multiple variants:
- simple sequential execution

1. Take the next action from the plan.
2. Send the action to control.
3. Wait for the action to complete.
4. GOTO 1.

```
0.000:  (goto_waypoint wp0)      [10.000]
10.01: (observe ip3)             [5.000]
15.02: (grasp_object box4)       [60.000]
```



```
move_base/MoveBaseGoal
geometry_msgs/PoseStamped
target_pose
  std_msgs/Header header
  ...
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      ...
```

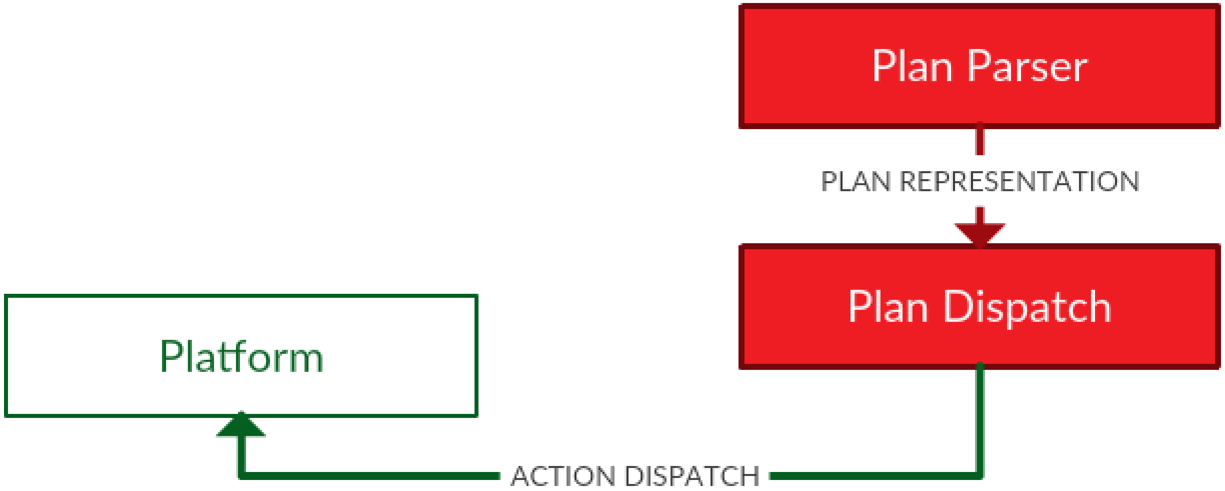
```
ActionDispatch
  action_id = 0
  name = goto_waypoint
  diagnostic_msgs/KeyValue[] parameters
    key = "wp"
    value = "wp0"
  duration = 10.000
  dispatch_time = 0.000
```

Dispatch Loop without feedback

How does the “Plan Execution” ROS node work? There are multiple variants:
- simple sequential execution

1. Take the next action from the plan.
2. Send the action to control.
3. Wait for the action to complete.
4. GOTO 1.

Feedback is returned to the simple dispatcher (action success or failure) through a ROS message *ActionFeedback.*

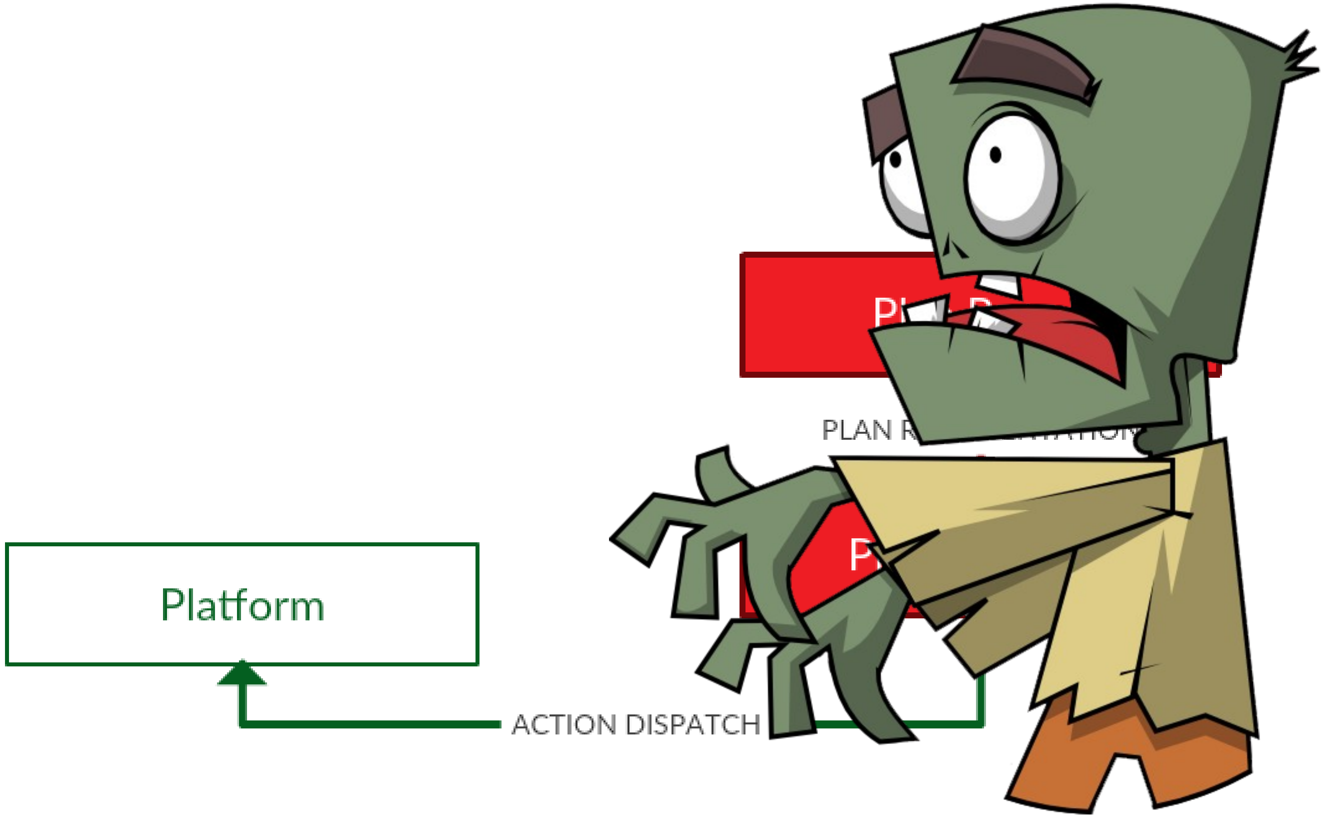


Plan Execution Failure

This form of simple dispatch has some problems. The robot often exhibits zombie-like behaviour in one of two ways:

1. An action fails, and the recovery is handled by control.

2. The plan fails, but the robot doesn't notice.



Bad behaviour 1: Action Failure

An action might never terminate. For example:
- a navigation action that cannot find a path to its goal.
- a grasp action that allows retries.

At some point the robot must give up.

Bad behaviour 1: Action Failure

An action might never terminate. For example:
- a navigation action that cannot find a path to its goal.
- a grasp action that allows retries.

At some point the robot must give up.

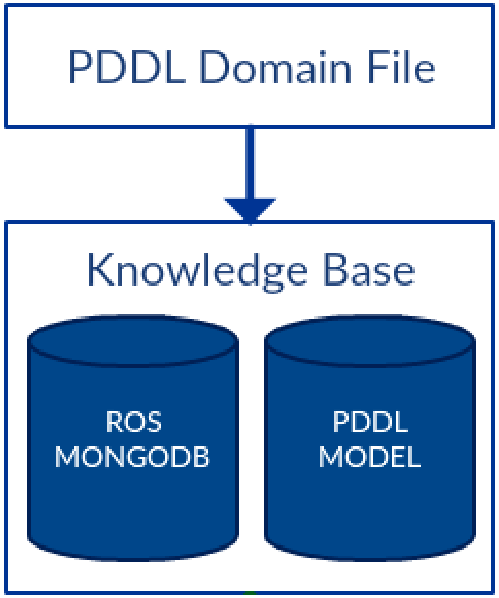
If we desire persistent autonomy, then the robot must be able to plan again, from the new current state, without human intervention.

The problem file must be regenerated.

PDDL Model

To generate the problem file automatically, the agent must store a model of the world.

In ROSPlan, a PDDL model is stored in a ROS node called the Knowledge Base.



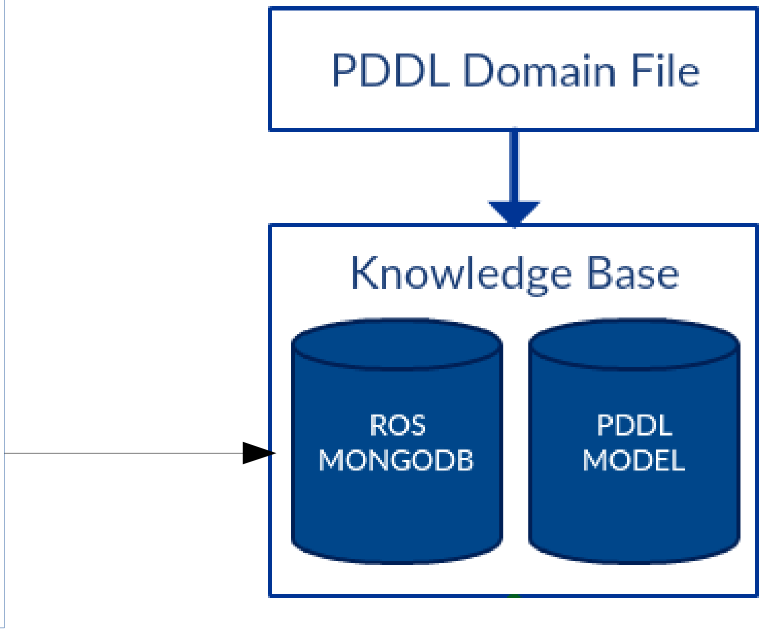
PDDL Model

To generate the problem file automatically, the agent must store a model of the world.

In ROSPlan, a PDDL model is stored in a ROS node called the Knowledge Base.

rosplan_knowledge_msgs/KnowledgeItem

```
uint8 INSTANCE=0
uint8 FACT=1
uint8 FUNCTION=2
uint8 knowledge_type
string instance_type
string instance_name
string attribute_name
diagnostic_msgs/KeyValue[] values
  string key
  string value
float64 function_value
bool is_negative
```



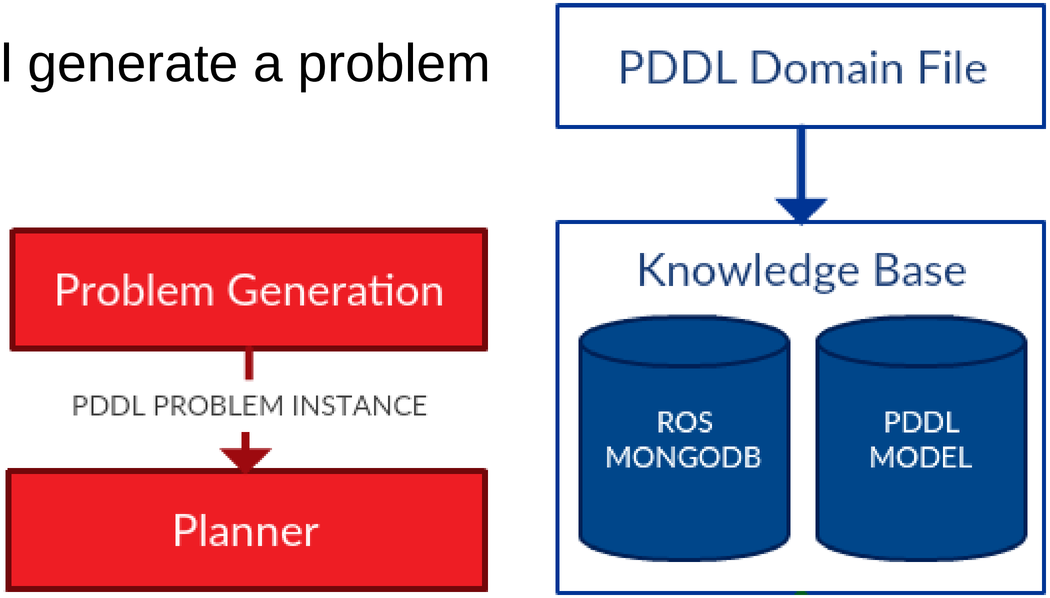
PDDL Model

To generate the problem file automatically, the agent must store a model of the world.

In ROSPlan, a PDDL model is stored in a ROS node called the Knowledge Base.

From this the initial state of a new planning problem can be created.

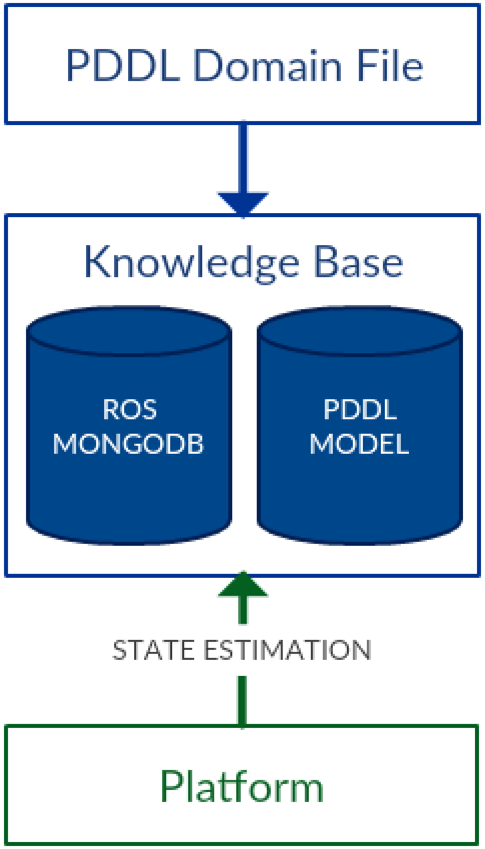
ROSPlan contains a node which will generate a problem file for the ROSPlan planning node.



PDDL Model

The model must be continuously updated from sensor data.

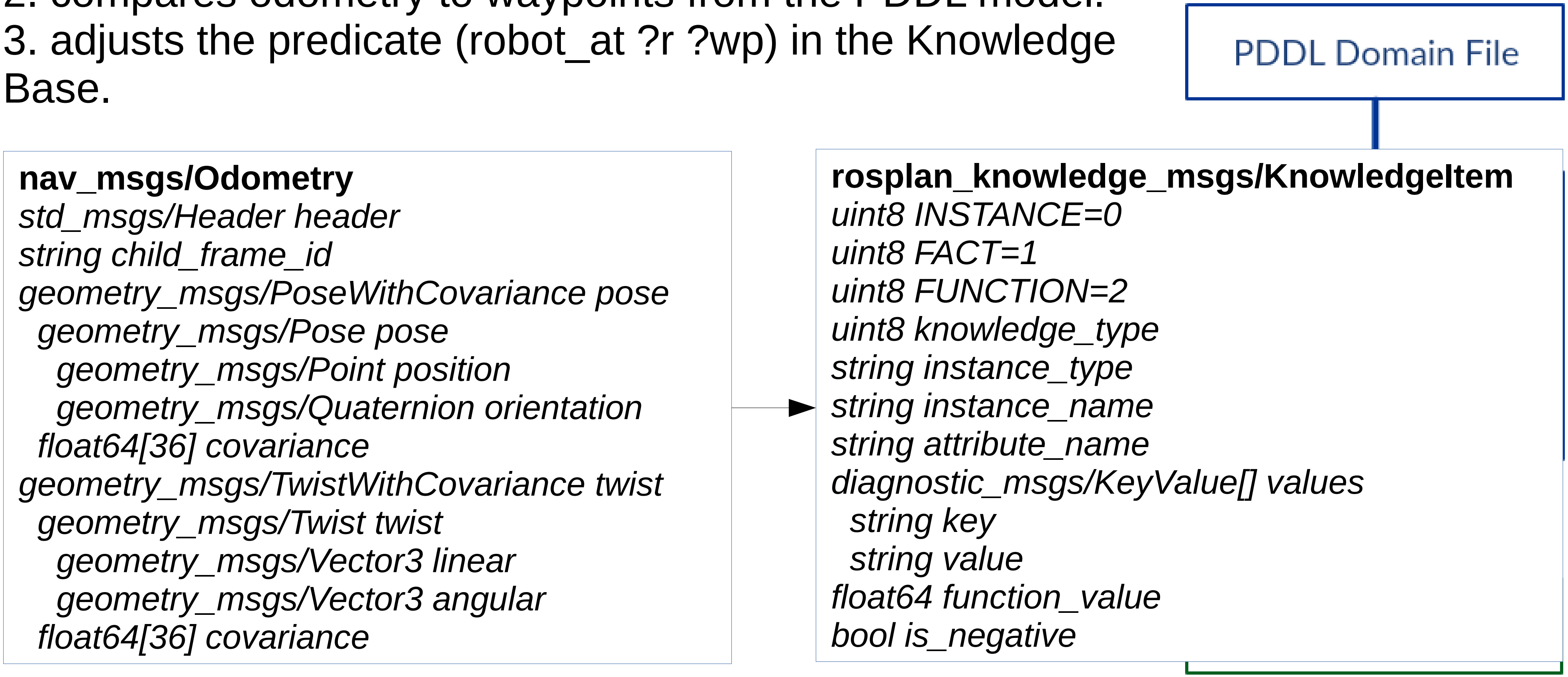
For example a new ROS node:
1. subscribes to odometry data.
2. compares odometry to waypoints from the PDDL model.
3. adjusts the predicate (robot_at ?r ?wp) in the Knowledge Base.



PDDL Model

The model must be continuously updated from sensor data.

For example a new ROS node:
1. subscribes to odometry data.
2. compares odometry to waypoints from the PDDL model.
3. adjusts the predicate (robot_at ?r ?wp) in the Knowledge Base.



nav_msgs/Odometry

```
std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
    geometry_msgs/Quaternion orientation
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
    geometry_msgs/Vector3 angular
  float64[36] covariance
```

rosplan_knowledge_msgs/KnowledgeItem

```
uint8 INSTANCE=0
uint8 FACT=1
uint8 FUNCTION=2
uint8 knowledge_type
string instance_type
string instance_name
string attribute_name
diagnostic_msgs/KeyValue[] values
  string key
  string value
float64 function_value
bool is_negative
```

Bad Behaviour 2: Plan Failure

What happens when the actions succeed, but the plan fails?

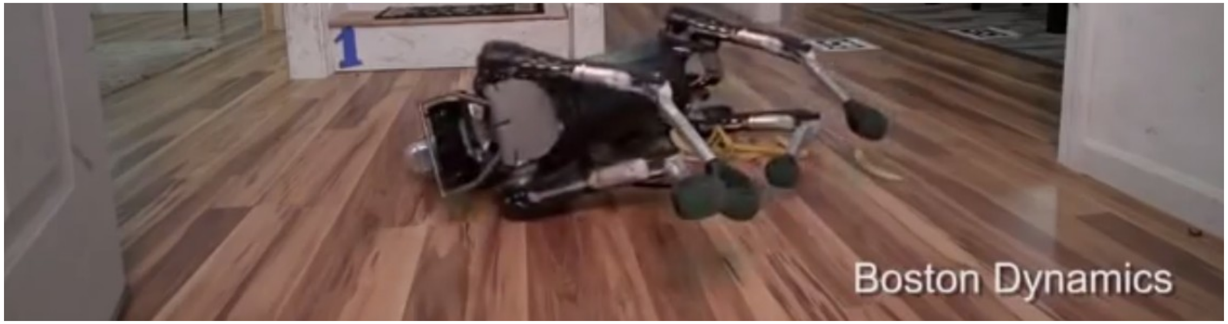
This can't always be detected by lower level control.

Bad Behaviour 2: Plan Failure

What happens when the actions succeed, but the plan fails?

This can't always be detected by lower level control.

PLAN COMPLETE



Bad Behaviour 2: Plan Failure

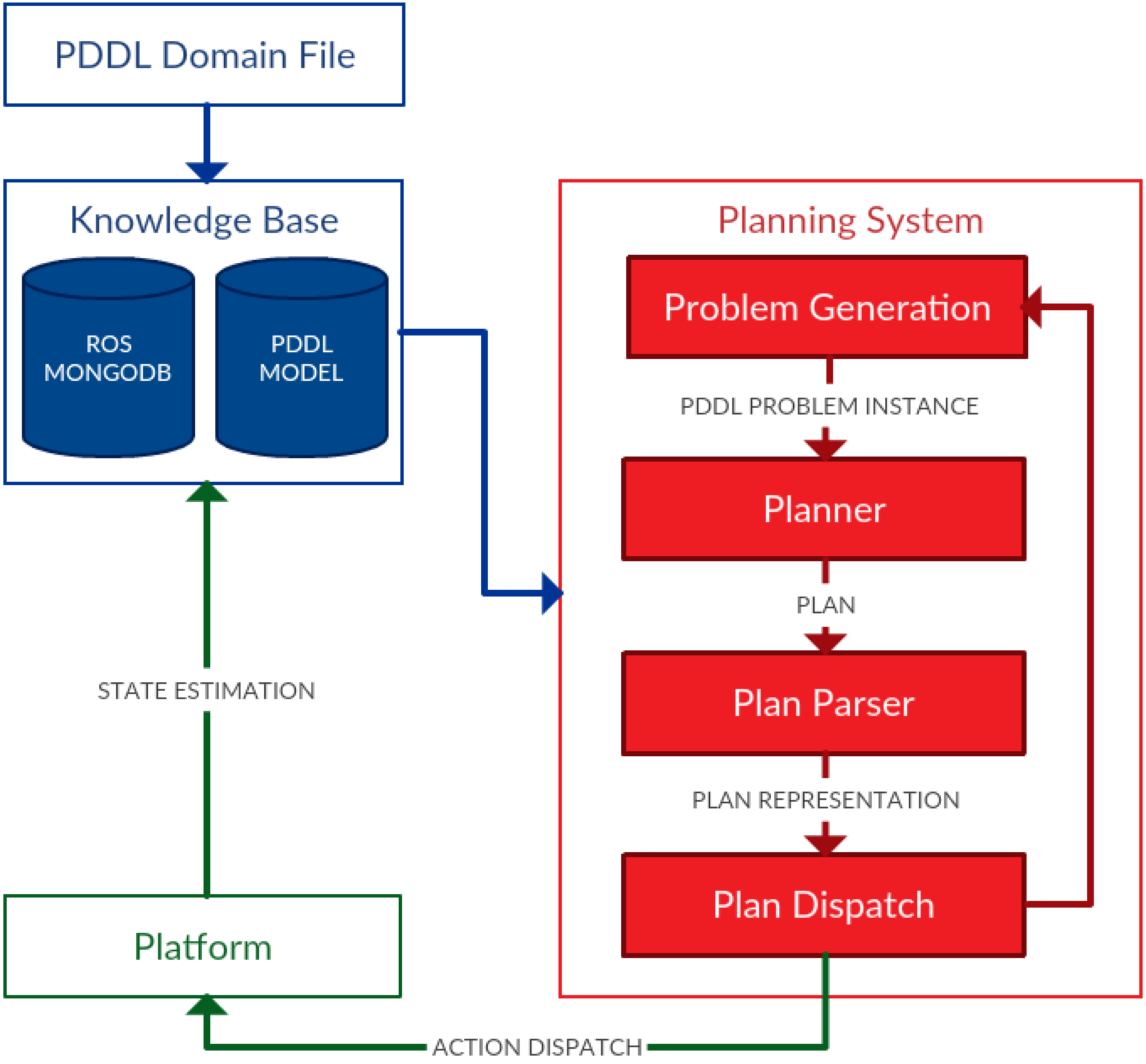
There should be diagnosis at the level of the plan.

If the plan will fail in the future, the robot should not continue to execute the plan for a long time without purpose.

Bad Behaviour 2: Plan Failure

There should be diagnosis at the level of the plan.

If the plan will fail in the future, the robot should not continue to execute the plan for a long time without purpose.



