AI Planning for Robotics and Human-Robot Interaction

Michael Cashmore — King's College London

Luca Iocchi — Sapienza University of Rome

Daniele Magazzeni — King's College London

ICAPS 2017

19 June 2017

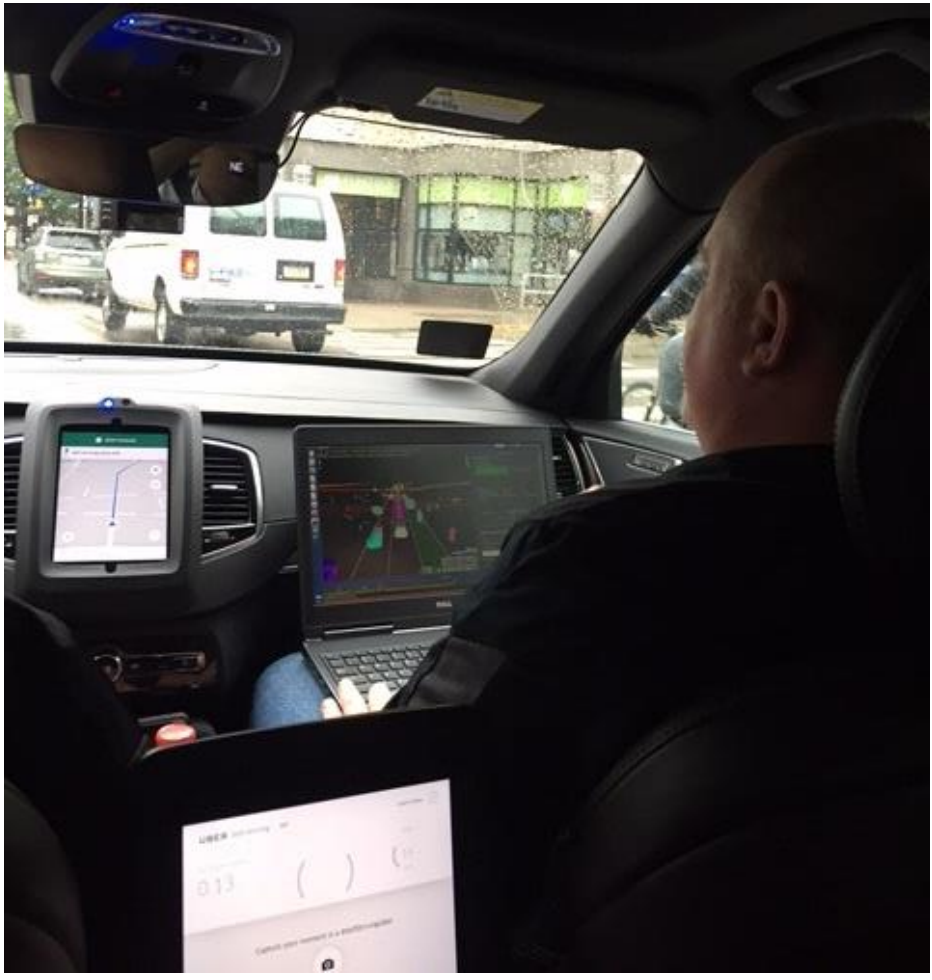
Pittsburgh –USA

Why Human-Robot Interaction is important…

Coming here this morning….

2 people for driving a car

AI is CREATING jobs!



Disclaimer 1

Planning and Robotics is a growing area!

ICAPS workshops PlanRob
ICAPS Special Track on Planning and Robotics
PlanRob workshop + tutorial at ICRA 2017
Dagstuhl workshop on Planning and Robotics

This tutorial covers only some aspects

PlanRob workshop tomorrow (full day)

Disclaimer 2

One can use several formalisms to model robotics domains.
And one can use several techniques for planning in these domains.