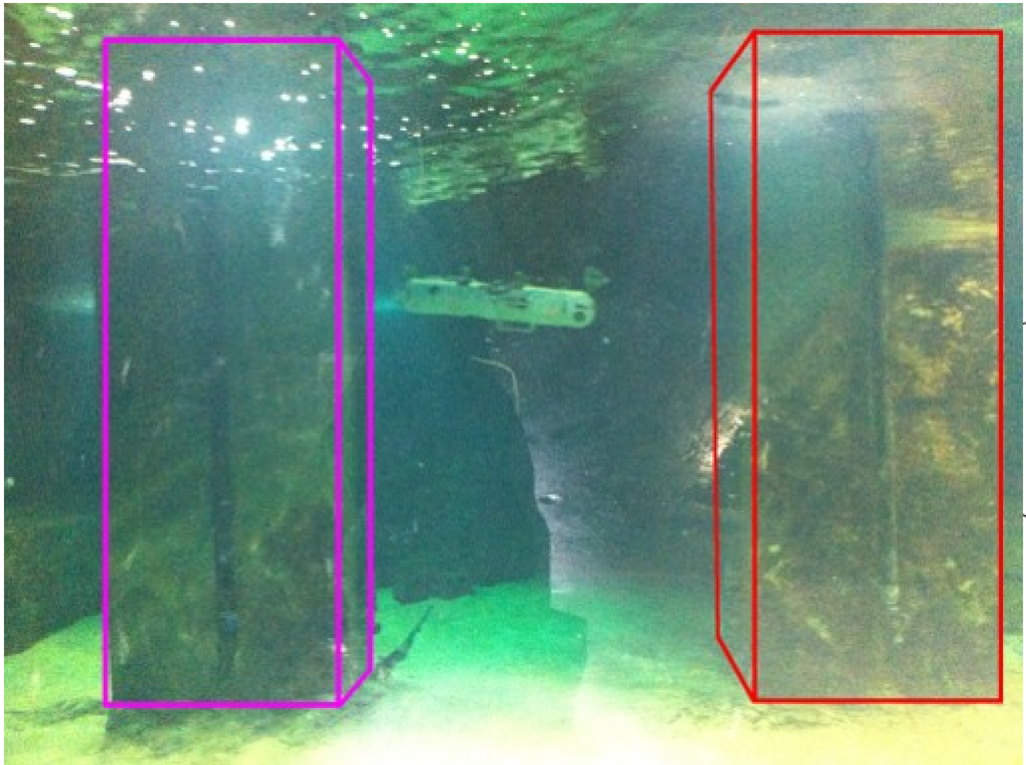
Bad Behaviour 2: Plan Failure

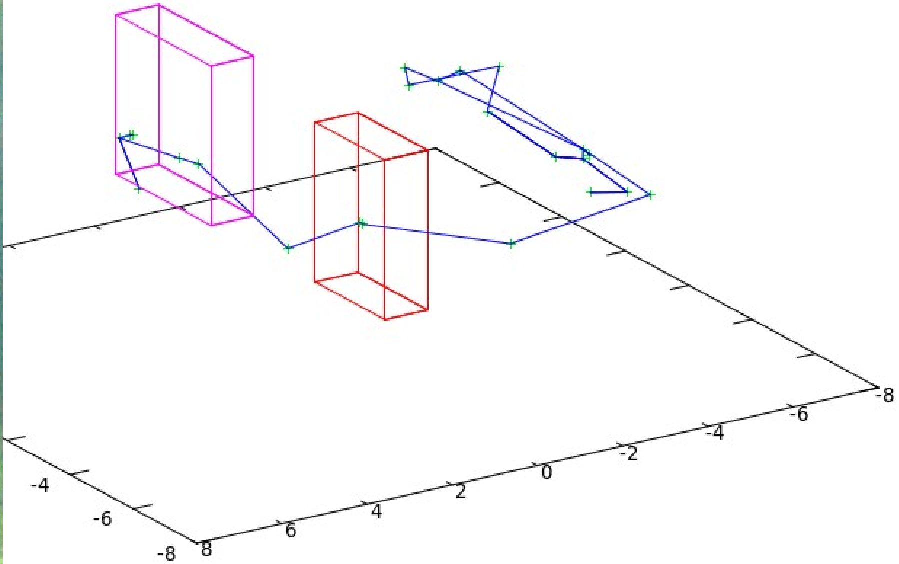
The AUV plans for inspection missions, recording images of pipes and welds.

It navigates through a probabilistic roadmap. The environment is uncertain, and the roadmap might not be correct.

Bad Behaviour 2: Plan Failure

The plan is continuously validated against the model.



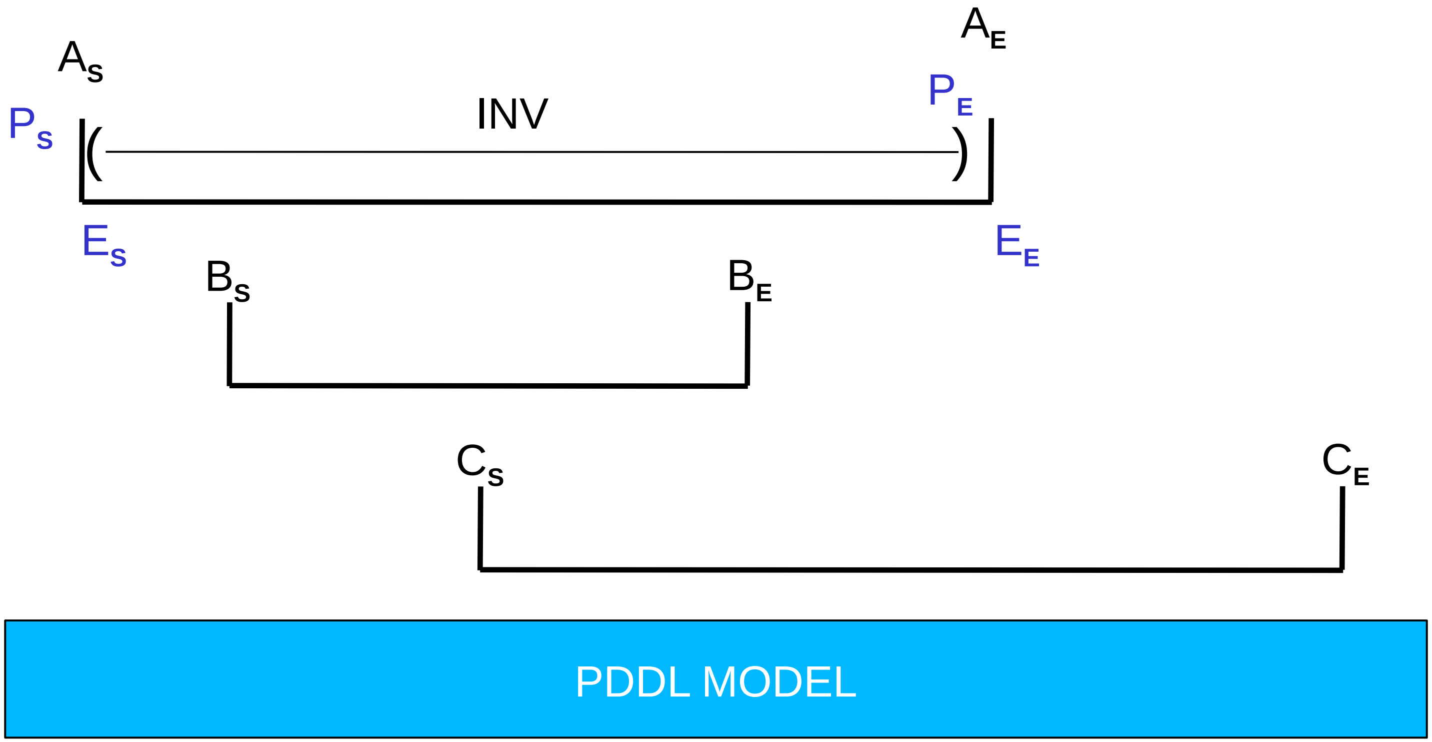


The planned inspection path is shown on the right. The AUV will move around to the other side of the pillars before inspecting the pipes on their facing sides.

After spotting an obstruction between the pillars, the AUV should re-plan early.

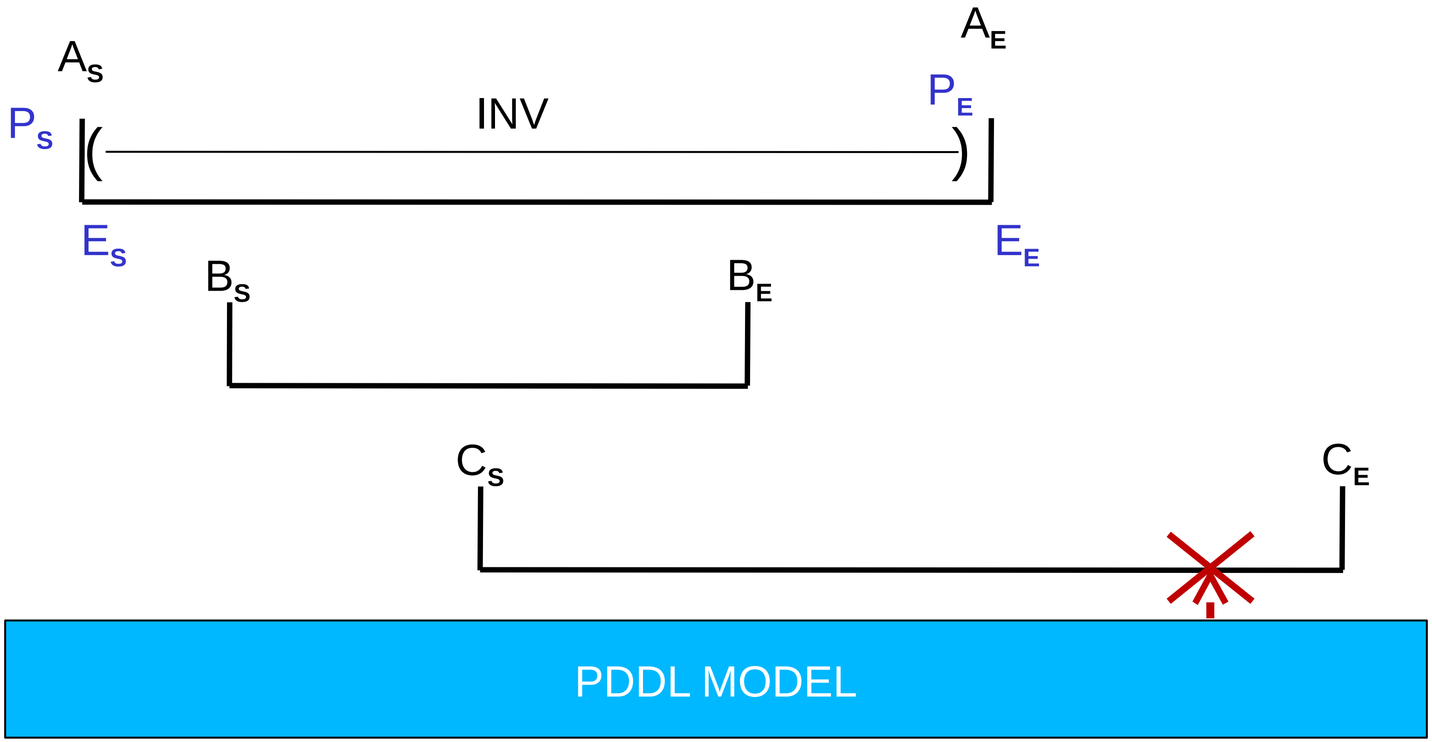
Bad Behaviour 2: Plan Failure

The plan is continuously validated against the model.



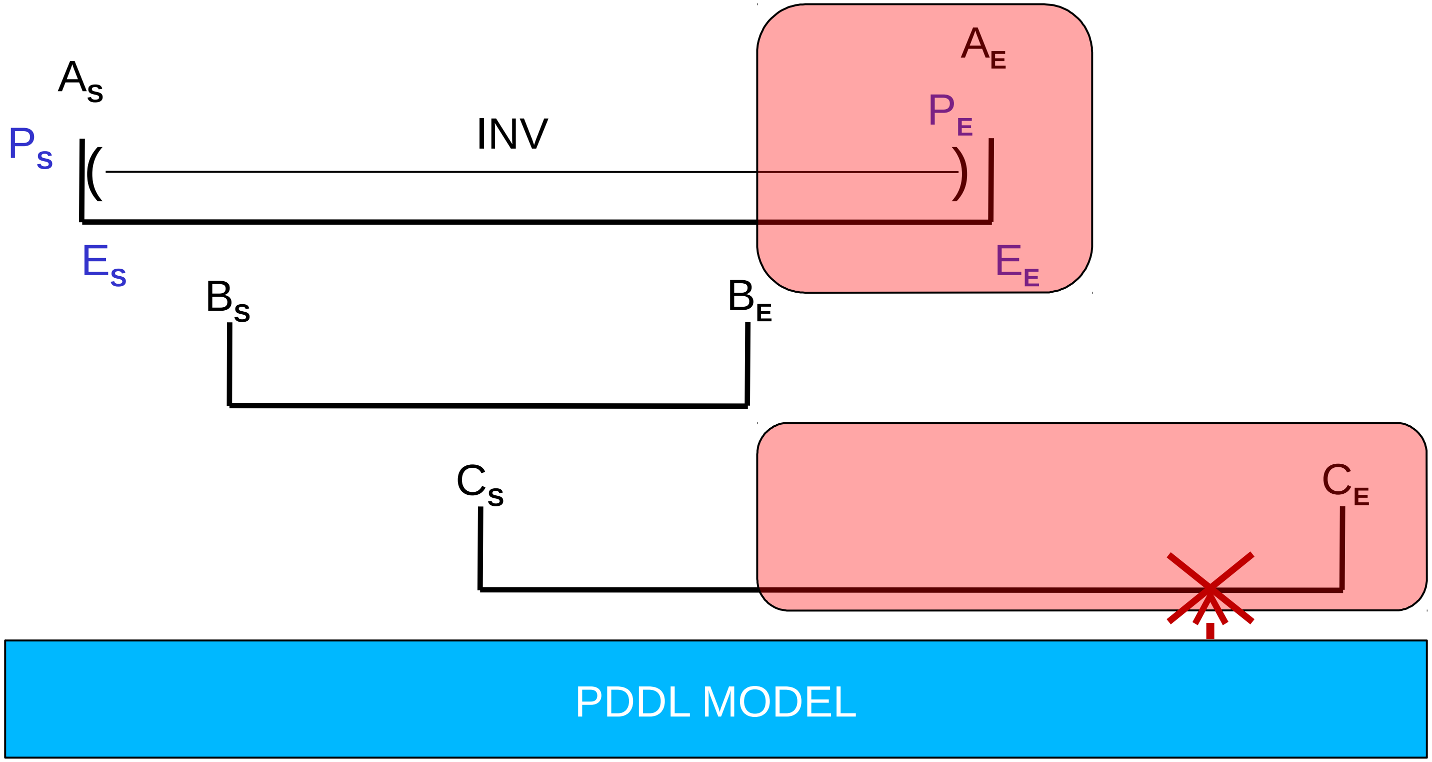
Bad Behaviour 2: Plan Failure

The plan is continuously validated against the model.



Bad Behaviour 2: Plan Failure

The plan is continuously validated against the model.

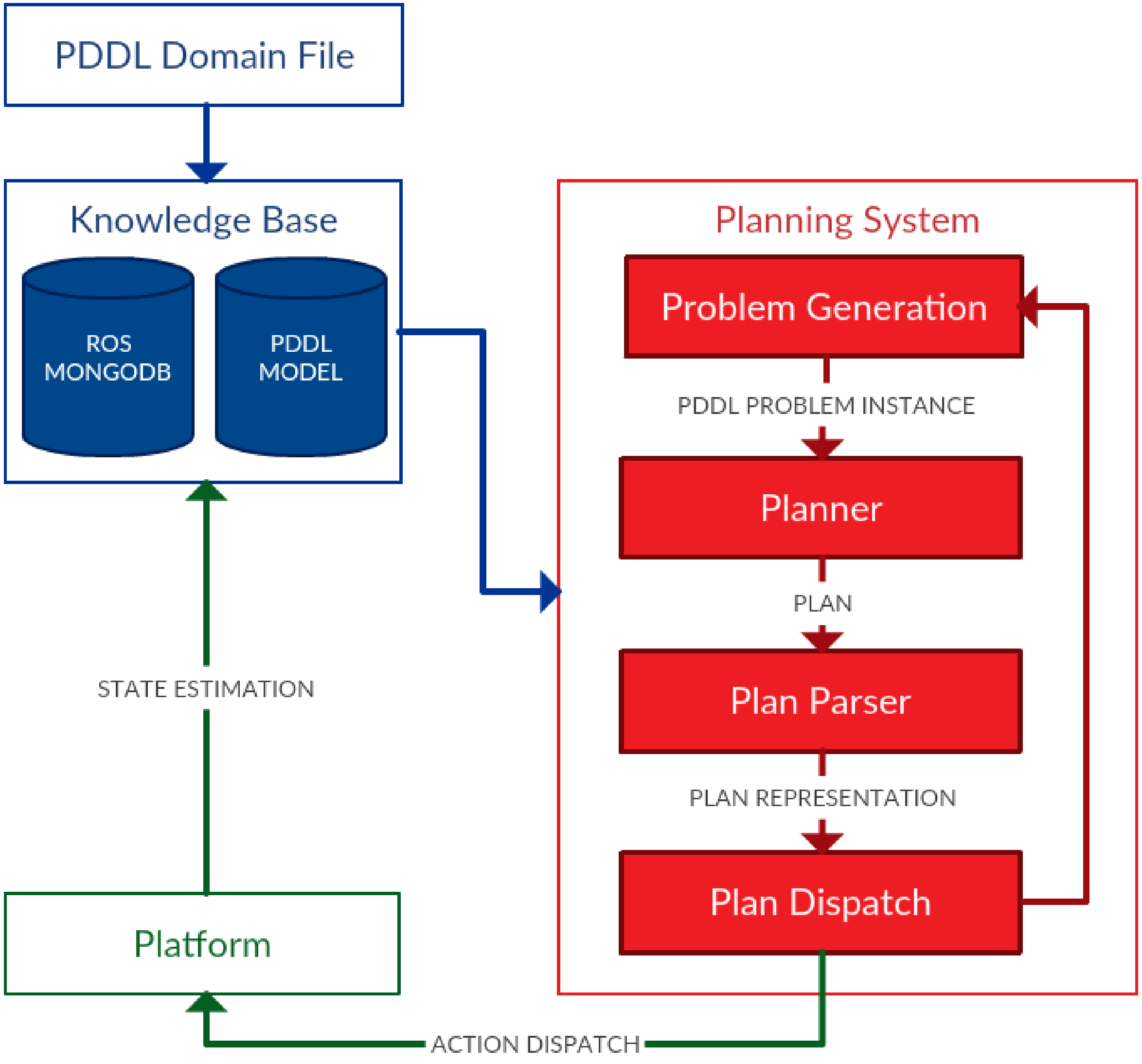


Bad Behaviour 2: Plan Failure

ROSPlan validates using VAL. [Fox et al. 2005]

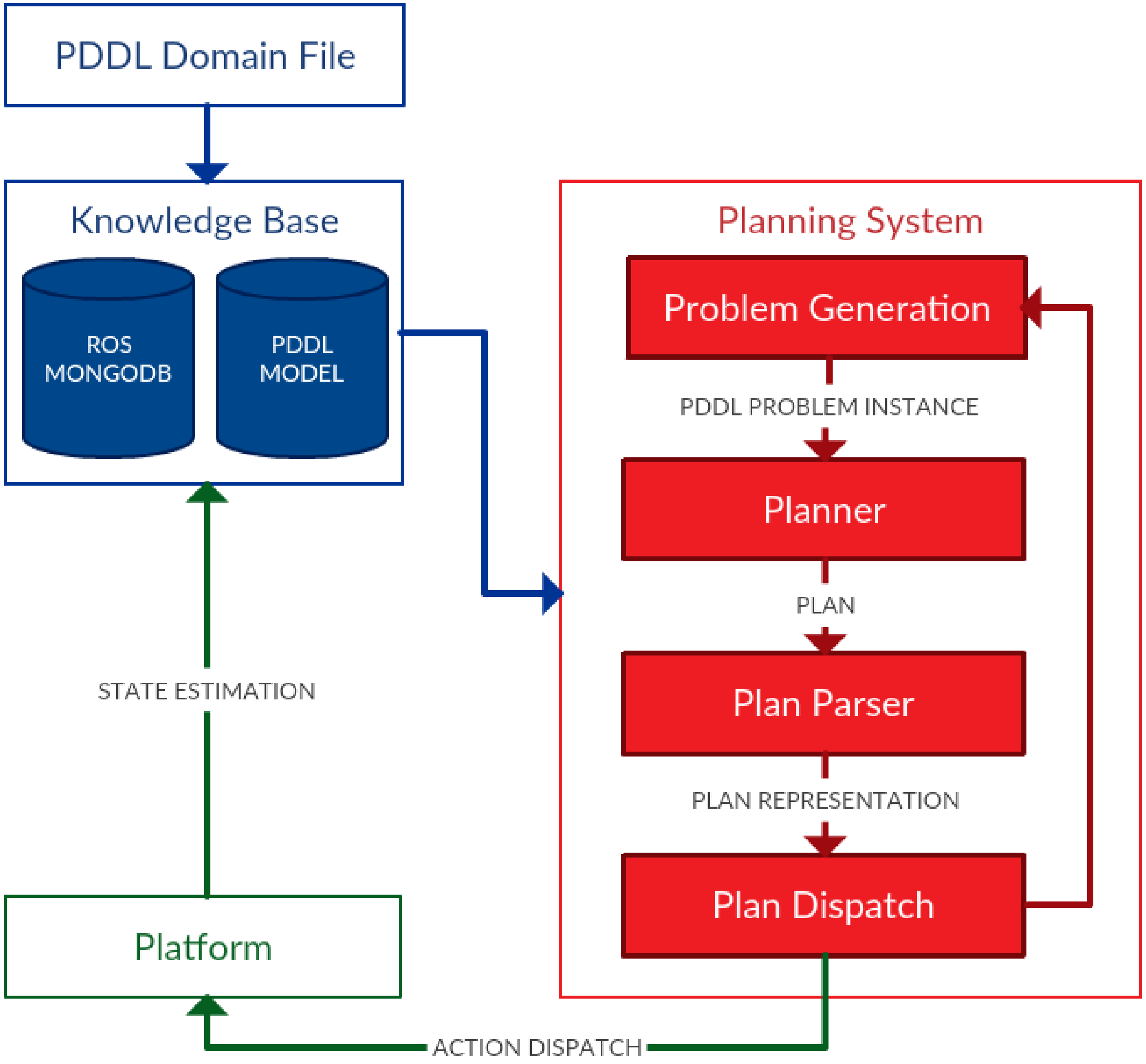
ROSPlan: Default Configuration

Now the system is more complex:
- PDDL model is continuously updated from sensor data.
- problem file is automatically generated.



ROSPlan: Default Configuration

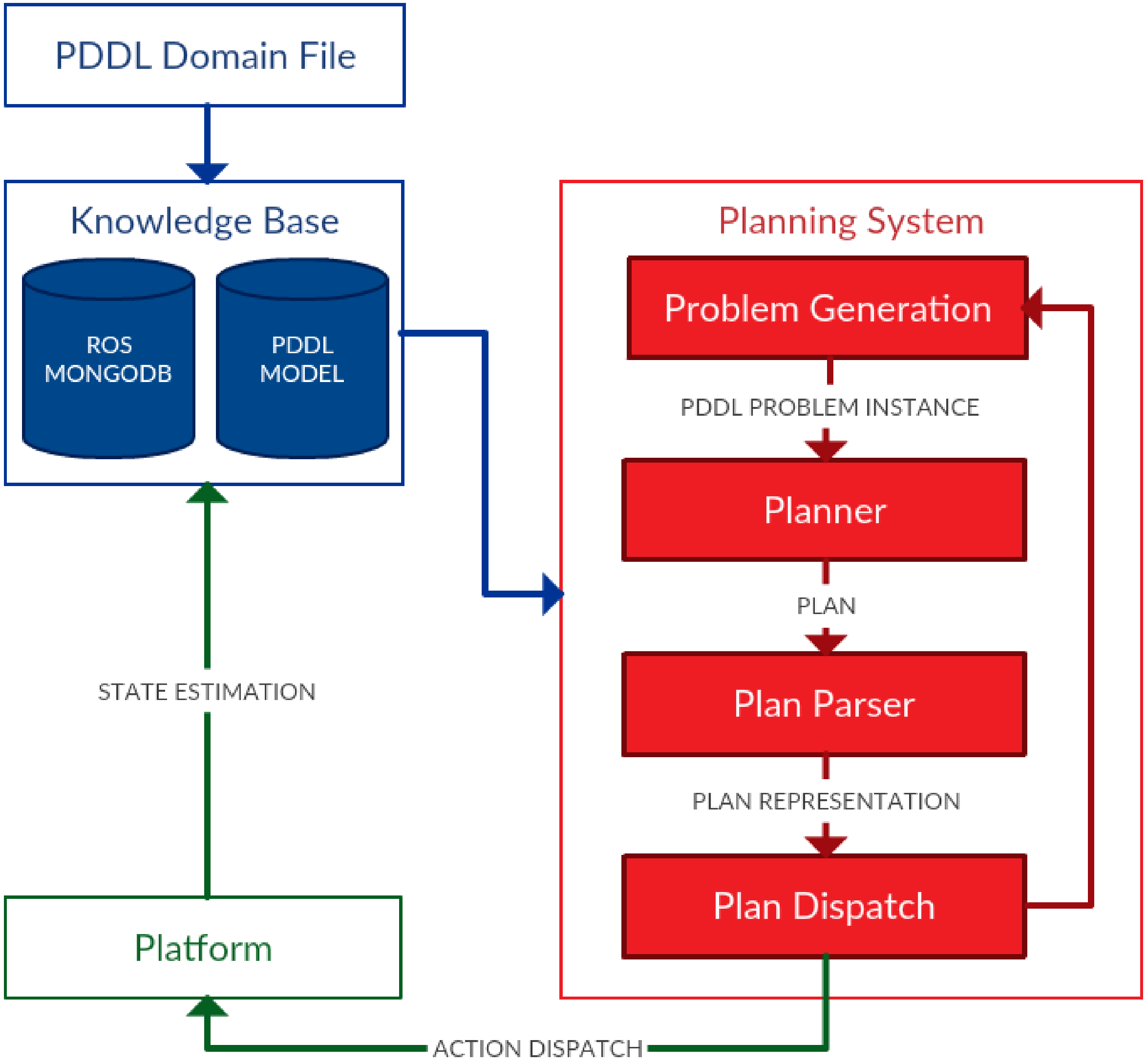
Now the system is more complex:
- PDDL model is continuously updated from sensor data.
- problem file is automatically generated.
- the planner generates a plan.
- the plan is dispatched action-by-action.



ROSPlan: Default Configuration

Now the system is more complex:

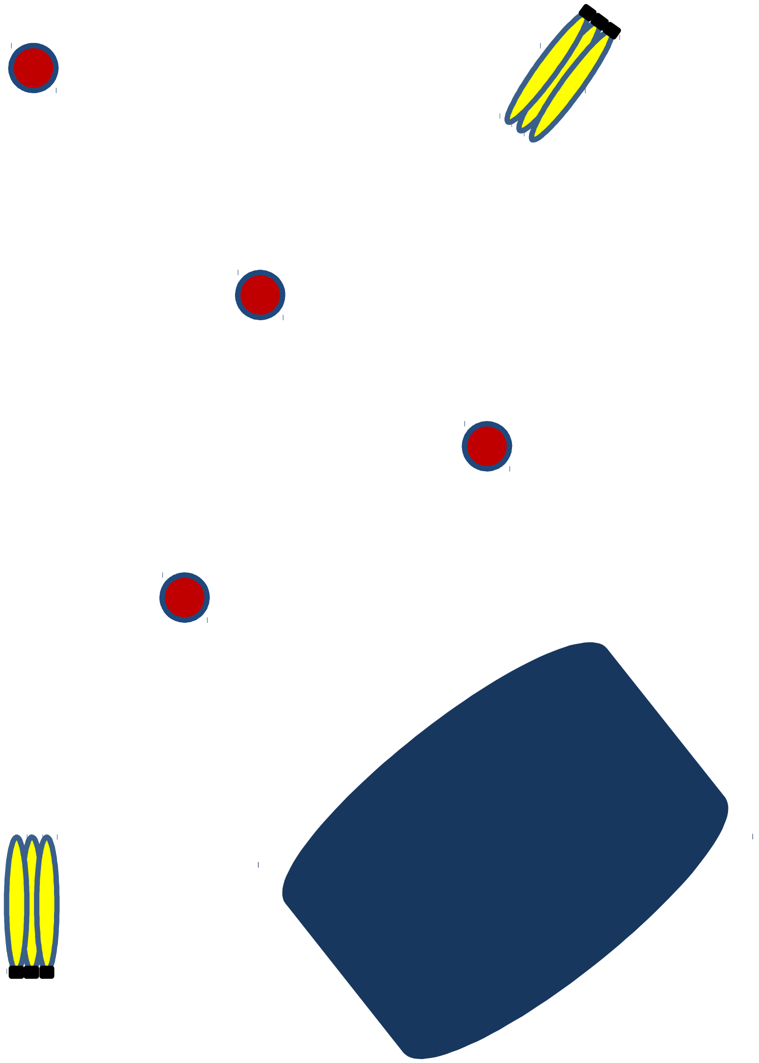
- PDDL model is continuously updated from sensor data.
- problem file is automatically generated.
- the planner generates a plan.
- the plan is dispatched action-by-action.
- feedback on action success and failure.
- the plan is validated against the current model.



Plan Execution 2: Very Simple Temporal Dispatch

The real world requires a temporal and numeric model:
- time and deadlines,
- battery power and consumption,
- direction of sea current, or traffic flow.

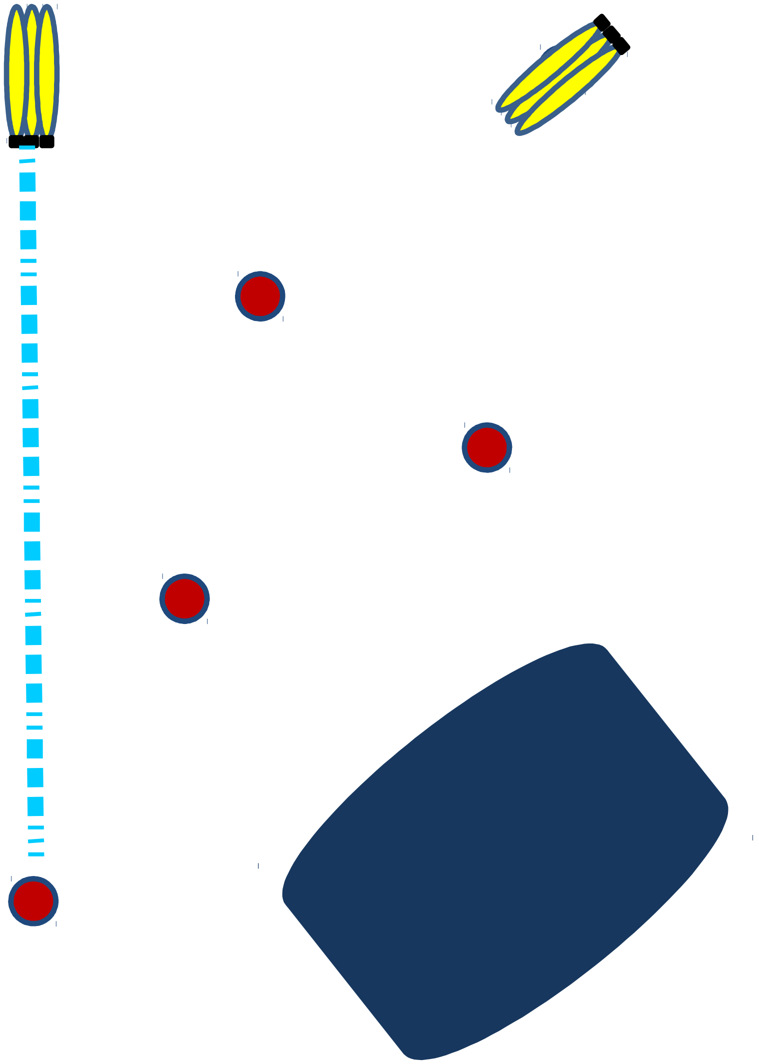
What happens when we add temporal constraints, and try to dispatch the plan as a sequence of actions?



Plan Execution 2: Very Simple Temporal Dispatch

The real world requires a temporal and numeric model:
- time and deadlines,
- battery power and consumption,
- direction of sea current, or traffic flow.

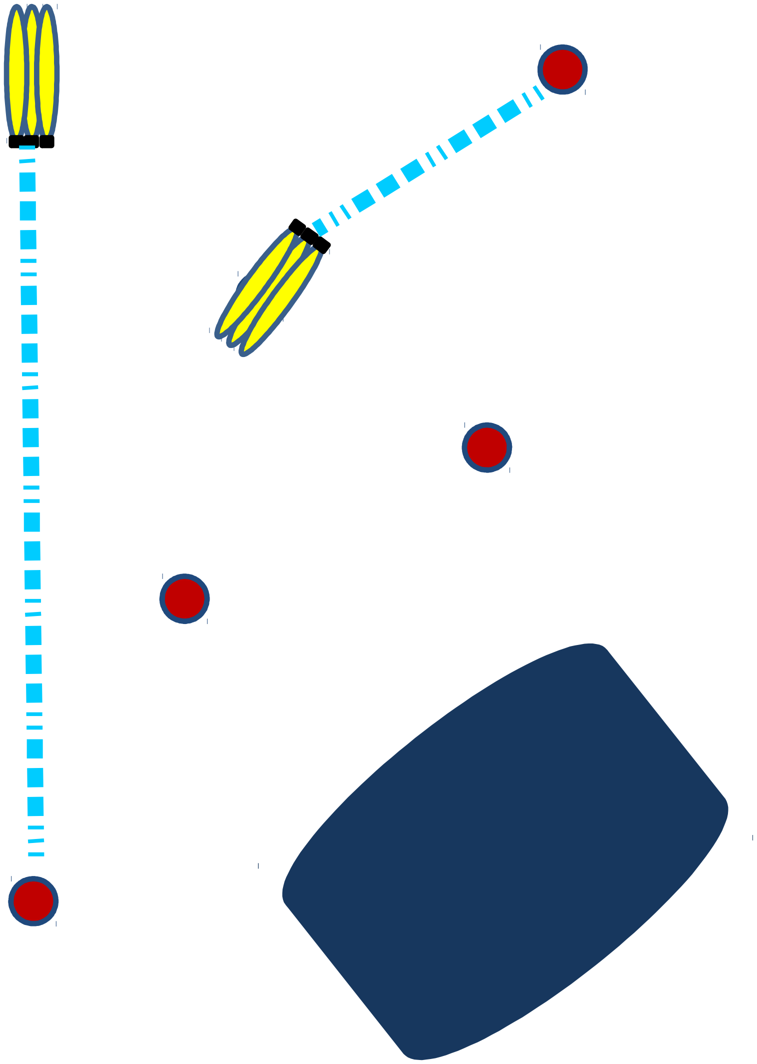
What happens when we add temporal constraints, and try to dispatch the plan as a sequence of actions?



Plan Execution 2: Very Simple Temporal Dispatch

The real world requires a temporal and numeric model:
- time and deadlines,
- battery power and consumption,
- direction of sea current, or traffic flow.

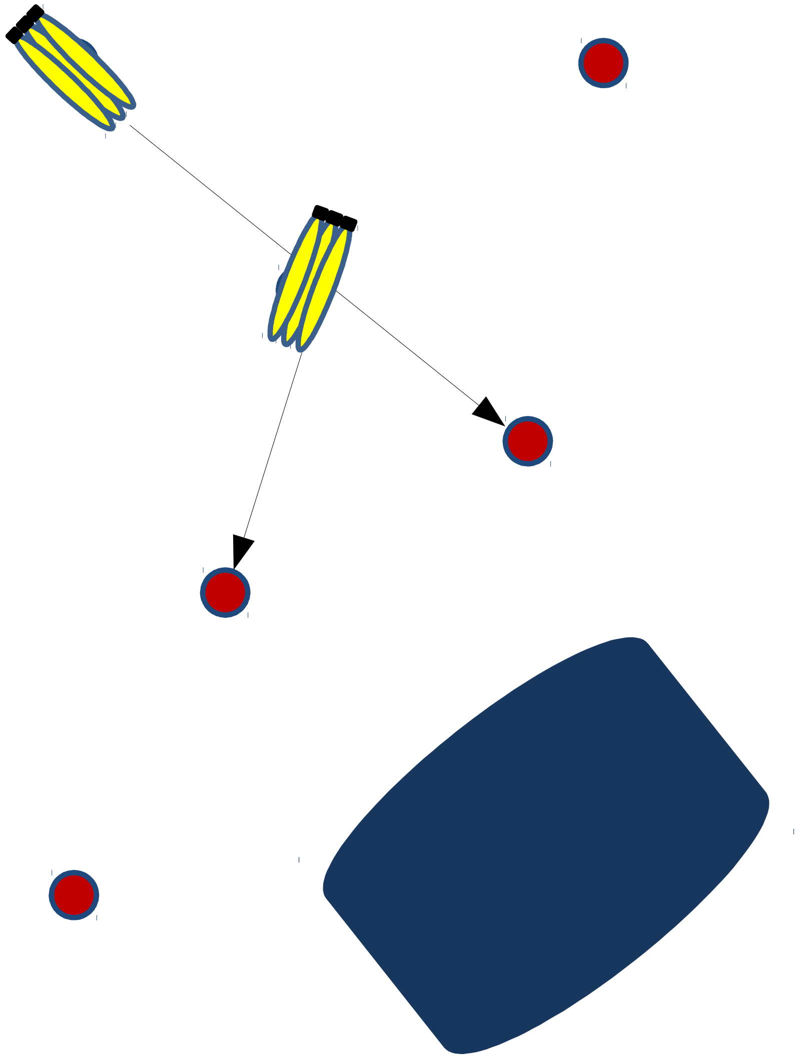
What happens when we add temporal constraints, and try to dispatch the plan as a sequence of actions?



Plan Execution 2: Very Simple Temporal Dispatch

The real world requires a temporal and numeric model:
- time and deadlines,
- battery power and consumption,
- direction of sea current, or traffic flow.

What happens when we add temporal constraints, and try to dispatch the plan as a sequence of actions?

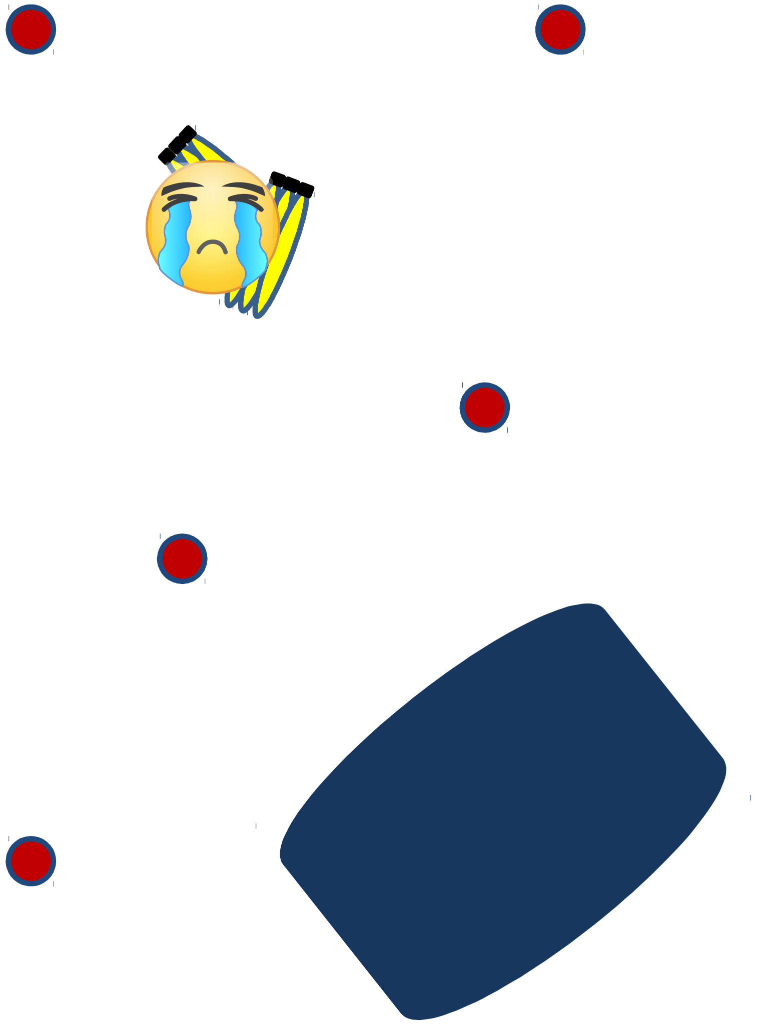


Plan Execution 2: Very Simple Temporal Dispatch

The real world requires a temporal and numeric model:
- time and deadlines,
- battery power and consumption,
- direction of sea current, or traffic flow.

What happens when we add temporal constraints, and try to dispatch the plan as a sequence of actions?

The plan is not only less efficient, but it may become incorrect and unsafe!



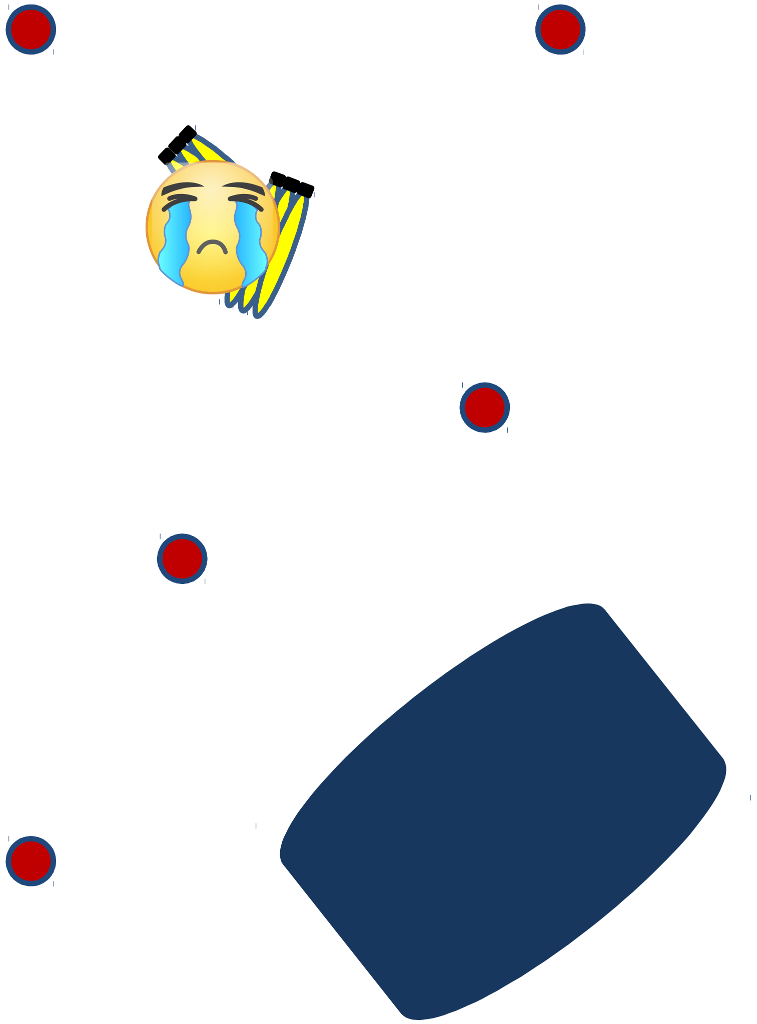
Plan Execution 2: Very Simple Temporal Dispatch

The real world requires a temporal and numeric model:
- time and deadlines,
- battery power and consumption,
- direction of sea current, or traffic flow.

What happens when we add temporal constraints, and try to dispatch the plan as a sequence of actions?

The plan is not only less efficient, but it may become incorrect and unsafe!

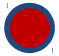
The plan execution loop could instead dispatch actions at their *estimated* timestamps.



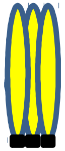
Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.



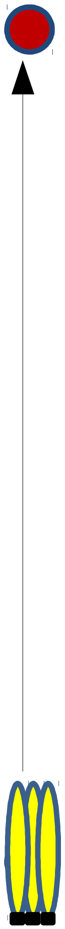
0.000: (goto_waypoint wp1) [10.0]
10.01: (goto_waypoint wp2) [14.3]
24.32: clean_chain wp2) [60.0]



Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.



0.000: (goto_waypoint wp1) [10.0]
10.01: (goto_waypoint wp2) [14.3]
24.32: clean_chain wp2) [60.0]

Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

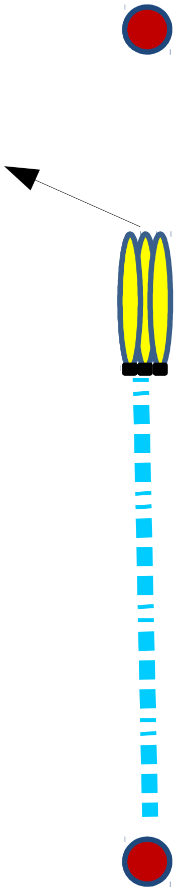
However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.

0.000: (goto_waypoint wp1) [10.0]
10.01: (goto_waypoint wp2) [14.3]
24.32: clean_chain wp2) [60.0]

Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.



0.000: (goto_waypoint wp1) [10.0]
10.01: (goto_waypoint wp2) [14.3]
24.32: clean_chain wp2) [60.0]

Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

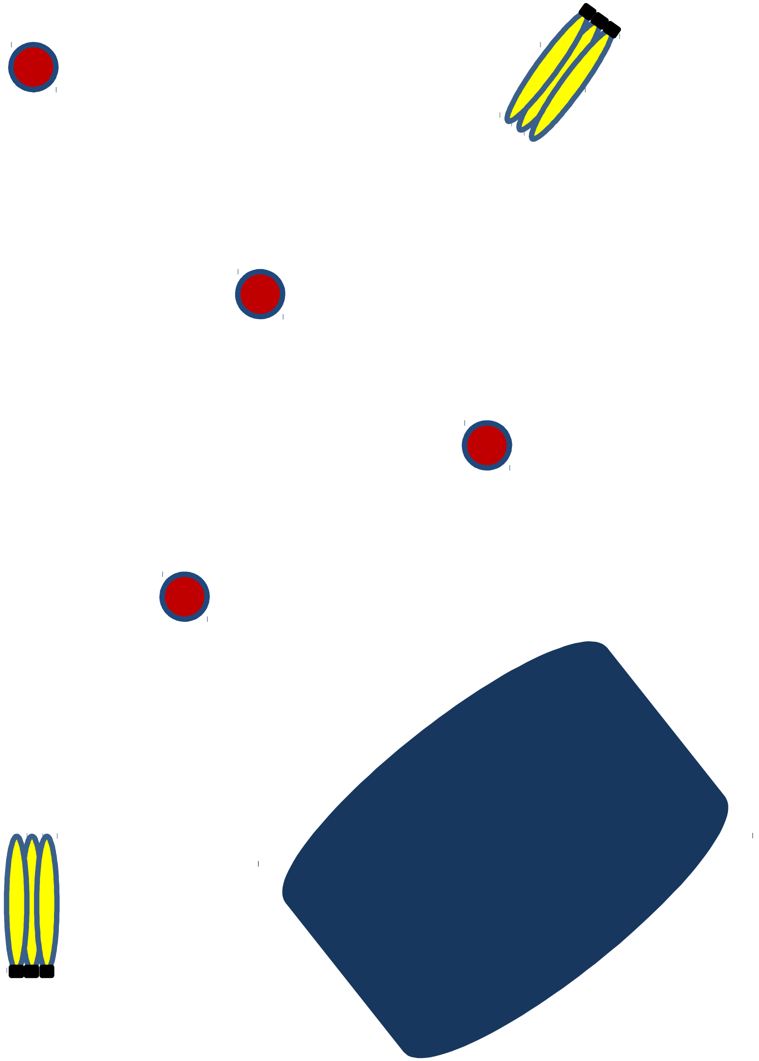
However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.

The plan execution loop could dispatch actions, while respecting the causal ordering between actions.

0.000: (goto_waypoint wp1) [10.0]
10.01: (goto_waypoint wp2) [14.3]
24.32: clean_chain wp2) [60.0]

Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.



However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.

The plan execution loop could dispatch actions, while respecting the causal ordering between actions.

However, some plans require *temporal coordination* between actions, and the controllable durations might be very far apart.

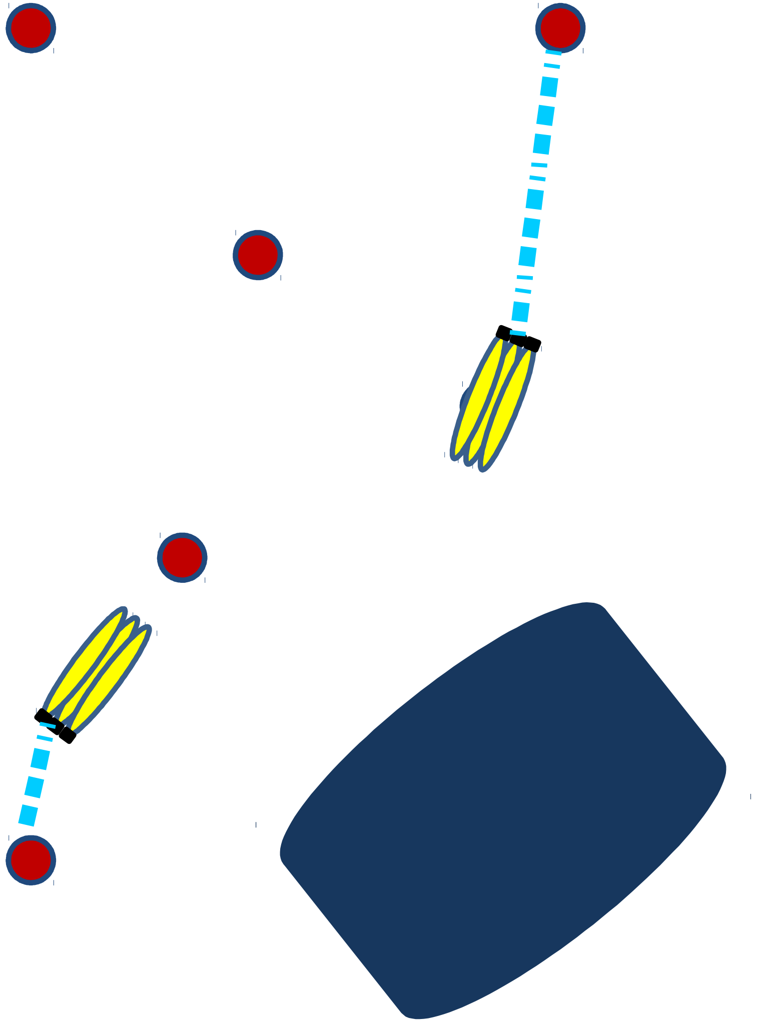
Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.

The plan execution loop could dispatch actions, while respecting the causal ordering between actions.

However, some plans require *temporal coordination* between actions, and the controllable durations might be very far apart.



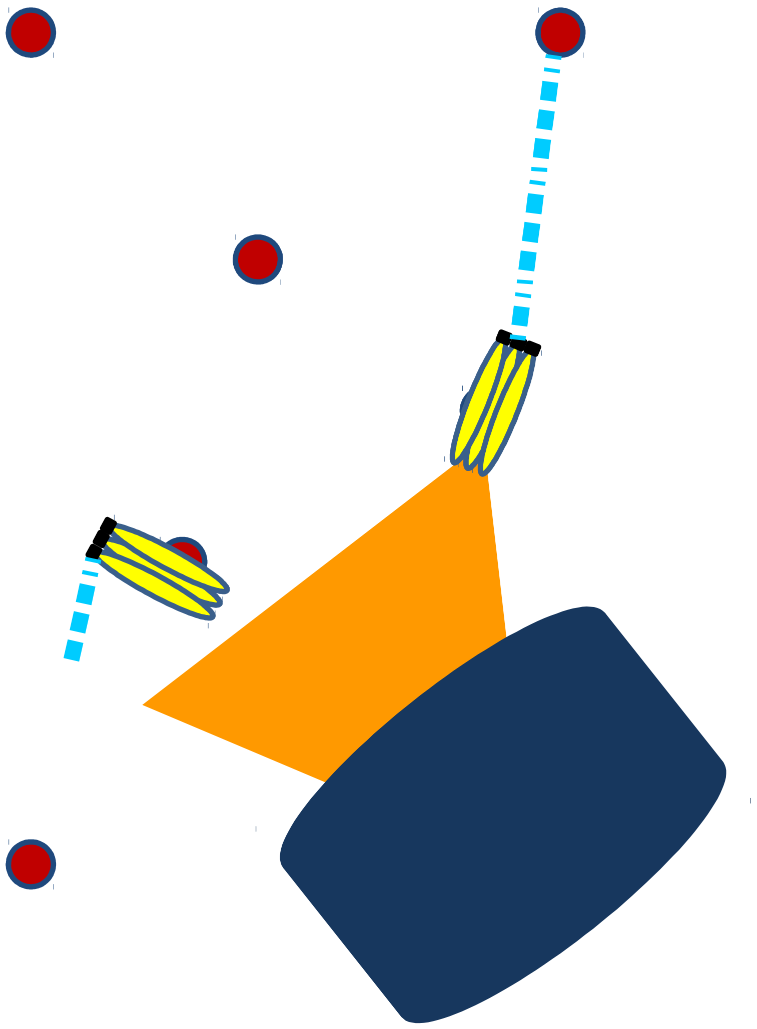
Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.

The plan execution loop could dispatch actions, while respecting the causal ordering between actions.