Having said that, this tutorial will focus on
Domain-Independent Planning through PDDLx

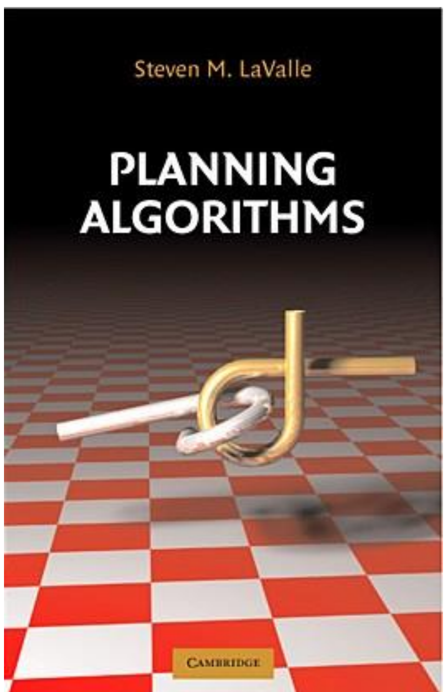
Disclaimer 3

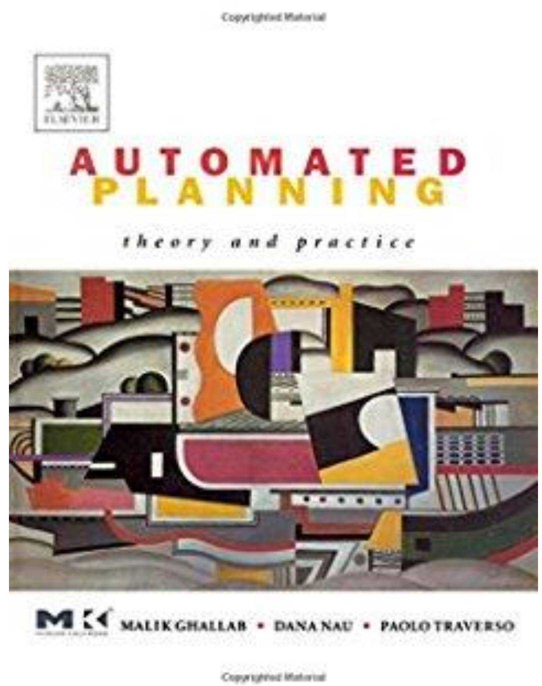
Planning is actually *plural*

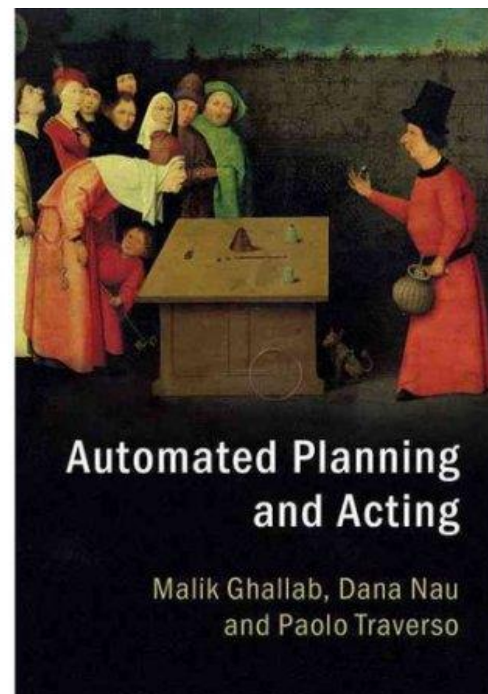
Thanks to Malik Ghallab!

planning includes many things

in this tutorial: “planning”=“task planning”







Disclaimer 4

This is a tutorial
and we agreed to make it an *accessible* one

Slides + Virtual Machine + Demo
available in the ROSPlan website

Outline

- Why PDDL Planning for Robotics and HRI?
- ROSPlan I: Planning with ROS

Coffee (10.30-11.00)

- ROSPlan II: Planning with Opportunities
- Petri Net Plan Execution
- Open challenges

Outline

- Why **PDDL** Planning for Robotics and HRI?

Where PDDL planning is NOT useful for Robotics?