However, some plans require *temporal coordination* between actions, and the controllable durations might be very far apart.



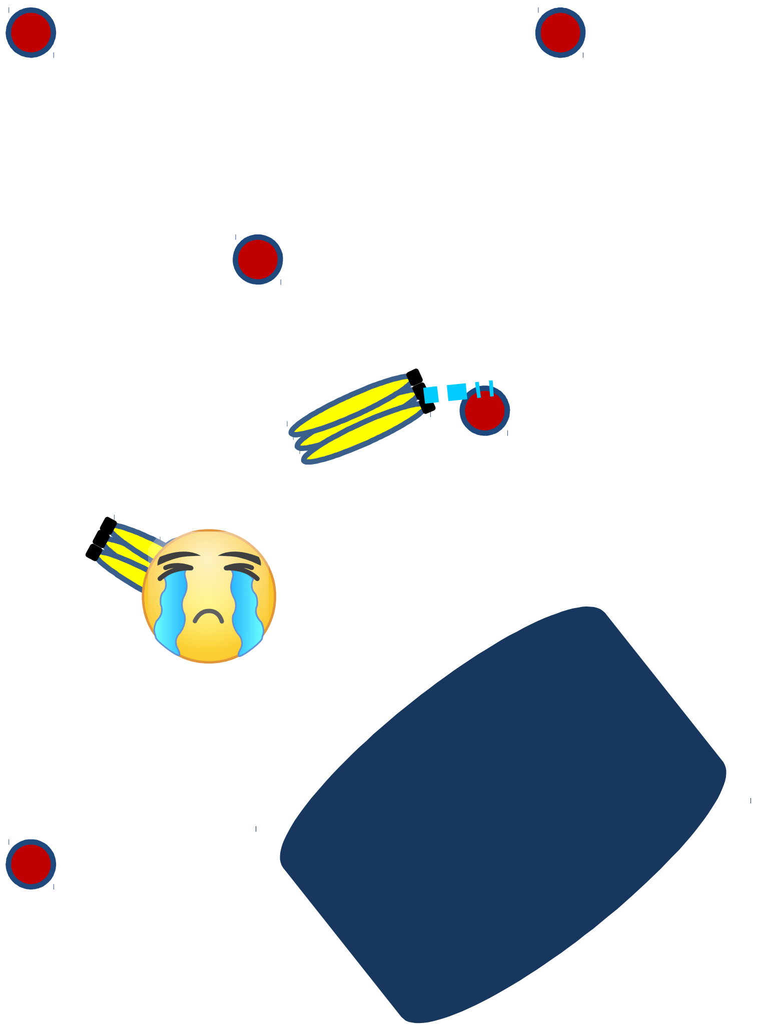
Temporal Constraints

The plan execution loop could instead dispatch actions at their *estimated* timestamps.

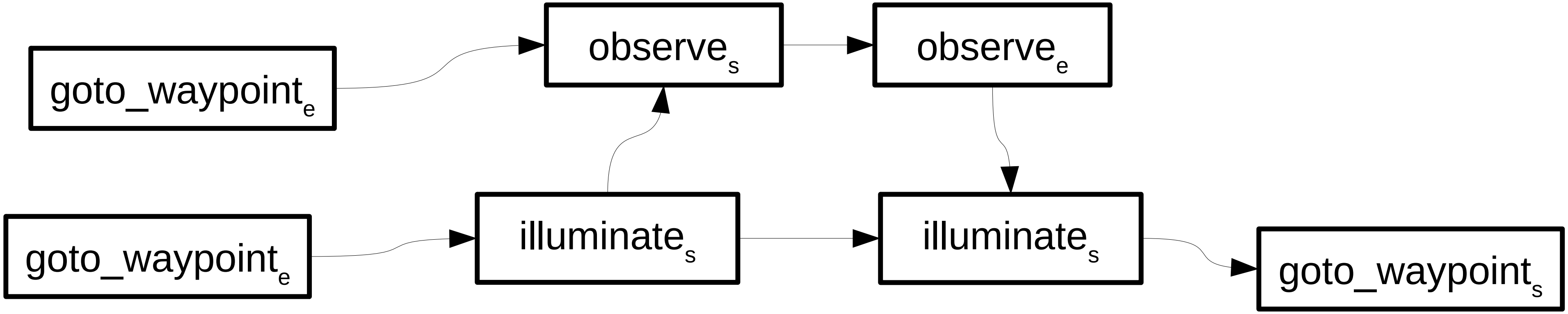
However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.

The plan execution loop could dispatch actions, while respecting the causal ordering between actions.

However, some plans require *temporal coordination* between actions, and the controllable durations might be very far apart.



Temporal Constraints

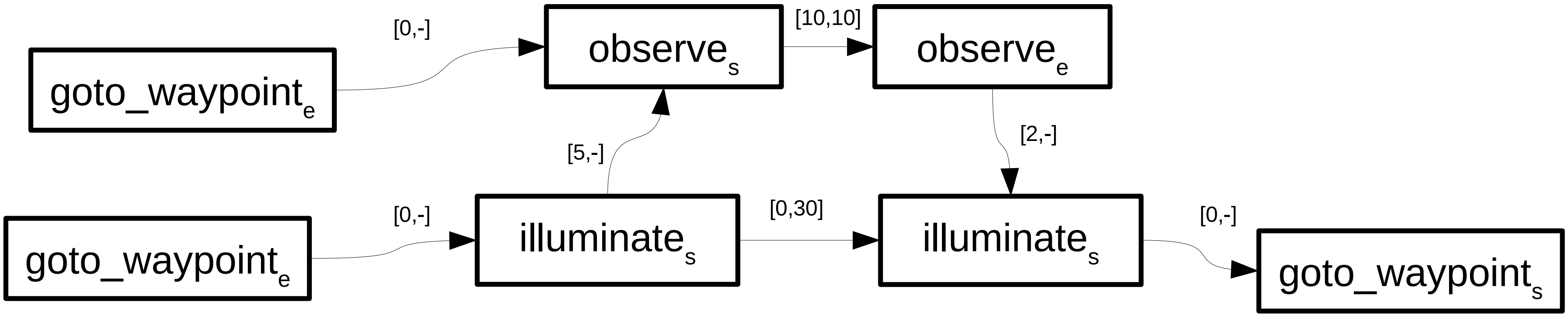


The temporal plan in which every action and duration can be controlled could be represented as a Simple Temporal Network.

The real upper and lower bounds, and ordering constraints on actions can be represented explicitly.

With this representation, the system is able to dispatch actions at times to maintain the consistency of the STN.

Temporal Constraints

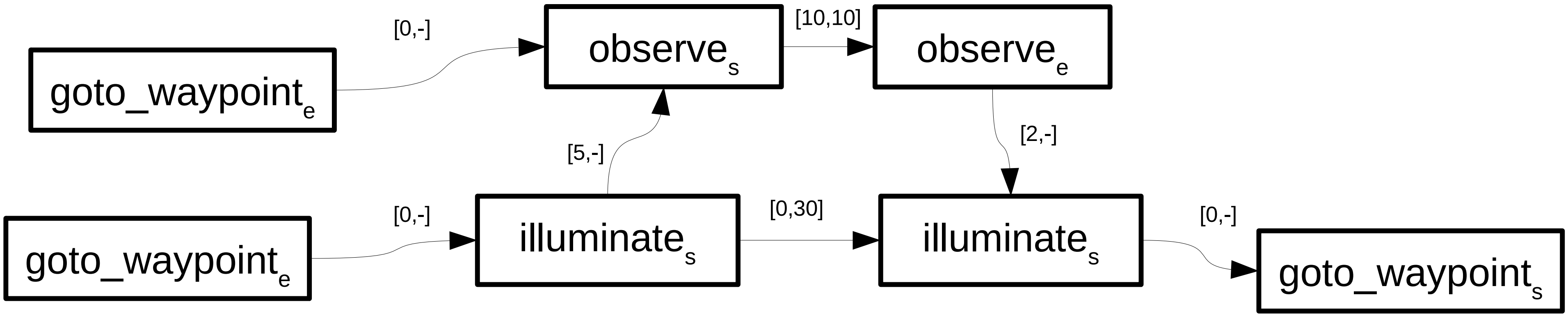


The temporal plan in which every action and duration can be controlled could be represented as a Simple Temporal Network.

The real upper and lower bounds, and ordering constraints on actions can be represented explicitly.

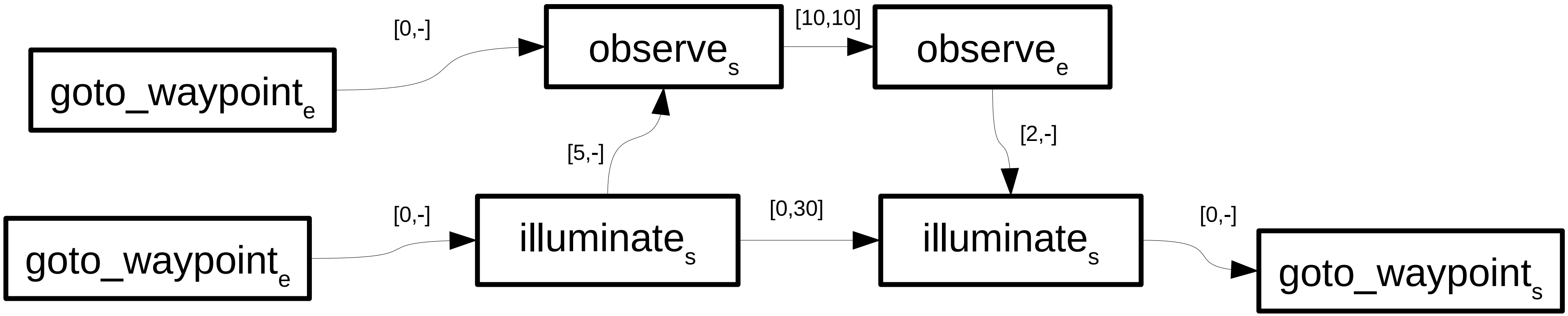
With this representation, the system is able to dispatch actions at times to maintain the consistency of the STN.

Temporal Constraints



The temporal plan with uncontrollable durations can also be represented as a *Temporal Plan Network* (TPN)*. [Vidal & Fargier 1999]*

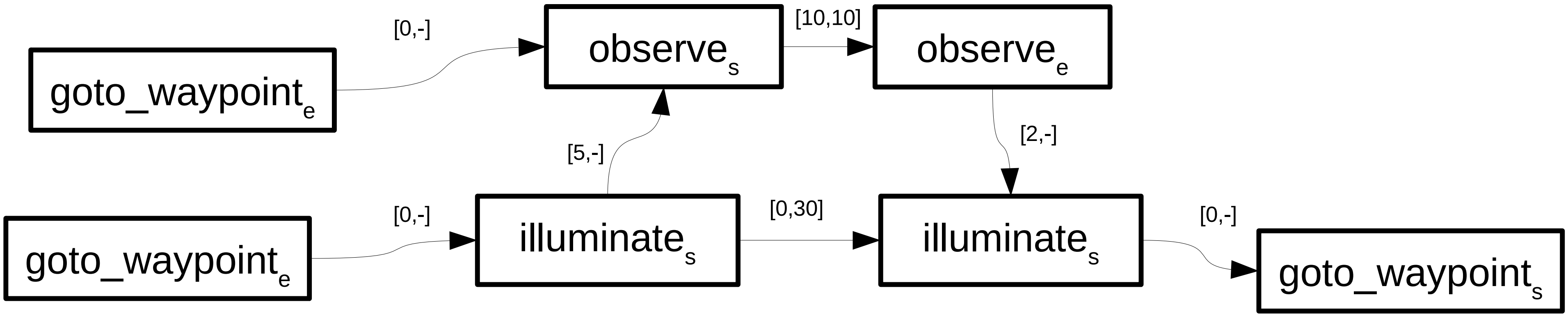
Temporal Constraints



The temporal plan with uncontrollable durations can also be represented as a *Temporal Plan Network* (TPN). *[Vidal & Fargier 1999]*

The time-points of a TPN are divided into *activated* time-points whose dispatch time can be chosen by the agent, and *received* time-points whose time is unpredictable.

Temporal Constraints



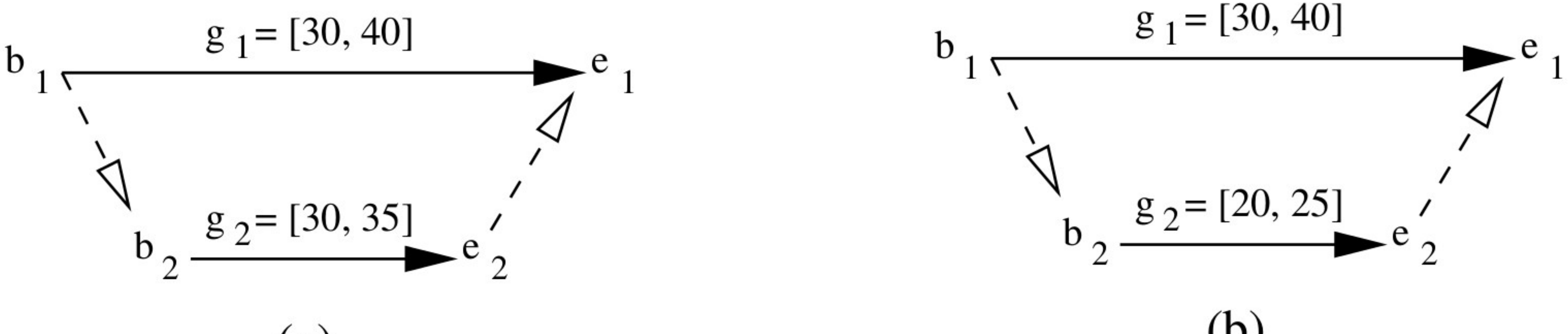
The temporal plan with uncontrollable durations can also be represented as a *Temporal Plan Network* (TPN)*. [Vidal & Fargier 1999]*

The time-points of a TPN are divided into *activated* time-points whose dispatch time can be chosen by the agent, and *received* time-points whose time is unpredictable.

The *Simple Temporal Problem under Uncertainty* (STPU) described by a TPN might be strongly, weakly, or dynamically controllable. *[Ciamatti, Micheli et al. 2016]*

STPUs: Strong controllability

An STPU is strongly controllable iff:
- the agent can commit to a time for all activated time-points,
- such that for any possible time for received time points,
- the temporal constraints are not violated.

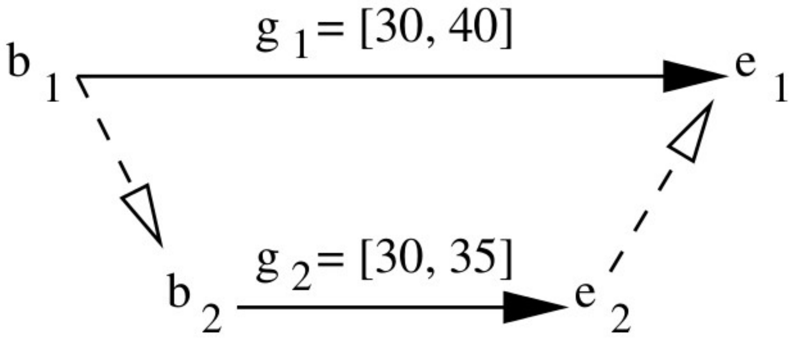


(a)

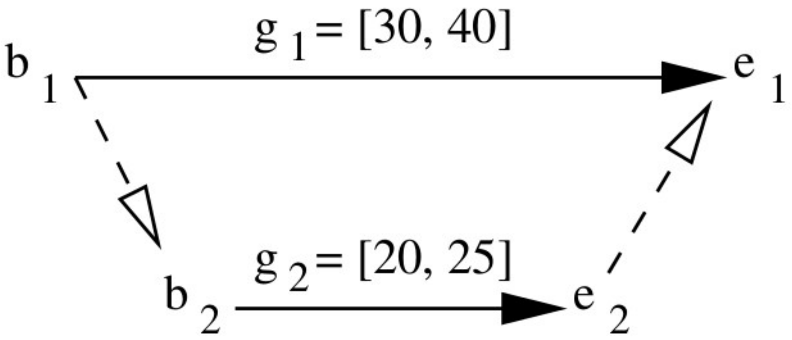
(b)

STPUs: Strong controllability

An STPU is strongly controllable iff:
- the agent can commit to a time for all activated time-points,
- such that for any possible time for received time points,
- the temporal constraints are not violated.



(a) ✗

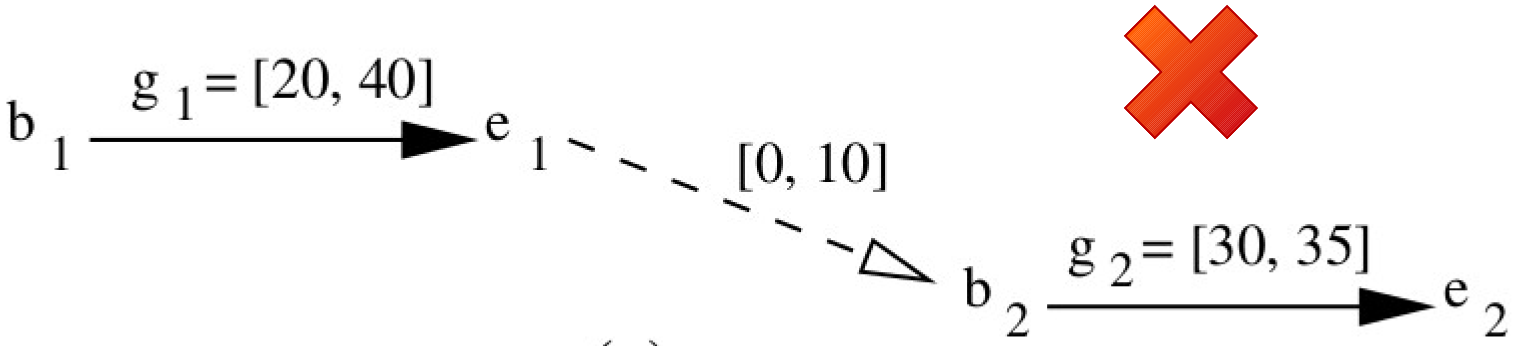


(b) ✓

Setting *t(b1) == t(b2)* will always obey the temporal constraints.

STPUs: Strong controllability

An STPU is strongly controllable iff:
- the agent can commit to a time for all activated time-points,
- such that for any possible time for received time points,
- the temporal constraints are not violated.



(c)

The STPU is not strongly controllable, but it is obviously executable.
We need dynamic controllability.

STPUs: Dynamic controllability

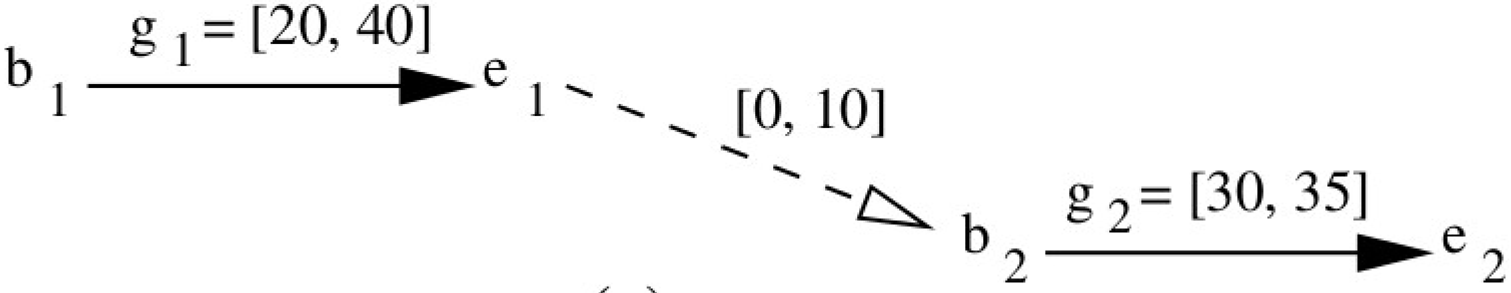
An STPU is dynamically controllable iff:
- at any point in time, the execution so far is ensured to extend to a complete solution such that the temporal constraints are not violated.

In this case, the agent does not have to commit to a time for any activated time points in advance.

STPUs: Dynamic controllability

An STPU is dynamically controllable iff:
- at any point in time, the execution so far is ensured to extend to a complete solution such that the temporal constraints are not violated.

In this case, the agent does not have to commit to a time for any activated time points in advance.

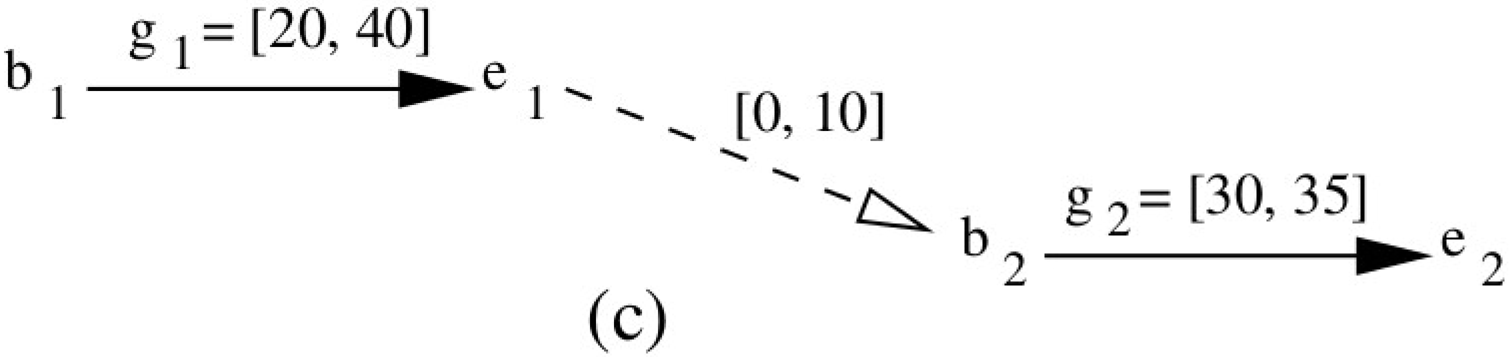


(c)

STPUs: Dynamic controllability

Not all problems will have solutions which have any kind of controllability. This does not mean they are impossible.

To reason about these kinds of issues we need to use a plan representation sufficient to capture the controllable and uncontrollable durations, causal orderings, and temporal constraints.

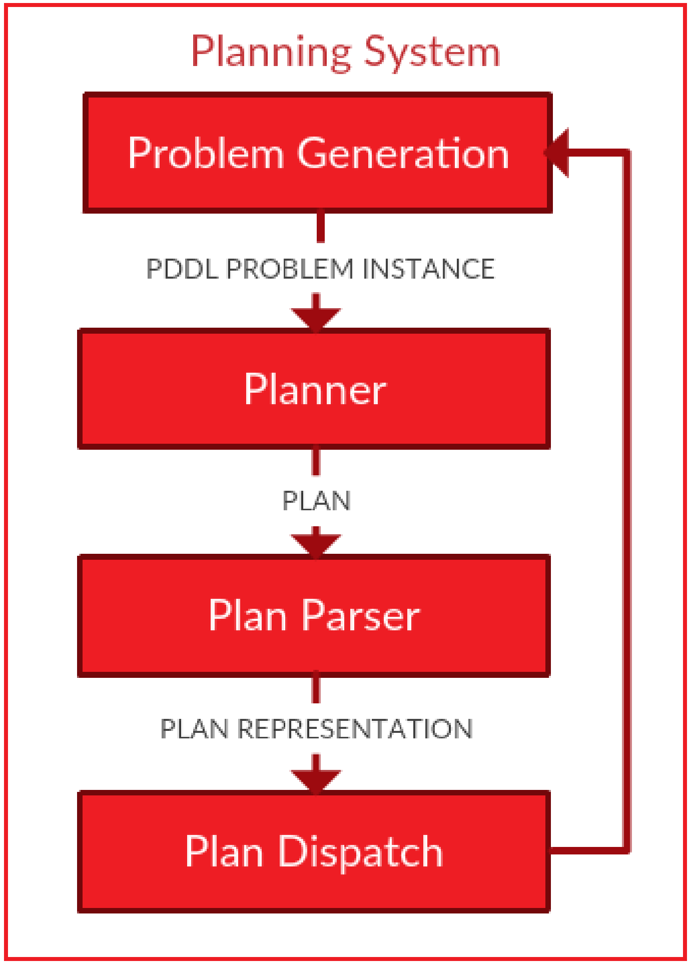


Plan dispatch in ROSPlan

To reason about these kinds of issues we need to use a plan representation sufficient to capture the controllable and uncontrollable durations, causal orderings, and temporal constraints.

The representation of a plan is coupled with the choice of dispatcher.

The problem generation and planner are not *necessarily* bound by the choice of representation.



Plan Execution 3: Conditional Dispatch

Uncertainty and lack of knowledge is a huge part of AI Planning for Robotics.

- Actions might fail or succeed.
- The effects of an action can be non-deterministic.
- The environment is dynamic and changing.
- The environment is often initially full of unknowns.

The domain model is *always* incomplete as well as inaccurate.

Uncertainty in AI Planning

- The environment is dynamic and changing.
- The environment is often initially full of unknowns.

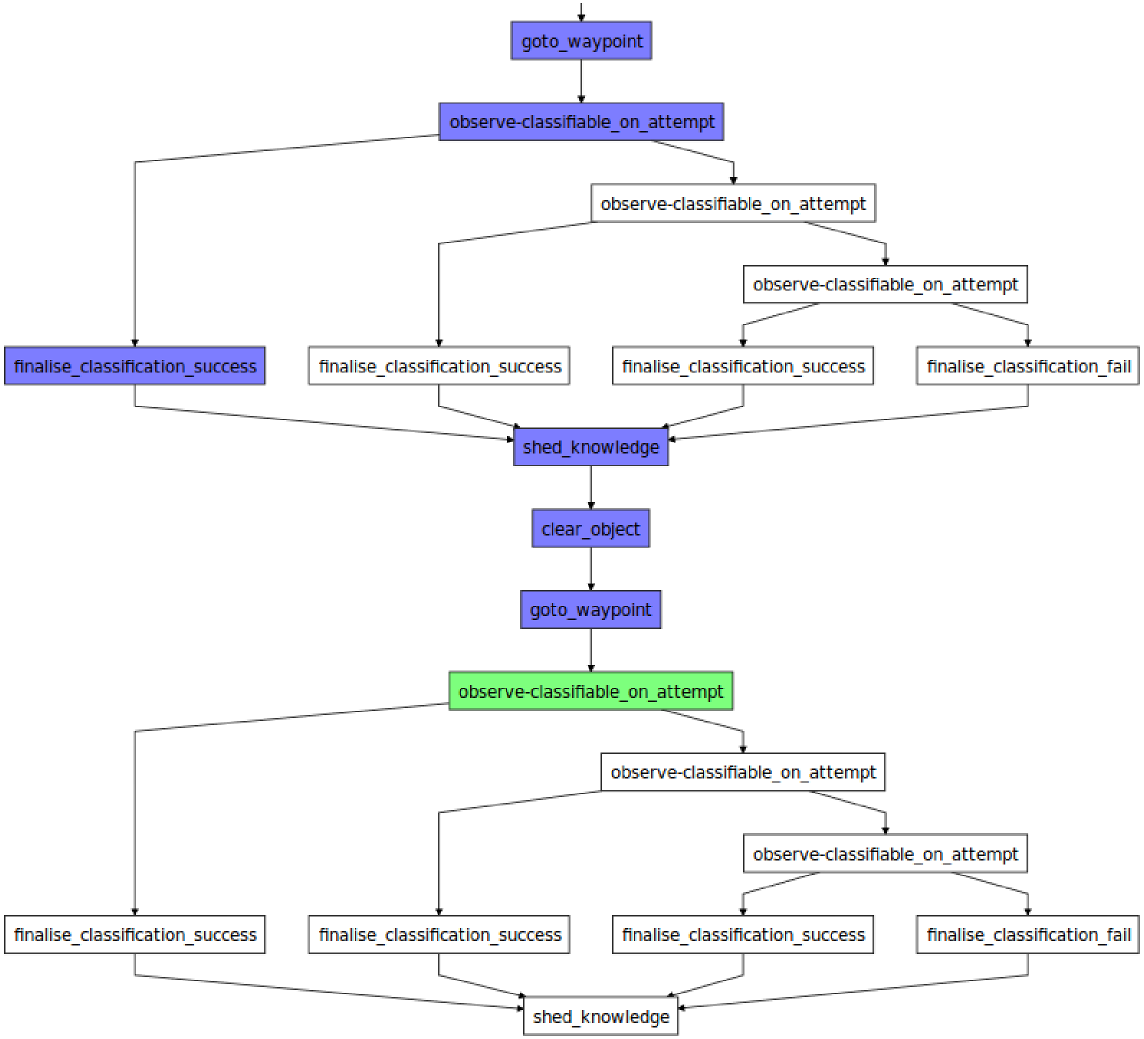
Uncertainty in AI Planning

Some uncertainty can be handled at planning time:

- Fully-Observable Non-deterministic planning.

- Partially-observable Markov decision Process.

- Conditional Planning with Contingent Planners. (e.g. ROSPlan with Contingent-FF)


goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge
clear_object
goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge

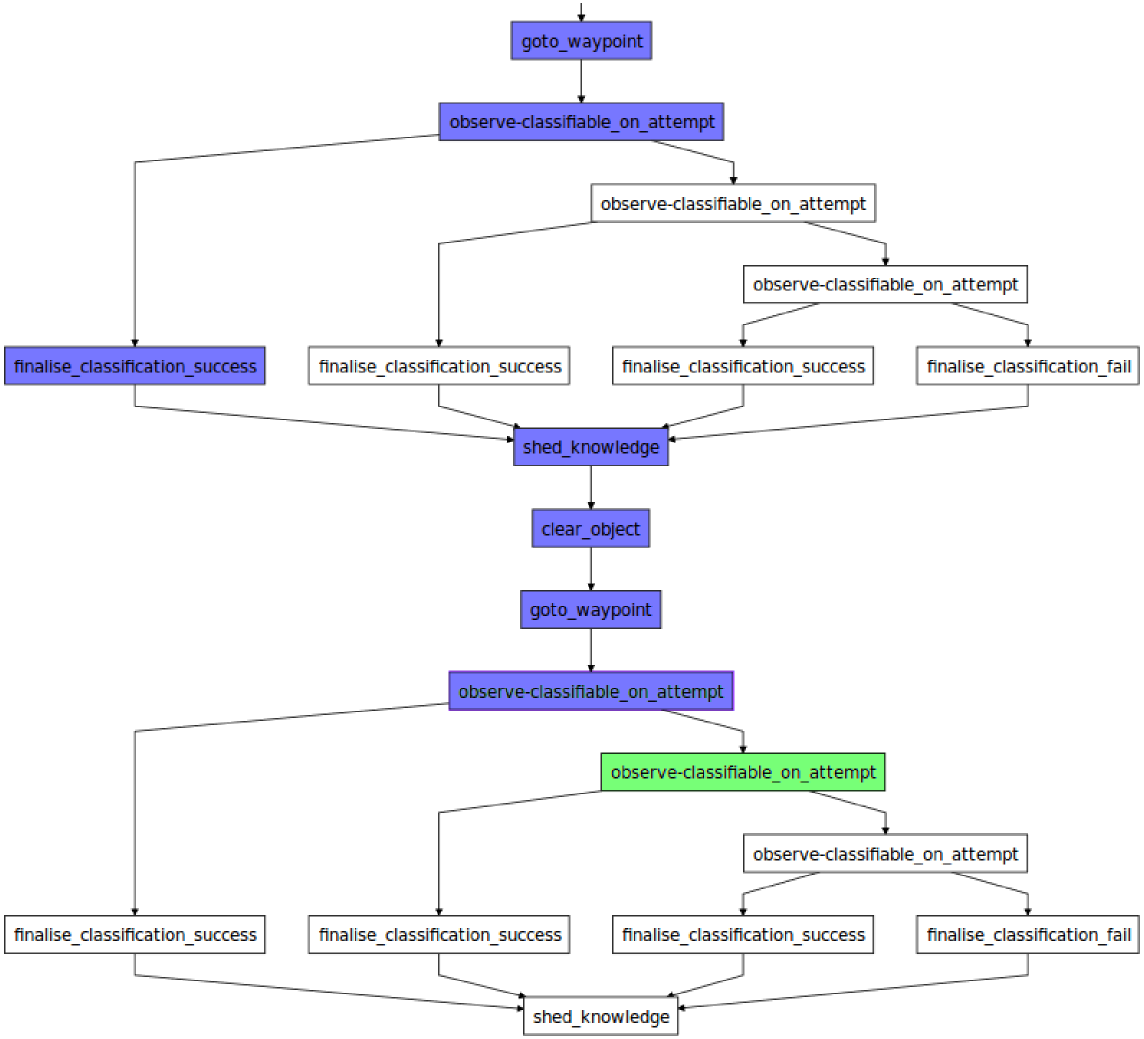
Uncertainty in AI Planning

Some uncertainty can be handled at planning time:

- Fully-Observable Non-deterministic planning.

- Partially-observable Markov decision Process.

- Conditional Planning with Contingent Planners. (e.g. ROSPlan with Contingent-FF)


goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge
clear_object
goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge

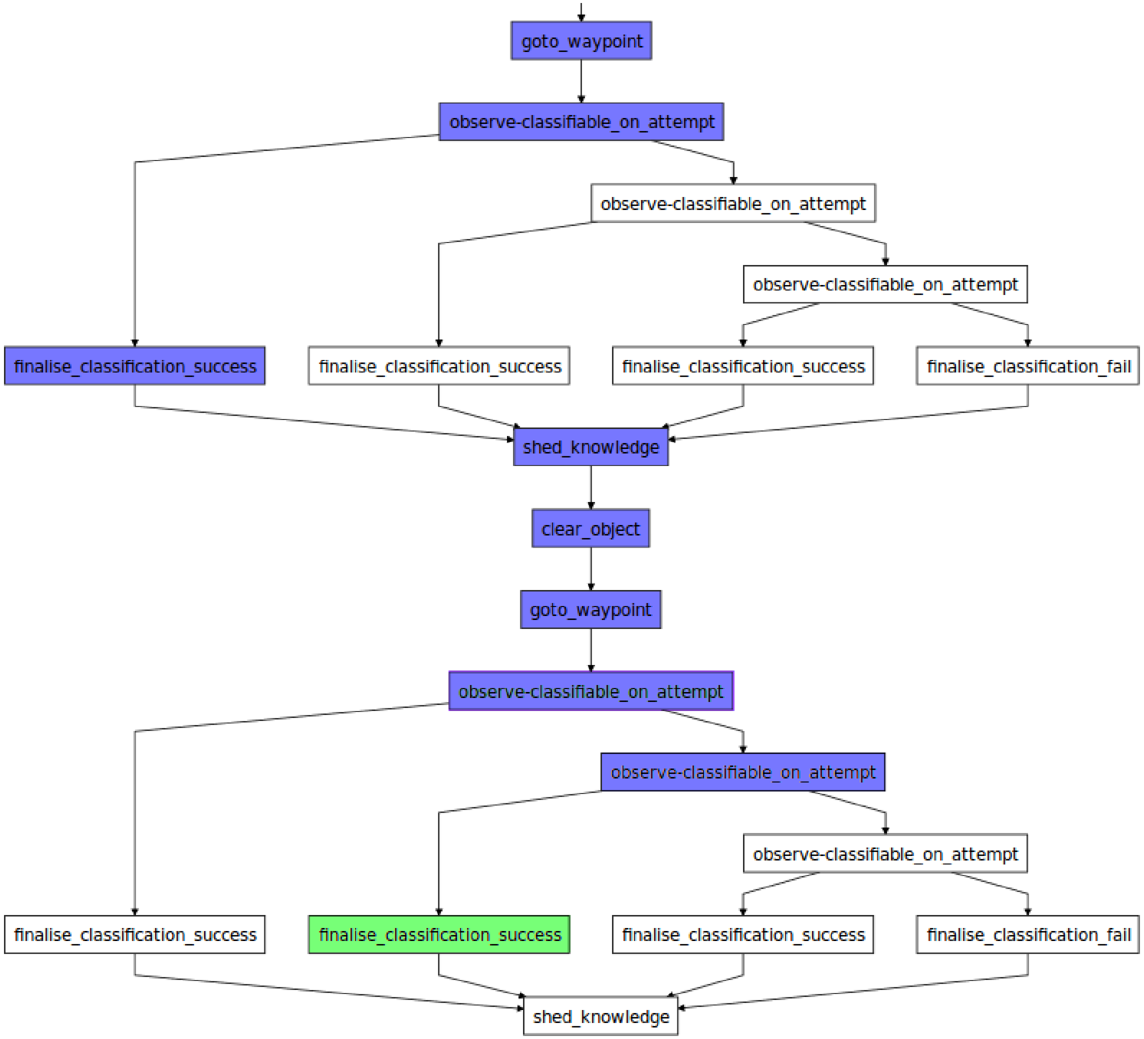
Uncertainty in AI Planning

Some uncertainty can be handled at planning time:

- Fully-Observable Non-deterministic planning.

- Partially-observable Markov decision Process.

- Conditional Planning with Contingent Planners. (e.g. ROSPlan with Contingent-FF)


goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge
clear_object
goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge

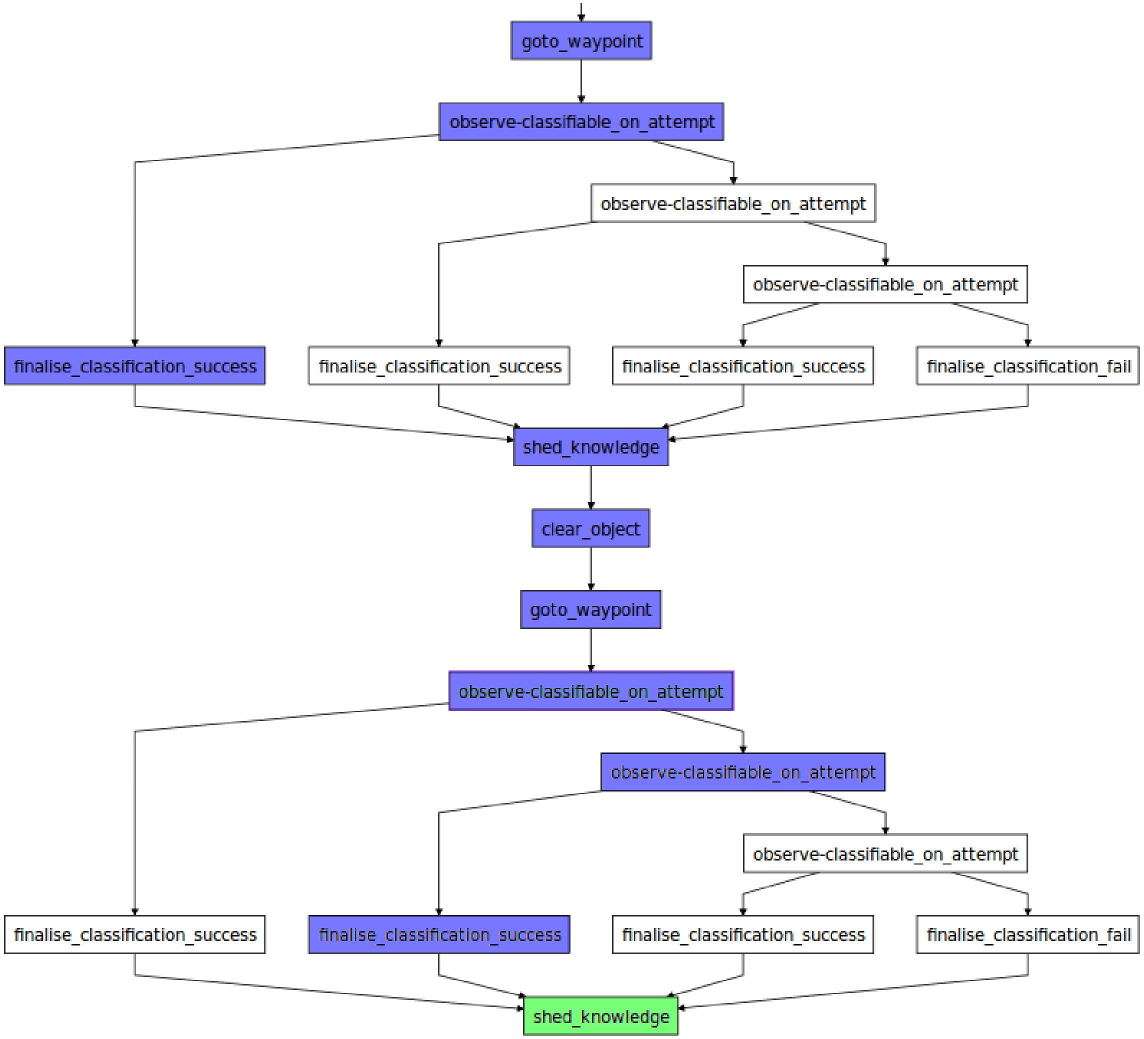
Uncertainty in AI Planning

Some uncertainty can be handled at planning time:

- Fully-Observable Non-deterministic planning.

- Partially-observable Markov decision Process.

- Conditional Planning with Contingent Planners. (e.g. ROSPlan with Contingent-FF)


goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge
clear_object
goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge

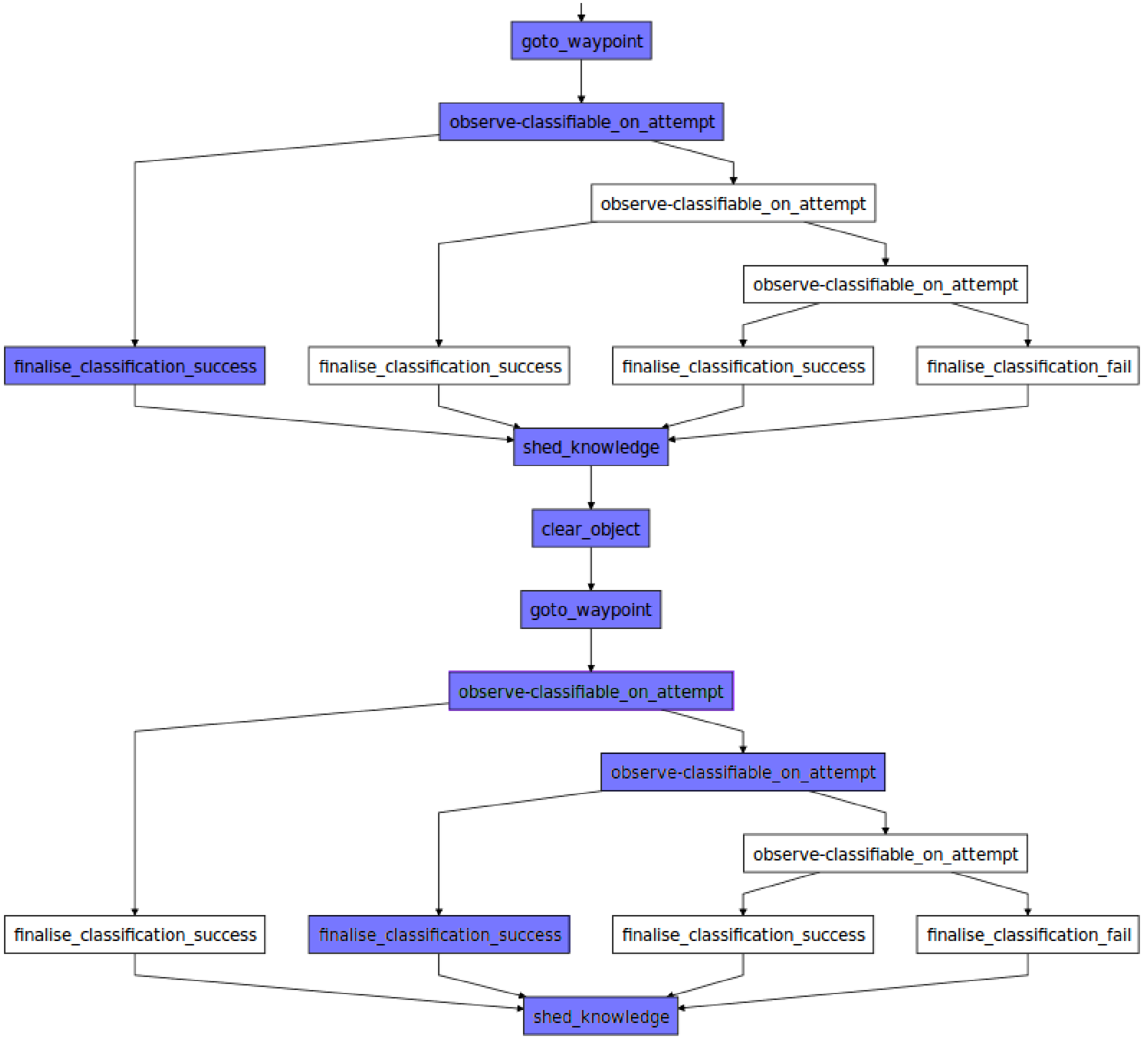
Uncertainty in AI Planning

Some uncertainty can be handled at planning time:

- Fully-Observable Non-deterministic planning.

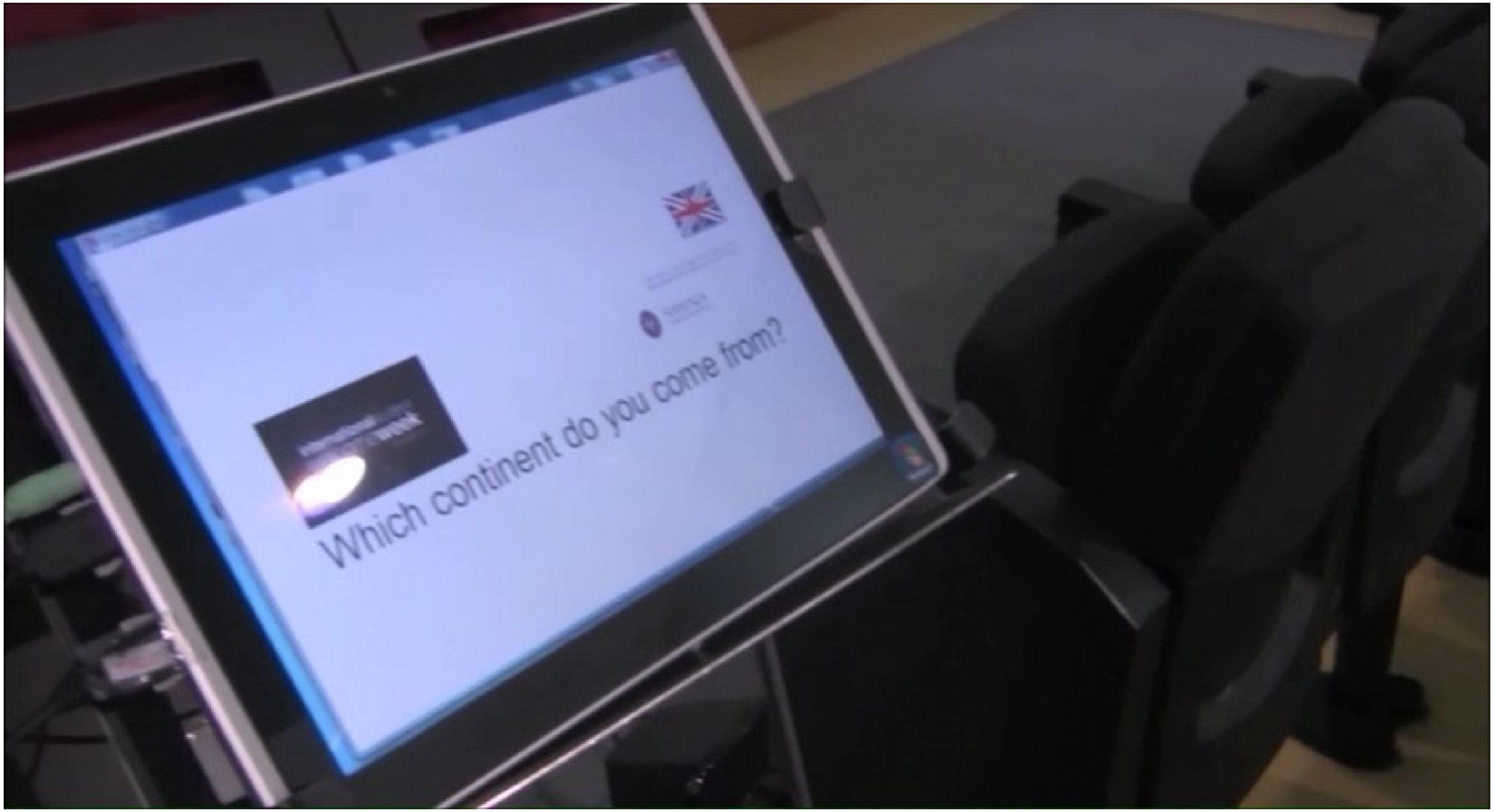
- Partially-observable Markov decision Process.

- Conditional Planning with Contingent Planners. (e.g. ROSPlan with Contingent-FF)


goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge
clear_object
goto_waypoint
observe-classifiable_on_attempt
observe-classifiable_on_attempt
observe-classifiable_on_attempt
finalise_classification_success
finalise_classification_success
finalise_classification_success
finalise_classification_fail
shed_knowledge

Uncertainty in AI Planning

Human Robot Interaction is filled with uncertainties.



Plan Execution 4: Temporal and Conditional Dispatch together

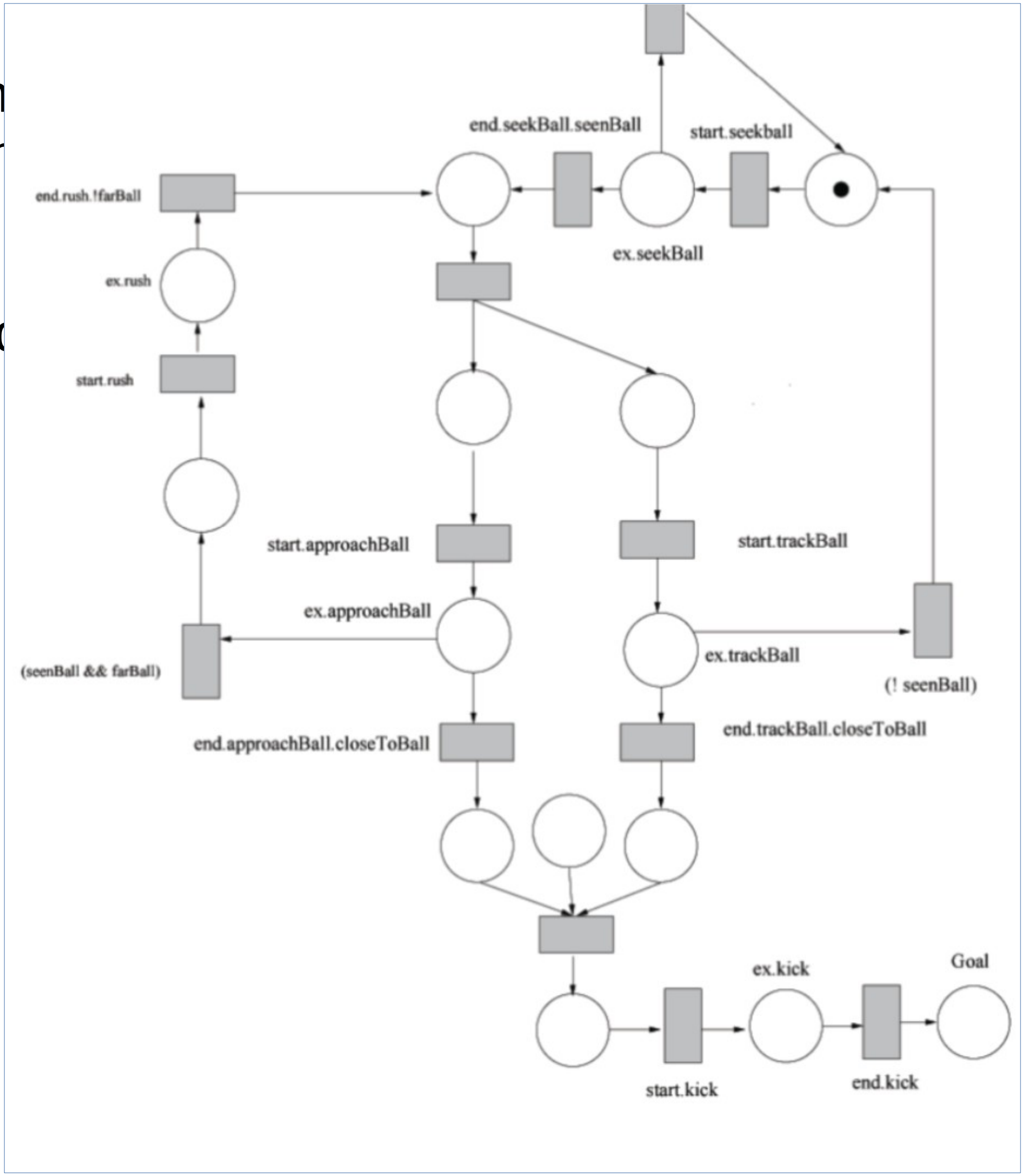
Robotics domains require a combination of temporal and conditional reasoning. Combining these two kinds of uncertainty can result in very complex structures.