- **Single/Repetitive Tasks (no PDDL for manipulation/grasping!)**
- **Safe Navigation (Sampling is much better!)**

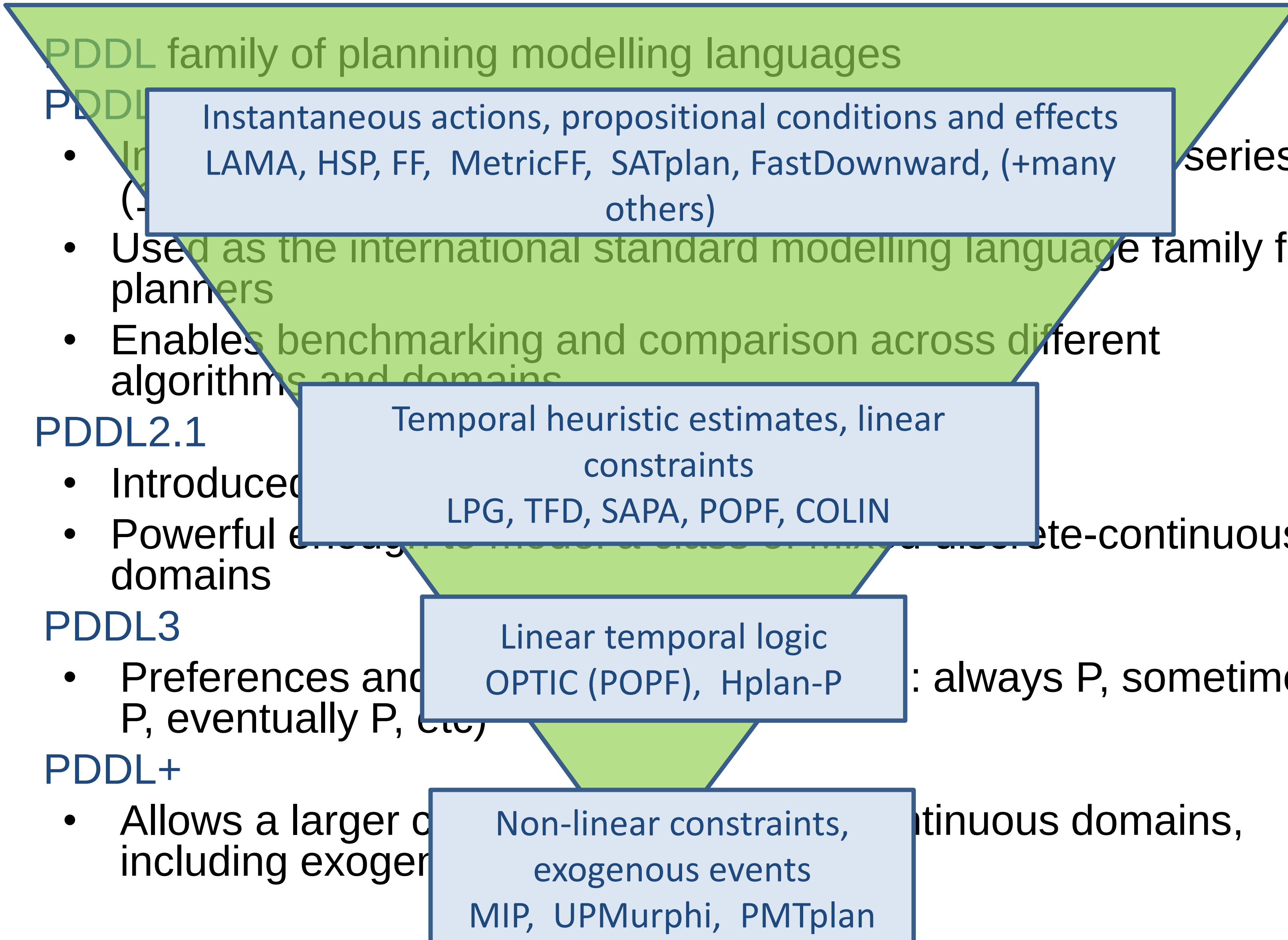
- **PDDL planning is really useful when there is room for optimisation at a task level**

Outline

- **Why PDDL Planning for Robotics and HRI?**
  - **Expressive Planning**
  - **Opportunistic Planning**
  - **Strategic Planning**
  - **eXplainable Planning (XAIP)**
  - **Planning with Uncertainty**