There are plan formalisms designed to describe these, e.g.:
- GOLOG plans. *[Claßen et al., 2012]*
- Petri-Net Plans. *[Ziparo et al. 2011]*

Plan Execution 4: Temporal and Conditional Dispatch together

Robotics domains require a combin
reasoning. Combining these two kir
complex structures.

There are plan formalisms designec
- GOLOG plans. *[Claßen et al., 2012]*
- Petri-Net Plans. *[Ziparo et al. 2011]*



Plan Execution 4: Temporal and Conditional Dispatch together

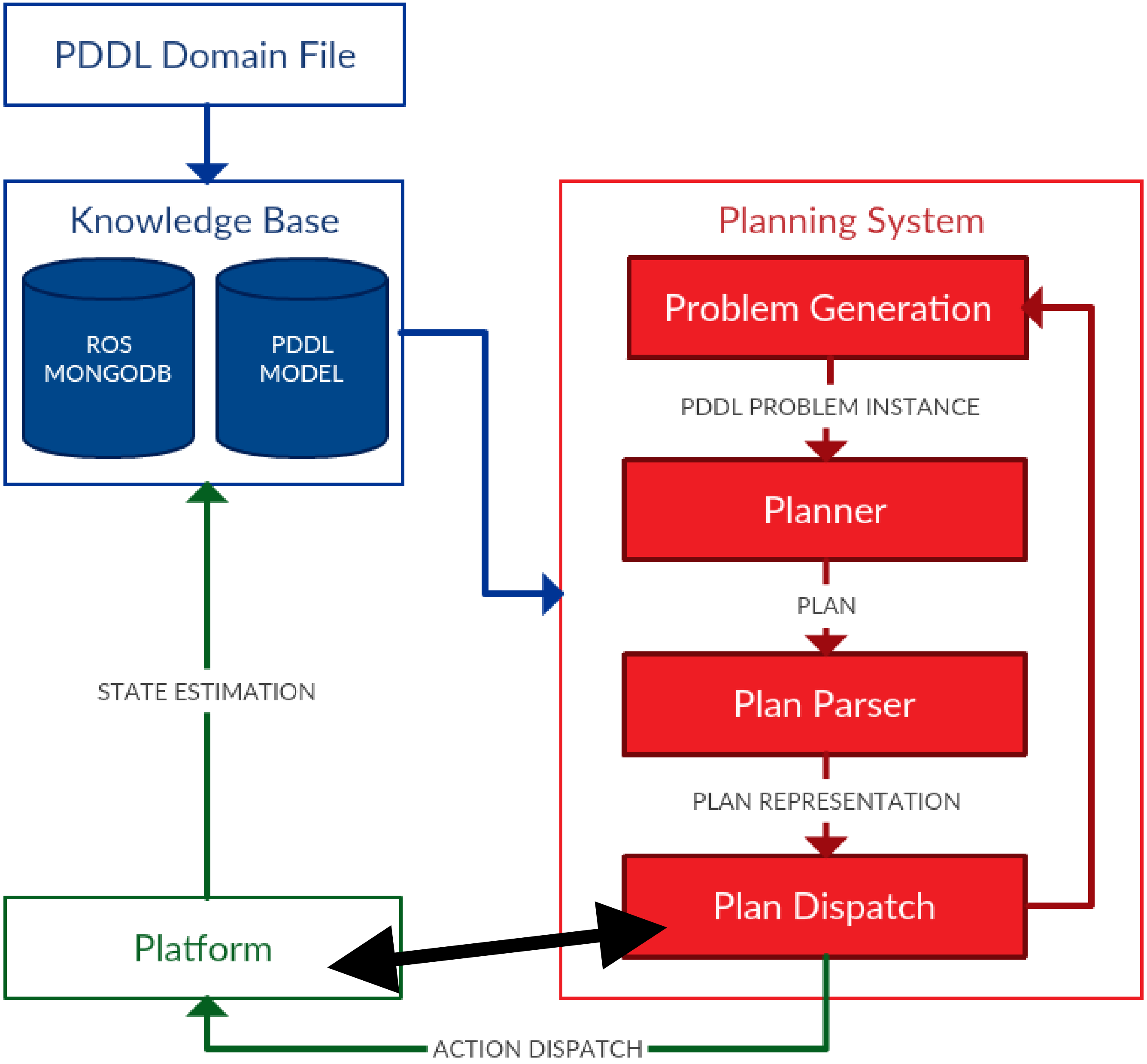
Robotics domains require a combination of temporal and conditional reasoning. Combining these two kinds of uncertainty can result in very complex structures.

There are plan formalisms designed to describe these, e.g.:
- GOLOG plans. *[Claßen et al., 2012]*
- Petri-Net Plans. *[Ziparo et al. 2011]*

ROSPlan is integrated with the PNPRos library for the representation and execution of Petri-Net plans. *[Sanelli et al. 2017]*

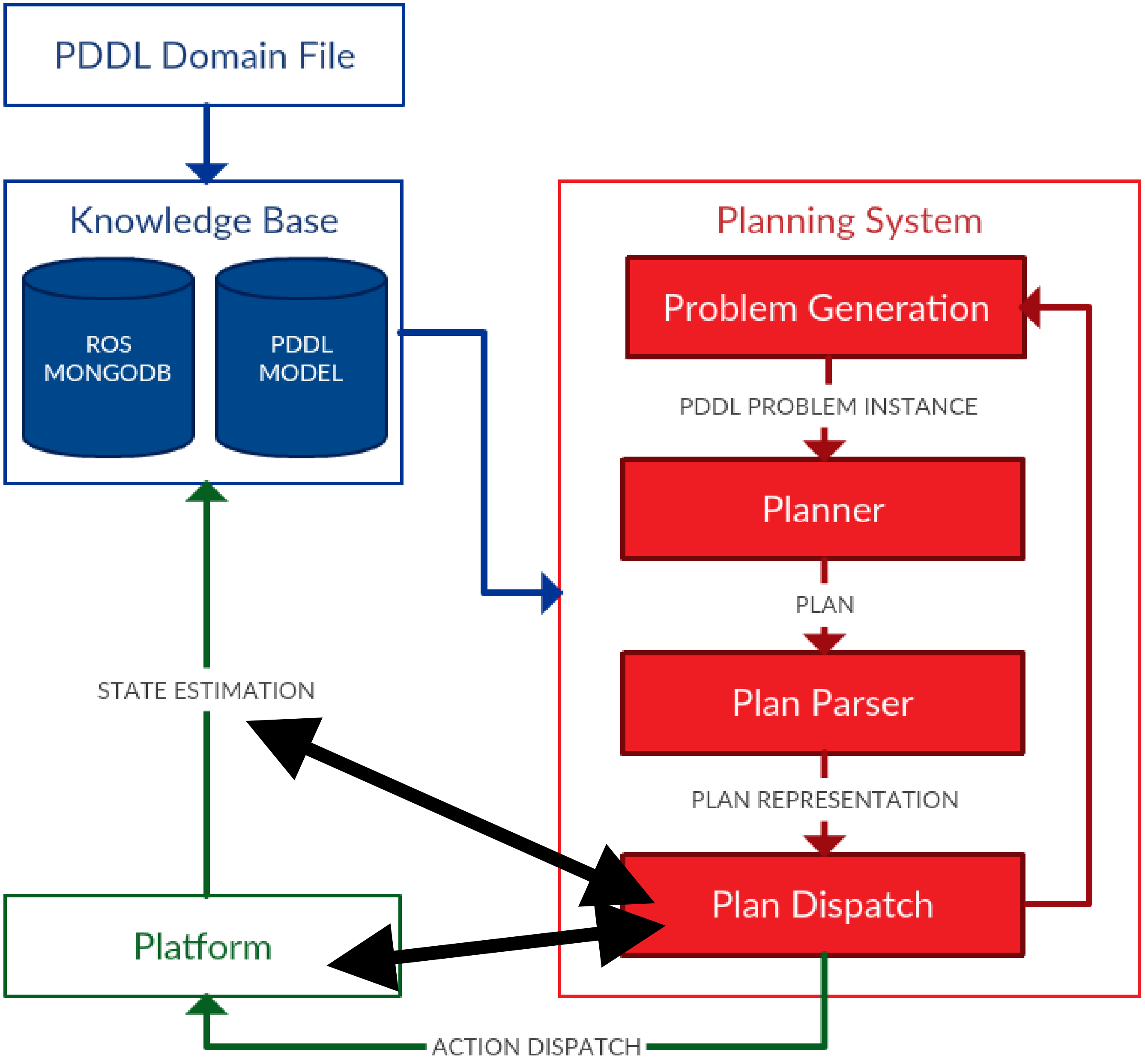
Summary of Very Simple Plan Execution

Plan Execution depends upon many components in the system. Changing any one of which will change the robot behaviour, and change the criteria under which the plan will succeed or fail.



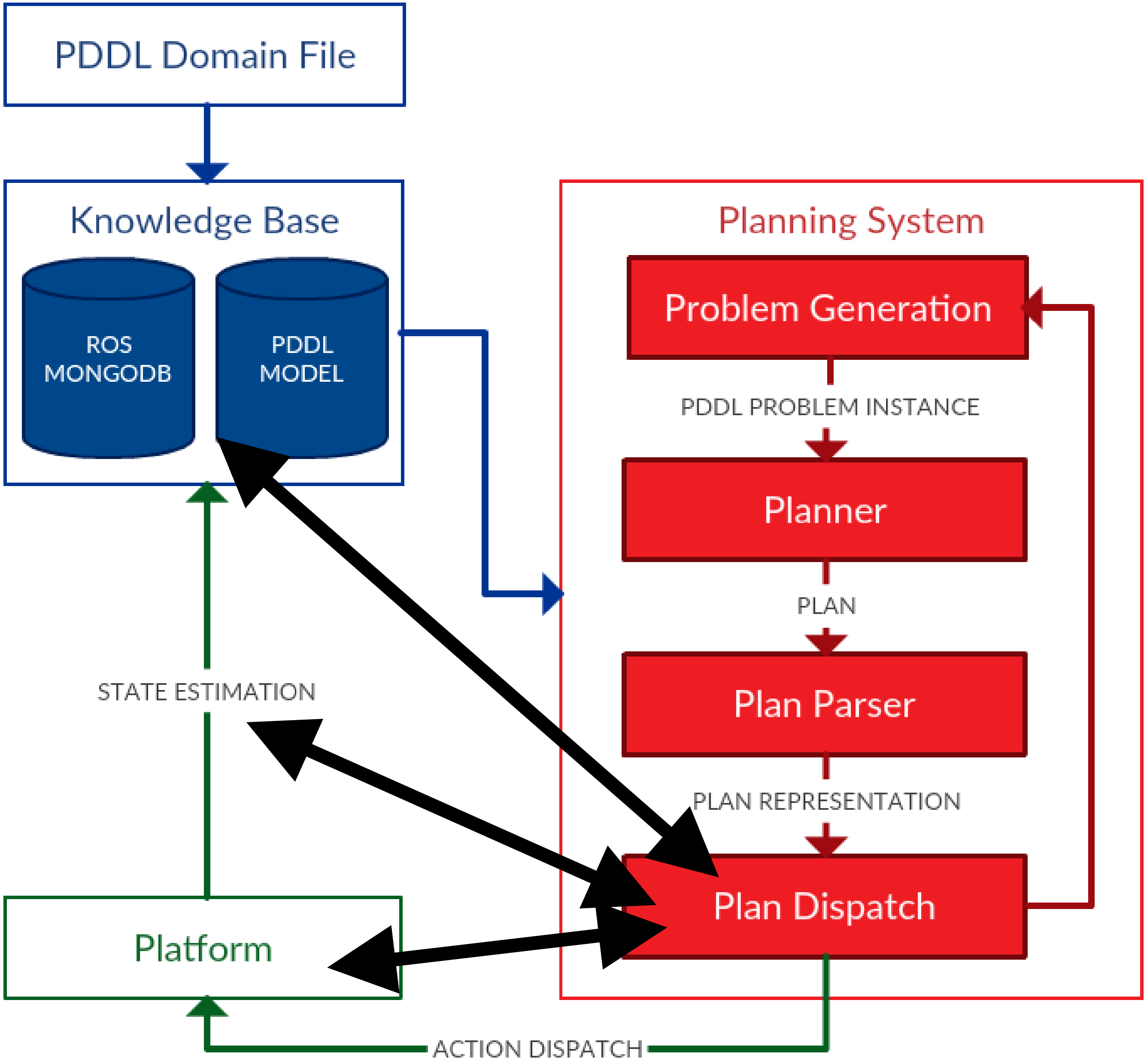
Summary of Very Simple Plan Execution

Plan Execution depends upon many components in the system. Changing any one of which will change the robot behaviour, and change the criteria under which the plan will succeed or fail.



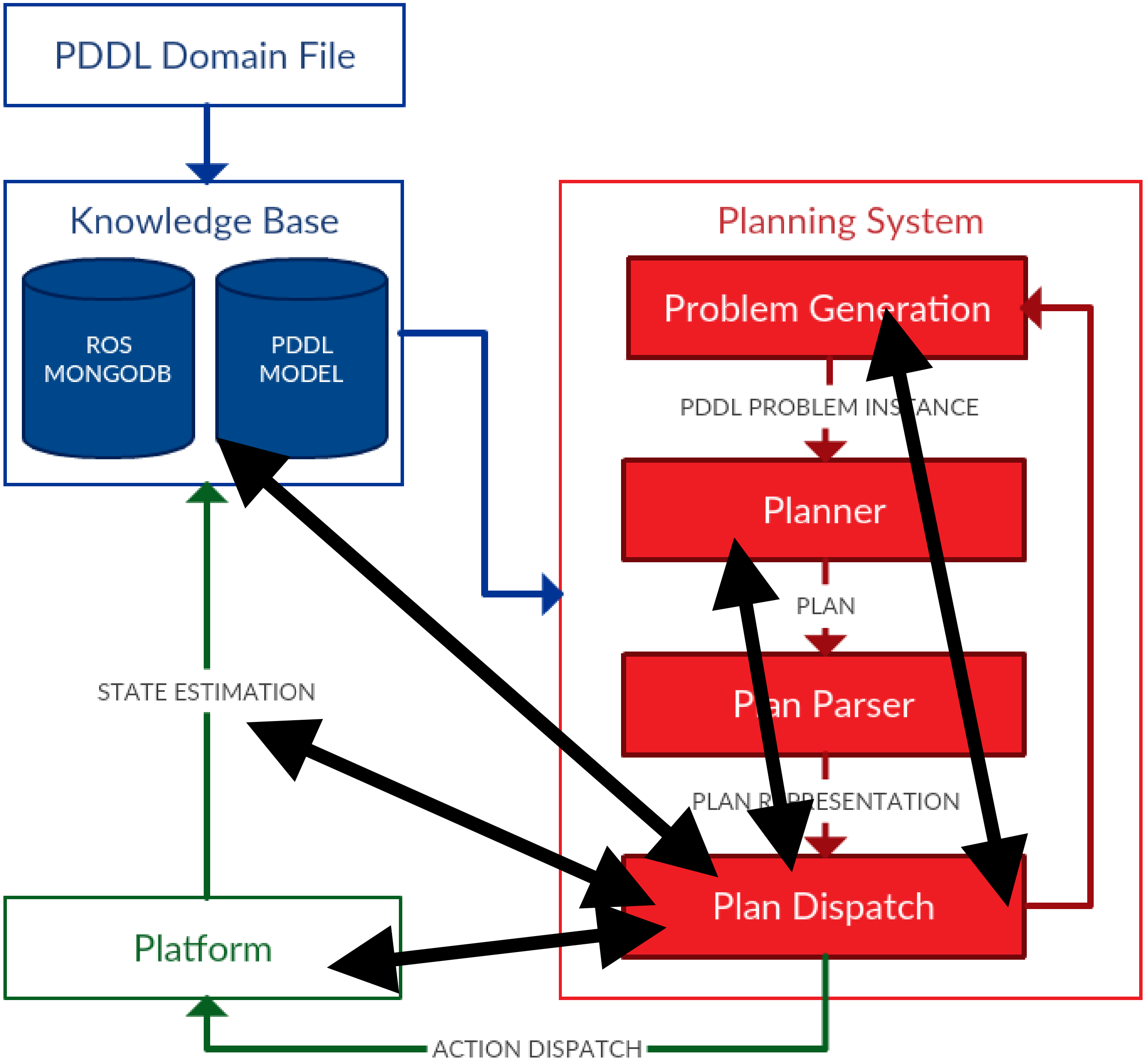
Summary of Very Simple Plan Execution

Plan Execution depends upon many components in the system. Changing any one of which will change the robot behaviour, and change the criteria under which the plan will succeed or fail.



Summary of Very Simple Plan Execution

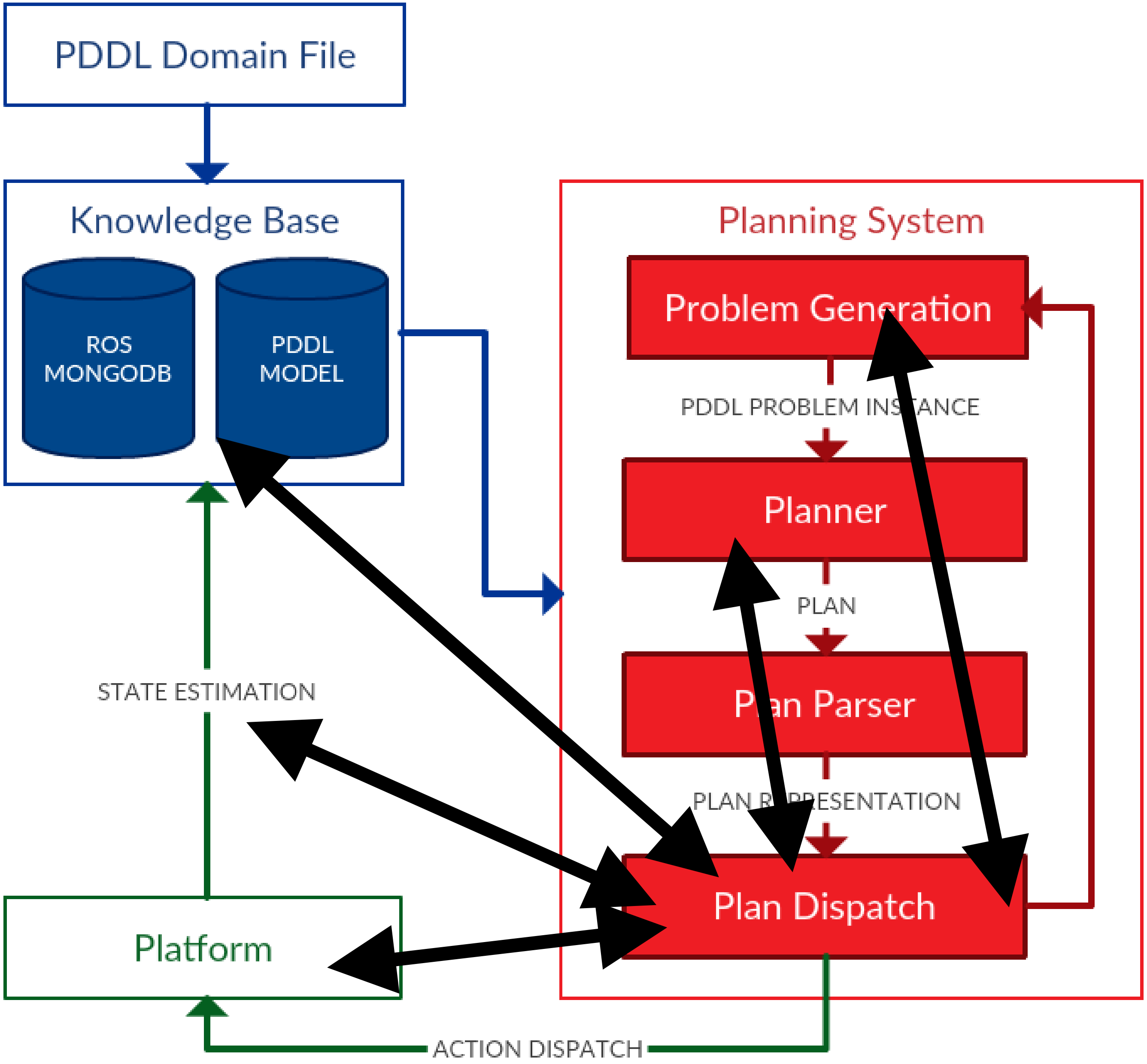
Plan Execution depends upon many components in the system. Changing any one of which will change the robot behaviour, and change the criteria under which the plan will succeed or fail.



Summary of Very Simple Plan Execution

Plan Execution depends upon many components in the system. Changing any one of which will change the robot behaviour, and change the criteria under which the plan will succeed or fail.

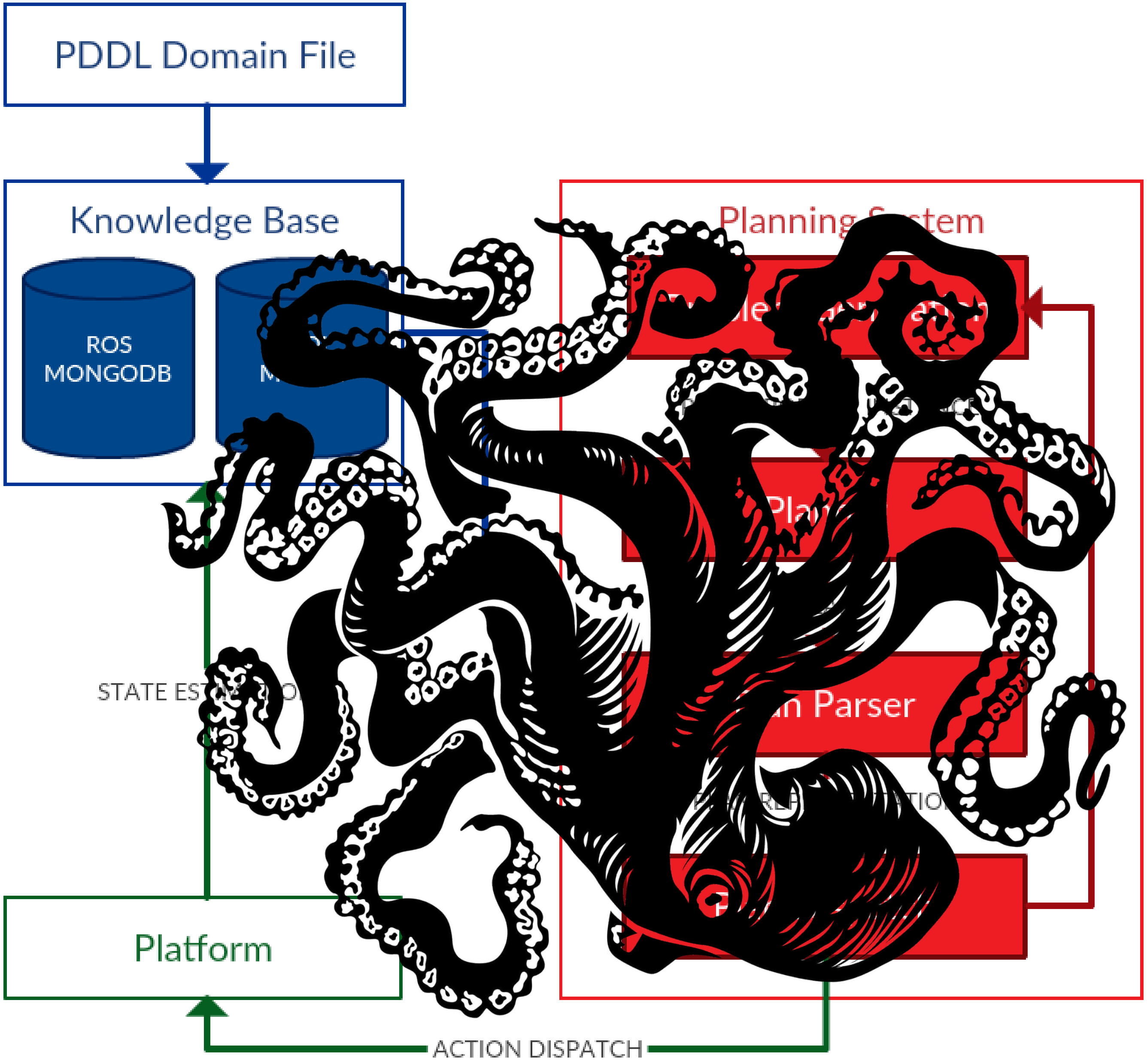
The execution of a plan is an emergent behaviour of the whole system.



Summary of Very Simple Plan Execution

Plan Execution depends upon many components in the system. Changing any one of which will change the robot behaviour, and change the criteria under which the plan will succeed or fail.

The execution of a plan is an emergent behaviour of the whole system.



Dispatching more than a Single Plan

The robot can have many different and interfering goals. A robot's behaviour might move toward achievement of multiple goals together.

Dispatching more than a Single Plan

The robot can have many different and interfering goals. A robot's behaviour might move toward achievement of multiple goals together.

The robot can also have:
- long-term goals (plans are abstract, with horizons of weeks)
- but also short-term goals (plans are detailed, with horizons of minutes)

Dispatching more than a Single Plan

The robot can have many different and interfering goals. A robot's behaviour might move toward achievement of multiple goals together.

The robot can also have:
- long-term goals (plans are abstract, with horizons of weeks)
- but also short-term goals (plans are detailed, with horizons of minutes)

The behaviour of a robot should not be restricted to only one plan.

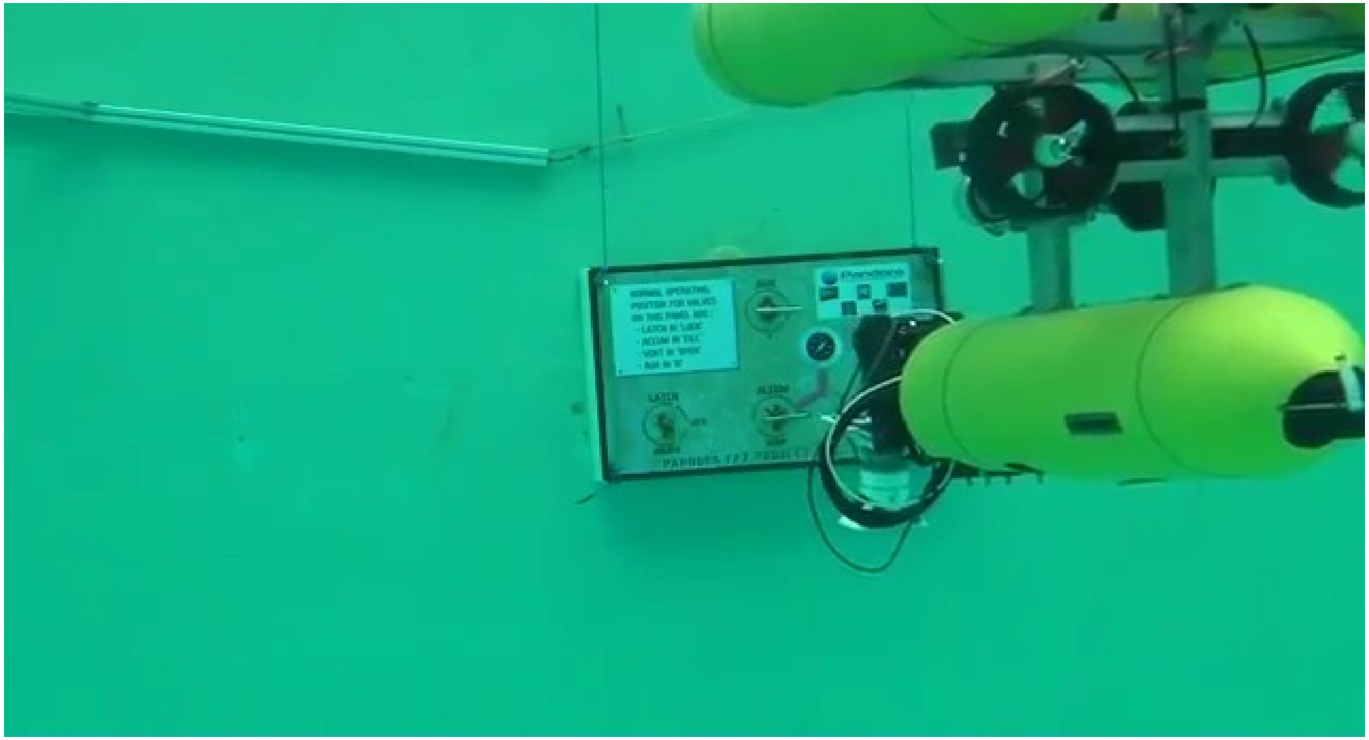
In a persistently autonomous system, the domain model, the planning process, and the plan are frequently revisited.

There is no “waterfall” sequence of boxes.

Dispatching more than a Single Plan

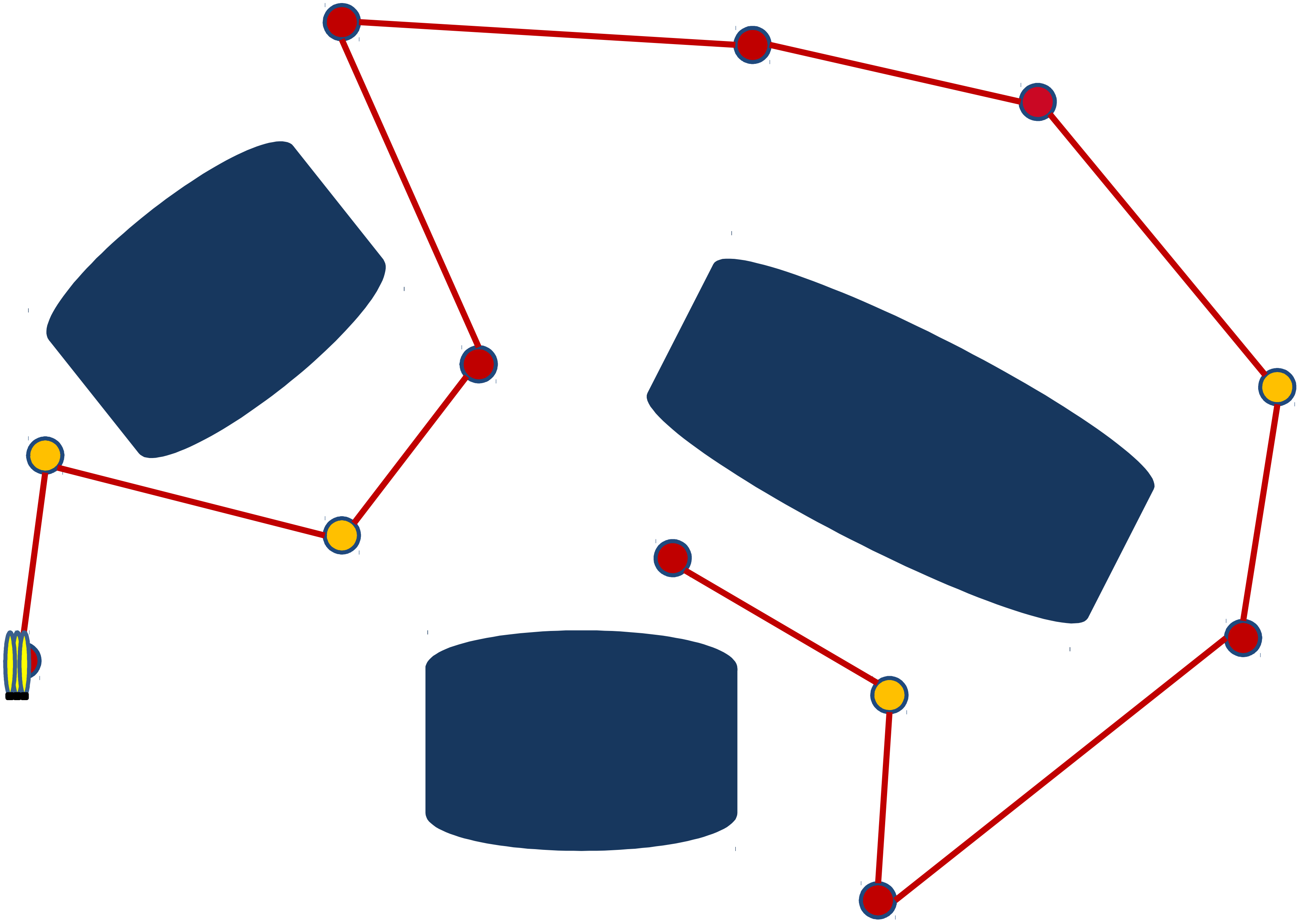
Example of multiple plans: What about unknowns in the environment?

One very common and simple scenario with robots is planning a search scenario. For tracking targets, tidying household objects, or interacting with people.

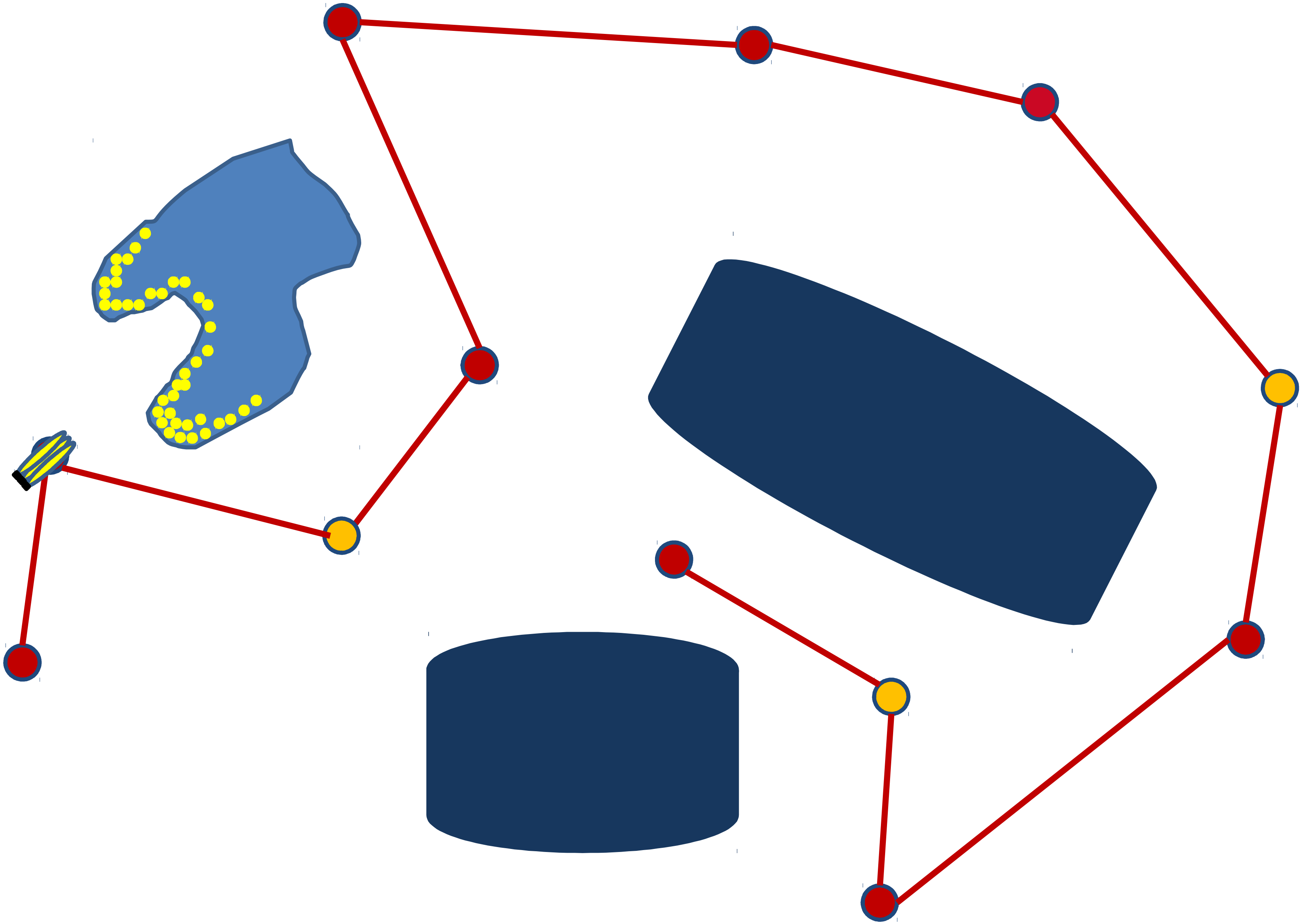


How do you plan from future situations that you can't predict?

Dispatching more than a Single Plan

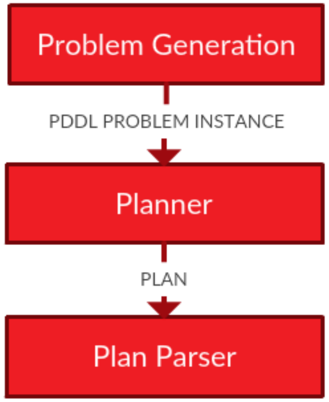


Dispatching more than a Single Plan



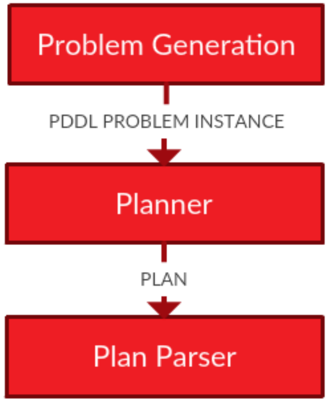
Hierarchical and Recursive Planning

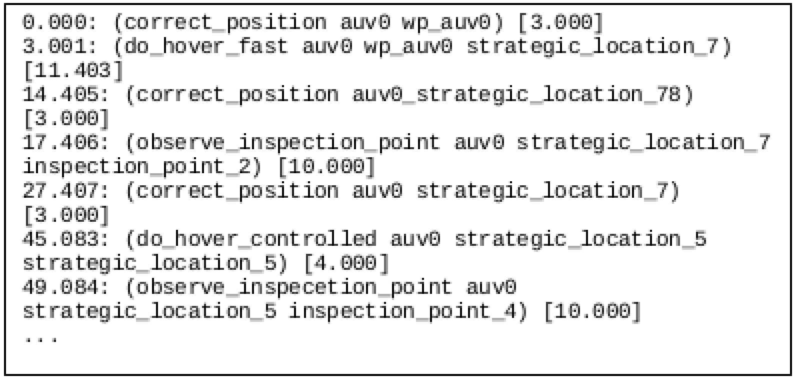
For each task we generate a *tactical plan*.



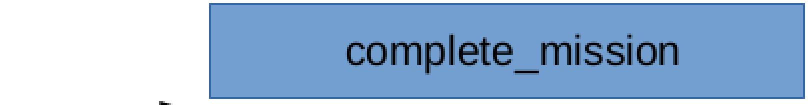
Hierarchical and Recursive Planning

For each task we generate a *tactical plan*. The time and resource constraints are used in the generation of the strategic problem.





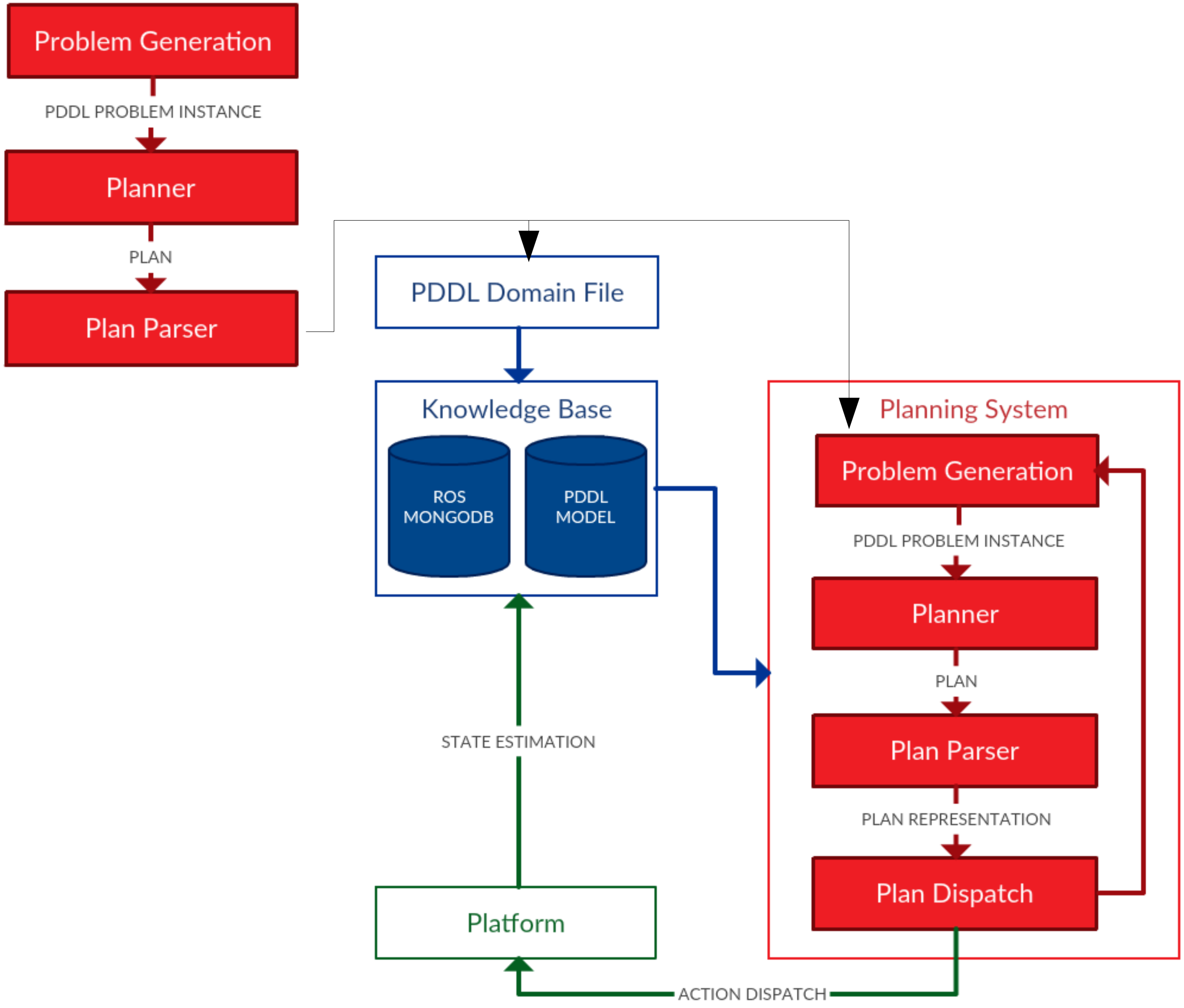
```
0.000: (correct_position auv0 wp_auv0) [3.000]
3.001: (do_hover_fast auv0 wp_auv0 strategic_location_7)
[11.403]
14.405: (correct_position auv0_strategic_location_78)
[3.000]
17.406: (observe_inspection_point auv0 strategic_location_7
inspection_point_2) [10.000]
27.407: (correct_position auv0 strategic_location_7)
[3.000]
45.083: (do_hover_controlled auv0 strategic_location_5
strategic_location_5) [4.000]
49.084: (observe_inspecetion_point auv0
strategic_location_5 inspection_point_4) [10.000]
...
```



Energy consumption = 10W
Duration = 86.43s

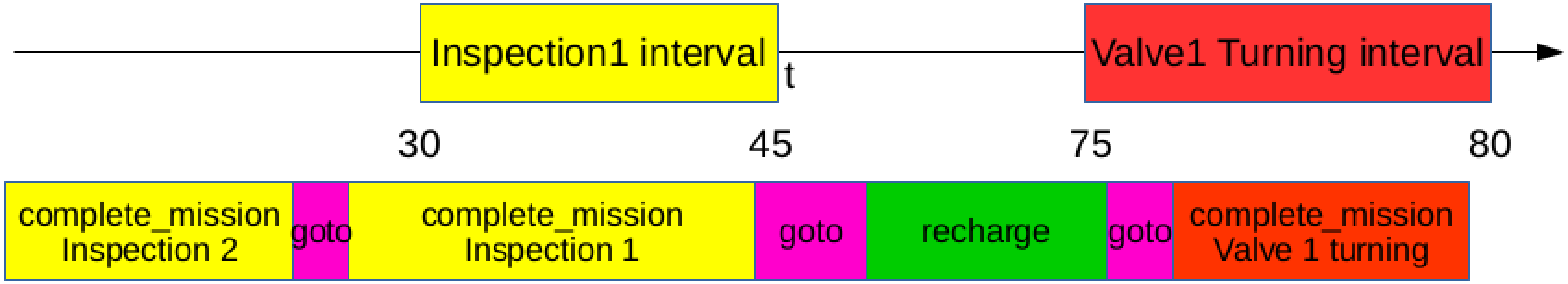
Hierarchical and Recursive Planning

For each task we generate a *tactical plan*. The time and resource constraints are used in the generation of the strategic problem.



Hierarchical and Recursive Planning

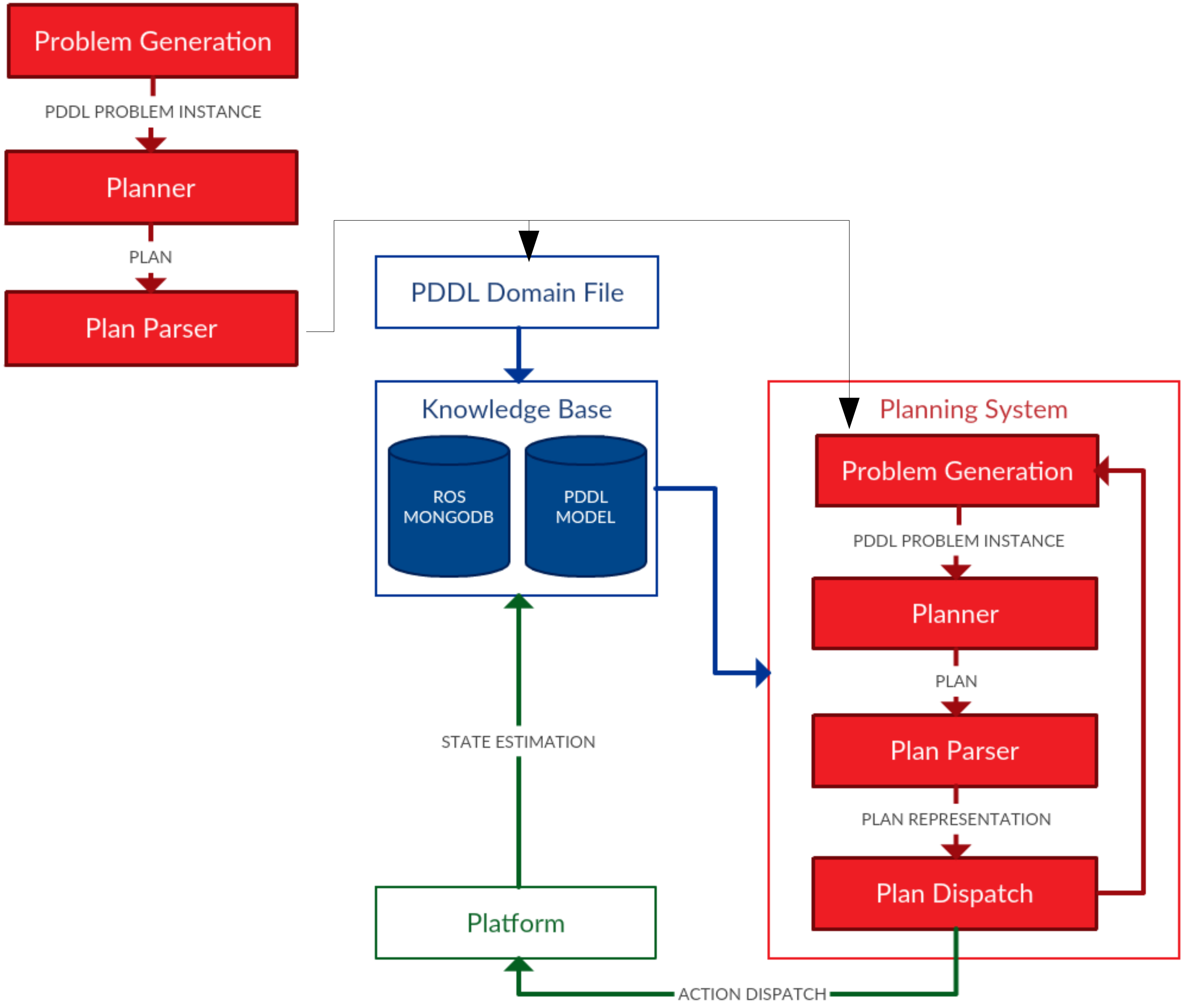
For each task we generate a *tactical plan*. The time and resource constraints are used in the generation of the strategic problem.


Inspection1 interval
t
Valve1 Turning interval
30
45
75
80
complete_mission Inspection 2
goto
complete_mission Inspection 1
goto
recharge
goto
complete_mission Valve 1 turning

A strategic plan is generated that does not violate the time and resource constraints of the whole mission.