Expressive Planning

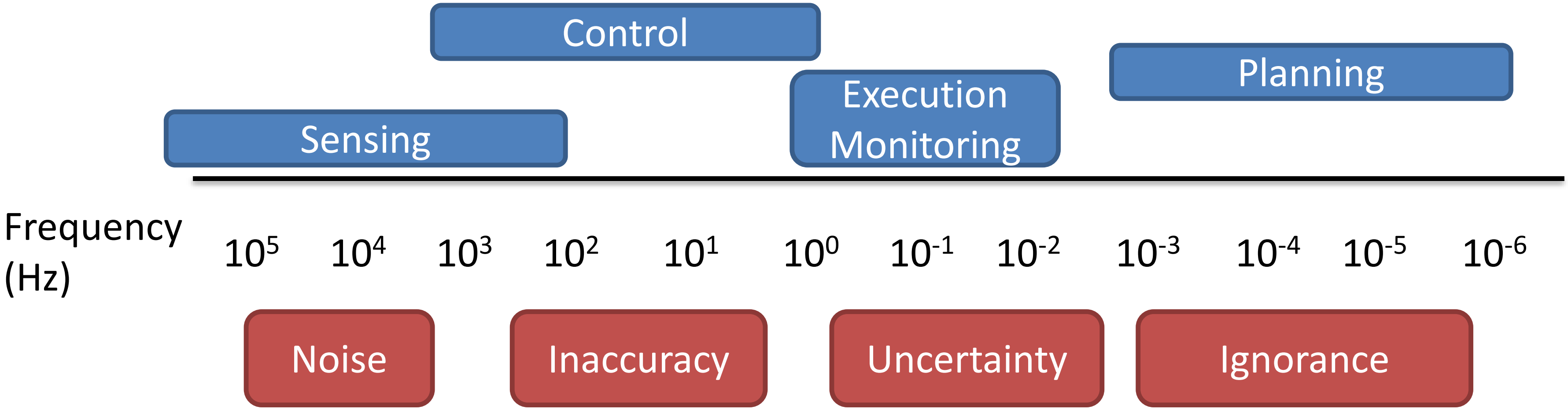
- PDDL family of planning modelling languages
- PDDL
  - In ... series (...
  - Used as the international standard modelling language family for planners
  - Enables benchmarking and comparison across different algorithms and domains
- PDDL2.1
  - Introduced ...
  - Powerful e... discrete-continuous domains
- PDDL3
  - Preferences and ... : always P, sometimes P, eventually P, etc)
- PDDL+
  - Allows a larger c... ntinuous domains, including exogen...