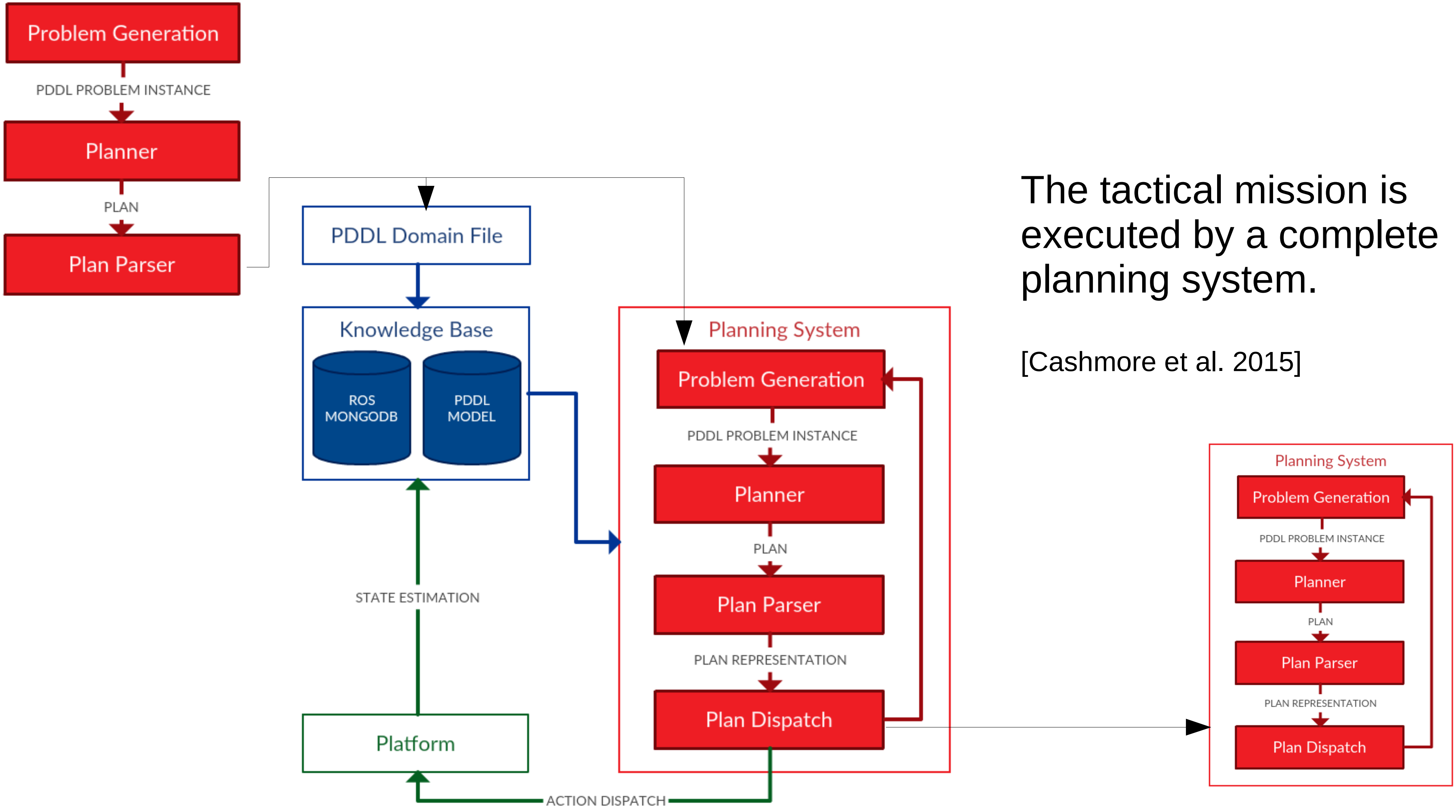
Hierarchical and Recursive Planning

When an abstract “complete_mission” action is dispatched, the tactical problem is regenerated, replanned, and executed.



Hierarchical and Recursive Planning

When an abstract “complete_mission” action is dispatched, the tactical problem is regenerated, replanned, and executed.



The tactical mission is executed by a complete planning system.

[Cashmore et al. 2015]

Dispatching more Plans: Opportunistic Planning

There might also be unknowns that we don't expect to discover.

For example, new opportunities are found during execution, and the robot should exploit them.

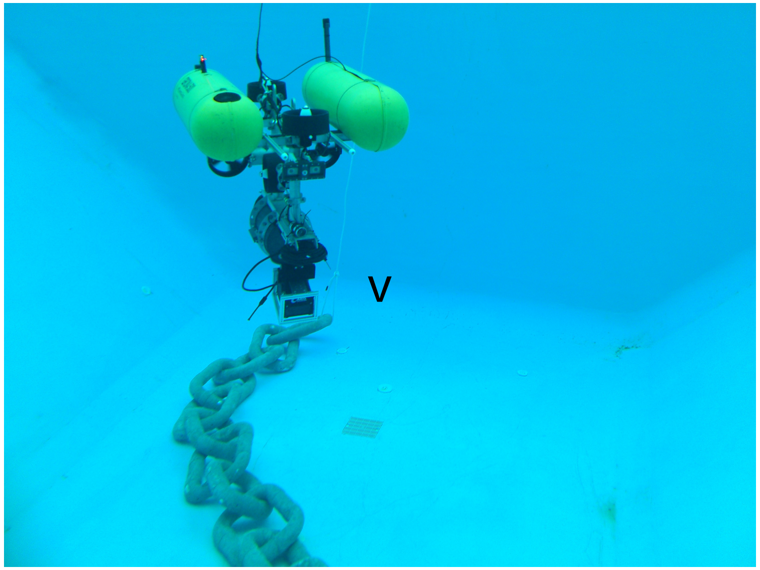
Dispatching more Plans: Opportunistic Planning

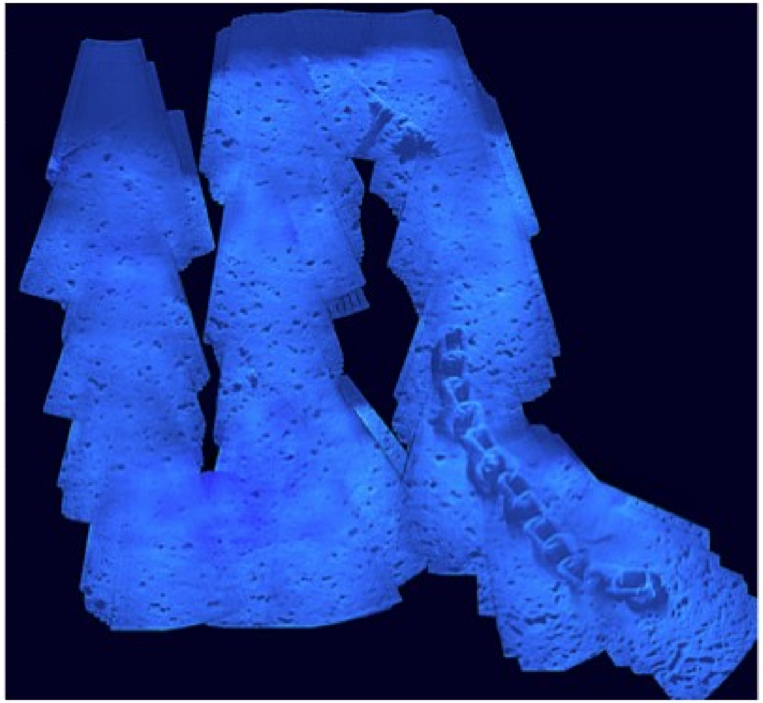
High Impact Low-Probability Events (HILPs)

- the probability distribution is unknown
- cannot be anticipated
- **our example is chain following**

If you see an unexpected chain, it's a good idea to investigate...

2011 Banff	5 of 10 lines parted.
2011 Volve	2 of 9 lines parted
2011 Gryphon Alpha	4 of 10 lines parted, vessel drifted a distance, riser broken
2010 Jubarte	3 lines parted between 2008 and 2010.
2009 Nan Hai Fa Xian	4 of 8 lines parted; vessel drifted a distance, riser broken
2009 Hai Yang Shi You	Entire yoke mooring column collapsed; vessel adrift, riser broken.
2006 Liuhua (N.H.S.L.)	7 of 10 lines parted; vessel drifted a distance, riser broken.
2002 Girassol buoy	3 (+2) of 9 lines parted, no damage to offloading lines (2 later)



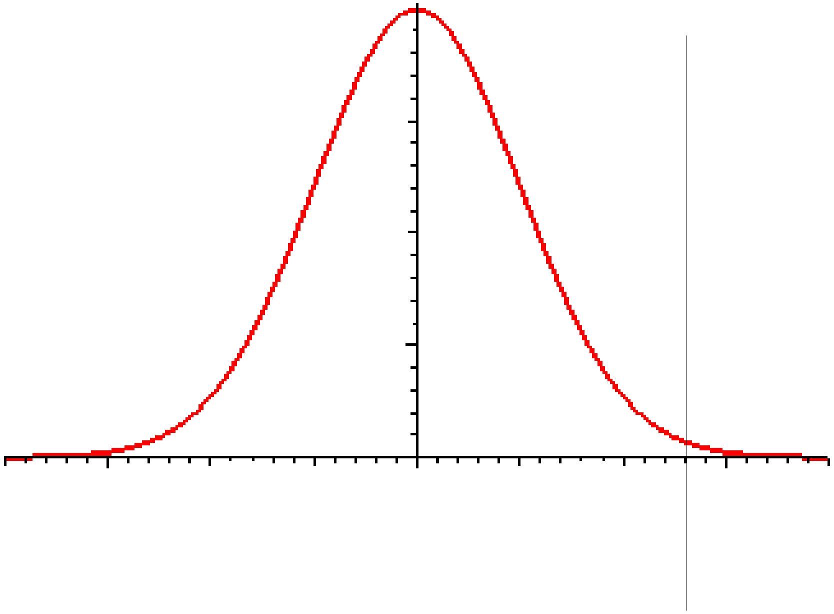


Dispatching more Plans: Opportunistic Planning

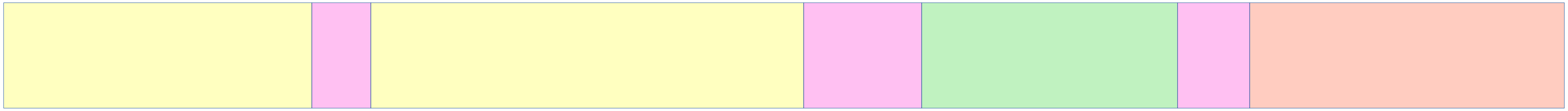
In PANDORA we planned and executed missions over long-term horizons (days or weeks)

Our planning strategy was based on the assumption that actions have durations normally distributed around the mean.

To build a robust plan we therefore used estimated durations for the actions that were 95th percentile of the normal distribution.



The resulting overestimation of actions builds a **free time window**

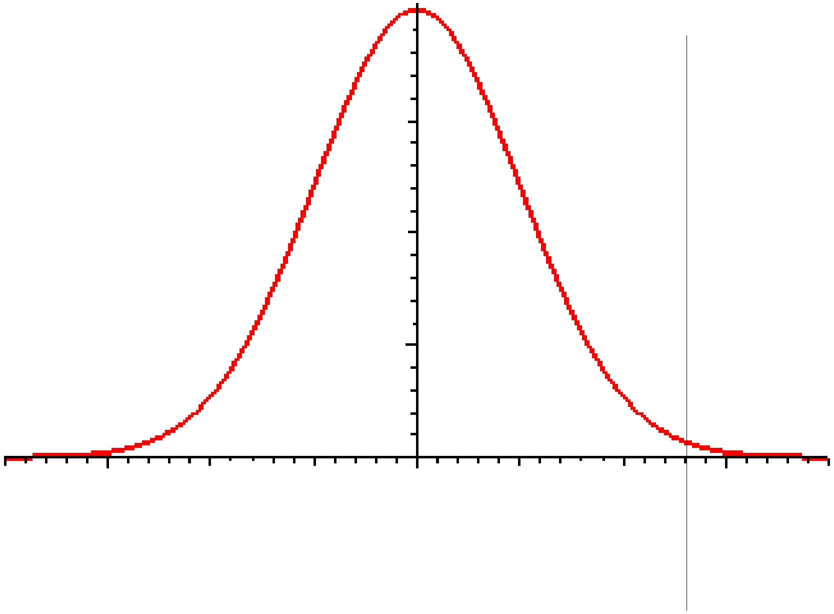


Dispatching more Plans: Opportunistic Planning

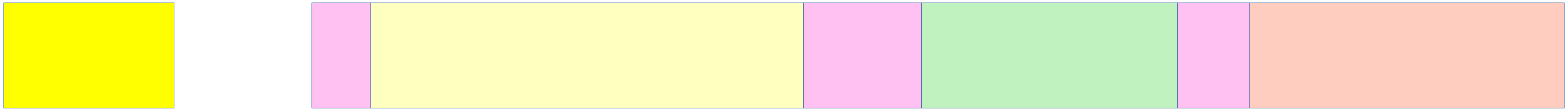
In PANDORA we planned and executed missions over long-term horizons (days or weeks)

Our planning strategy was based on the assumption that actions have durations normally distributed around the mean.

To build a robust plan we therefore used estimated durations for the actions that were 95th percentile of the normal distribution.



The resulting overestimation of actions builds a **free time window**

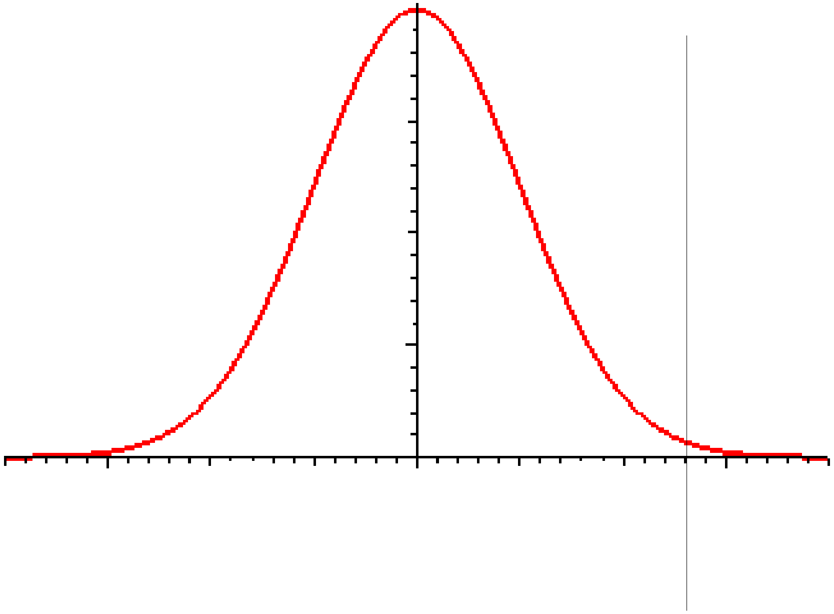


Dispatching more Plans: Opportunistic Planning

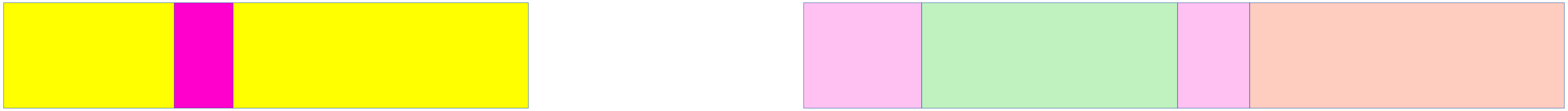
In PANDORA we planned and executed missions over long-term horizons (days or weeks)

Our planning strategy was based on the assumption that actions have durations normally distributed around the mean.

To build a robust plan we therefore used estimated durations for the actions that were 95th percentile of the normal distribution.



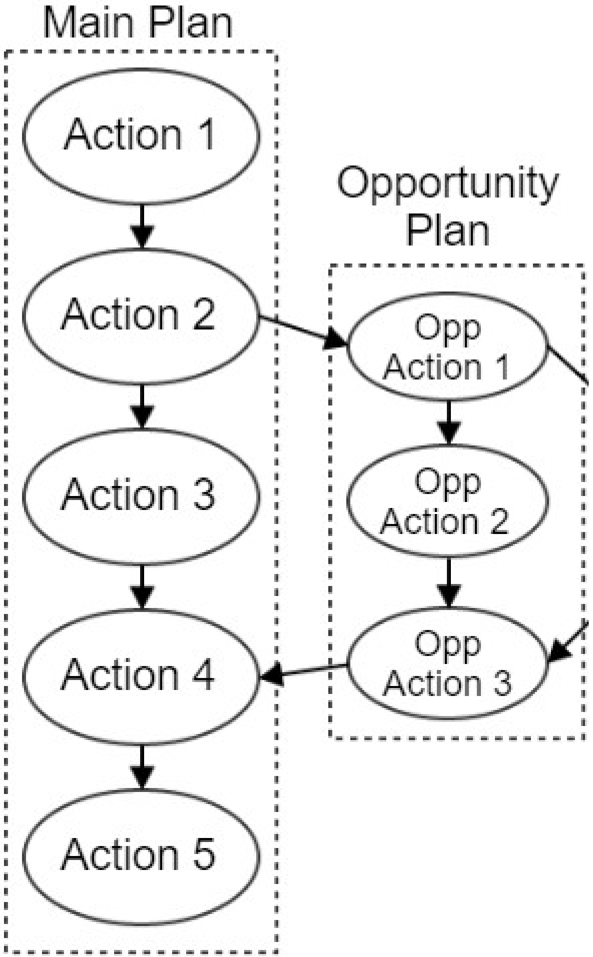
The resulting overestimation of actions builds a **free time window**



Dispatching more Plans: Opportunistic Planning

New plans are generated for the opportunistic goals and the goal of returning to the tail of the current plan.

If the new plan fits inside the free time window, then it is immediately executed.



Dispatching more Plans: Opportunistic Planning

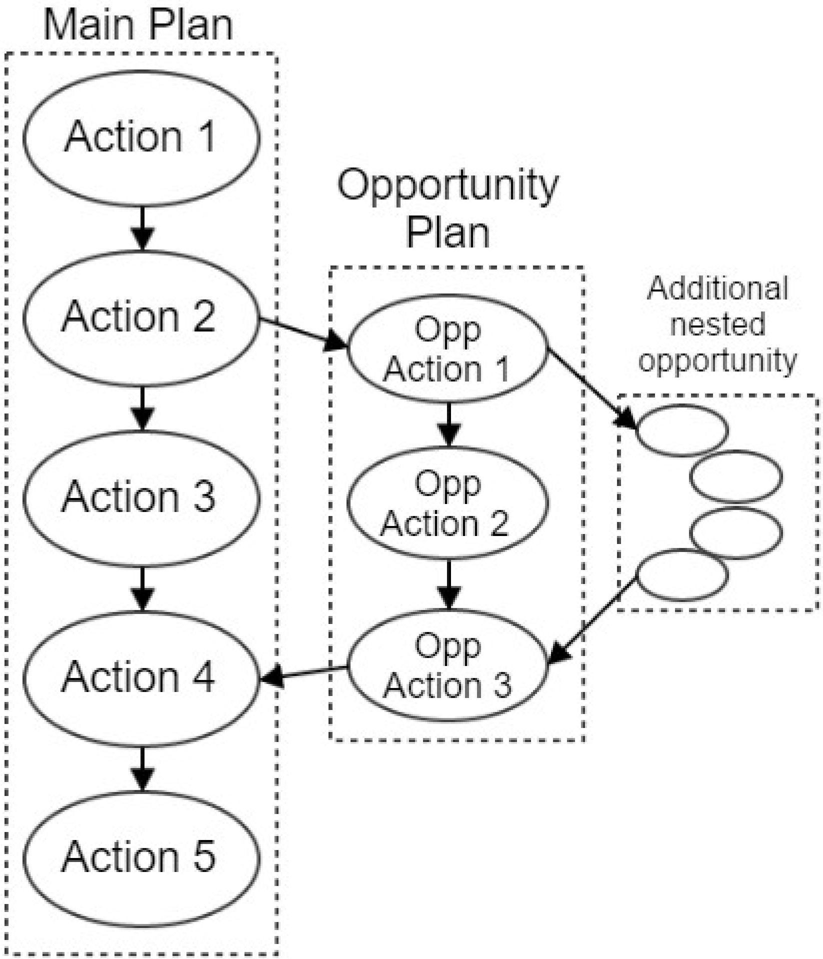
New plans are generated for the opportunistic goals and the goal of returning to the tail of the current plan.

If the new plan fits inside the free time window, then it is immediately executed.

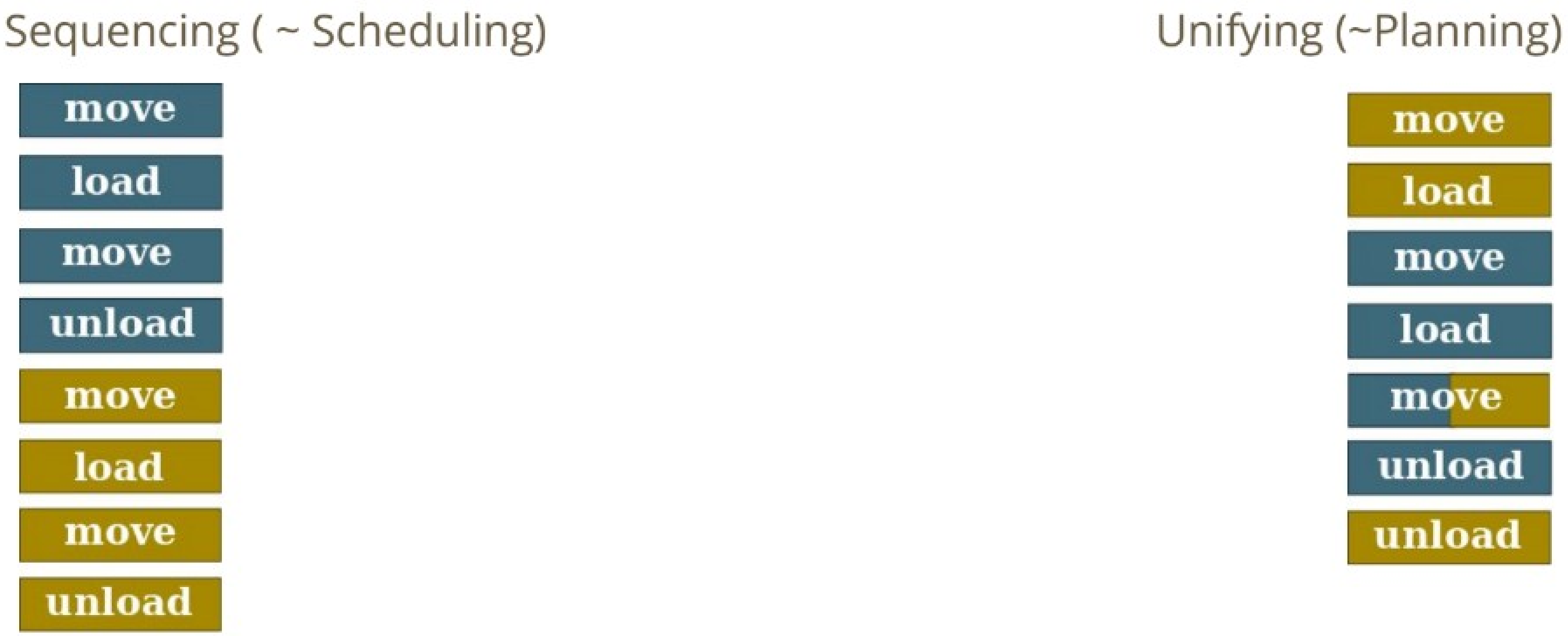
The approach is recursive

If an opportunity is spotted during the execution of a plan fragment, then the currently executing plan can be pushed onto the stack and a new plan can be executed.

[Cashmore et al. 2015]



Dispatching Plans at the same time

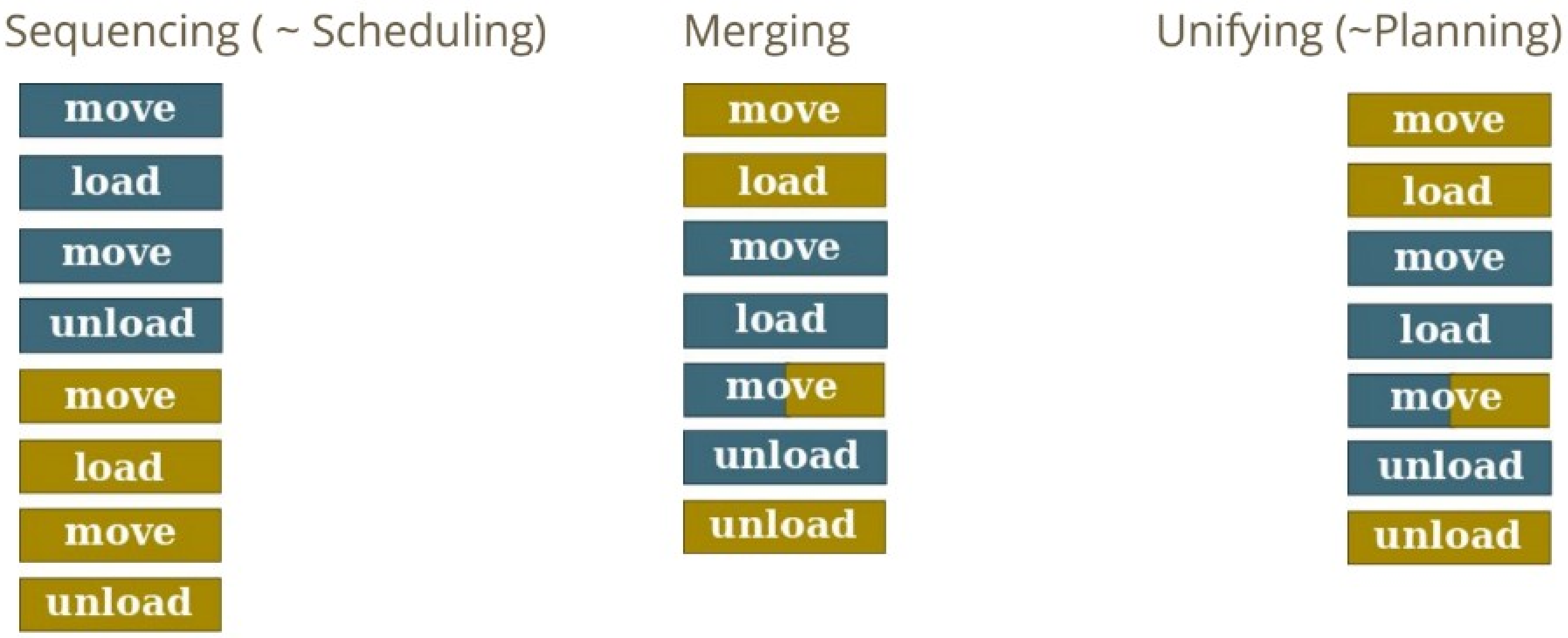


Sequencing (~ Scheduling)

Unifying (~Planning)

Separating tasks and scheduling is not as efficient.
Planning for everything together is not always practical.

Dispatching Plans at the same time



Separating tasks and scheduling is not as efficient.
Planning for everything together is not always practical.

Plans can be merged in a more intelligent way. A single action can support the advancement towards multiple goals. [Mudrova et al. 2016]

Questions?

What is the glue in a Plan Execution framework that is *always* required?

How do we modify a domain model during execution?

Which parts of a domain model are transferable to other tasks?

Which parts of a domain model can be generated automatically

- From a description of the robot?
- From a source ontology?

How can we get rid of the planning expert?

- Can a description of a task be written by a non-expert, and a generic domain extended?