Planning and Control

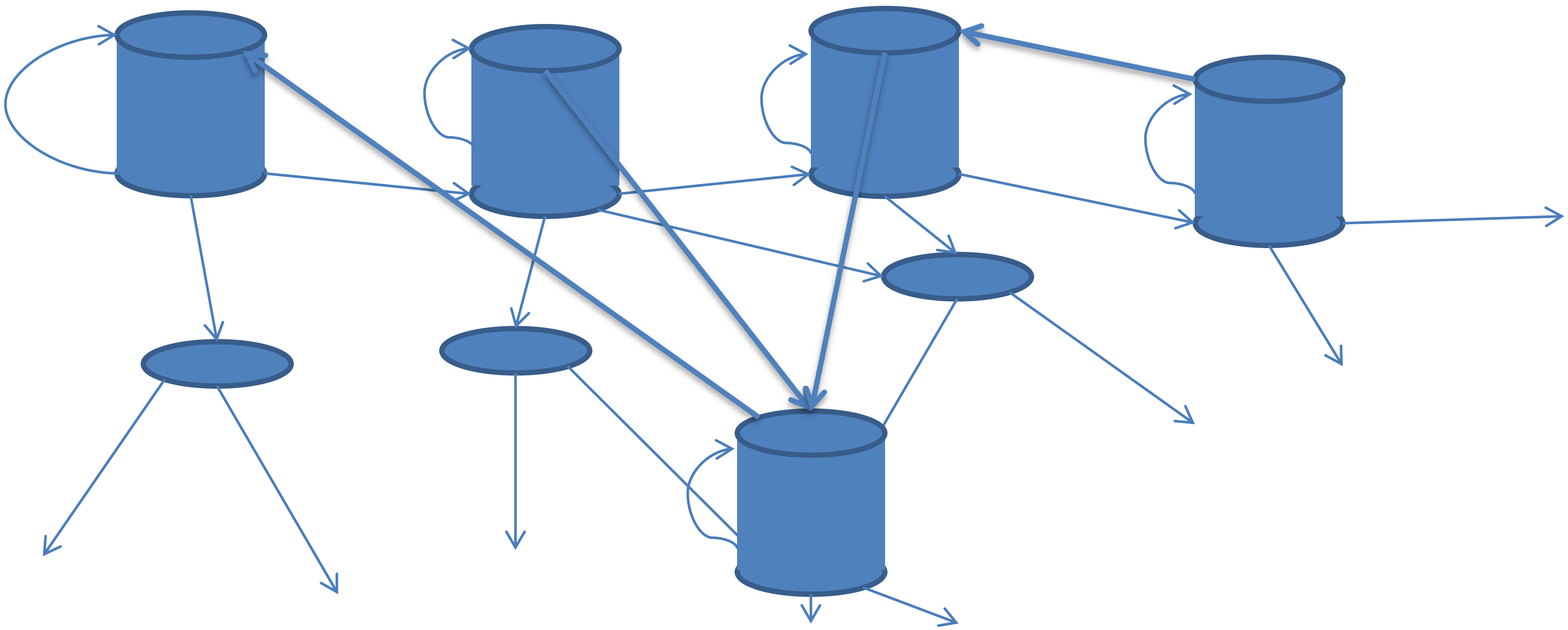
Planning is an AI technology that seeks to select and organise activities in order to achieve specific goals

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



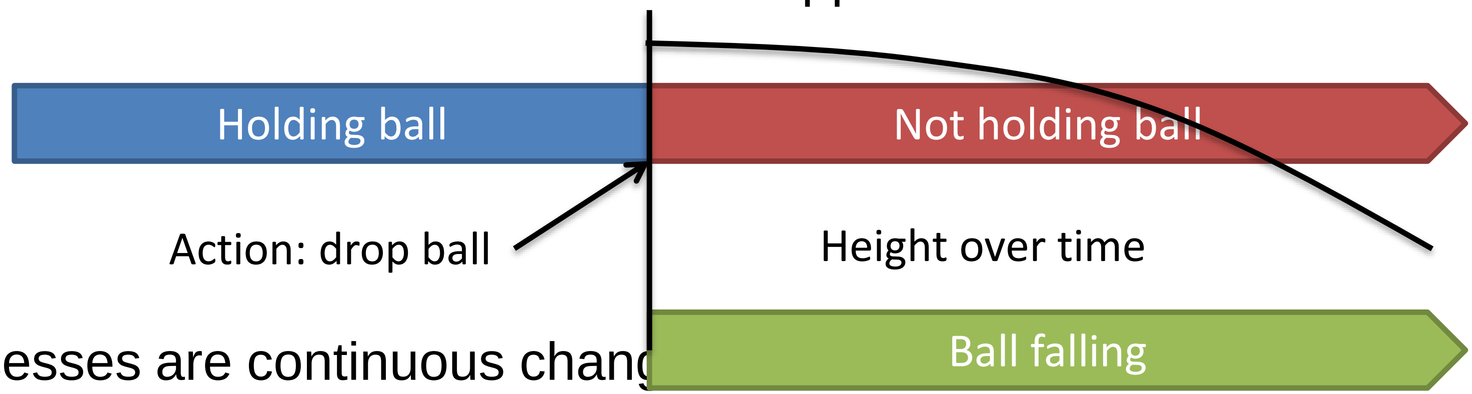
Planning with Time: An Additional Dimension

- 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

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!
$(\textbf{height b1})(t) = -5t^2 + 14.1421t + 2.22045e - 15$
$(\textbf{velocity b1})(t) = 10t - 14.1421$
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!
$(\textbf{height b1})(t) = -5t^2 + 14.1421t + 2.44943e - 15$

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

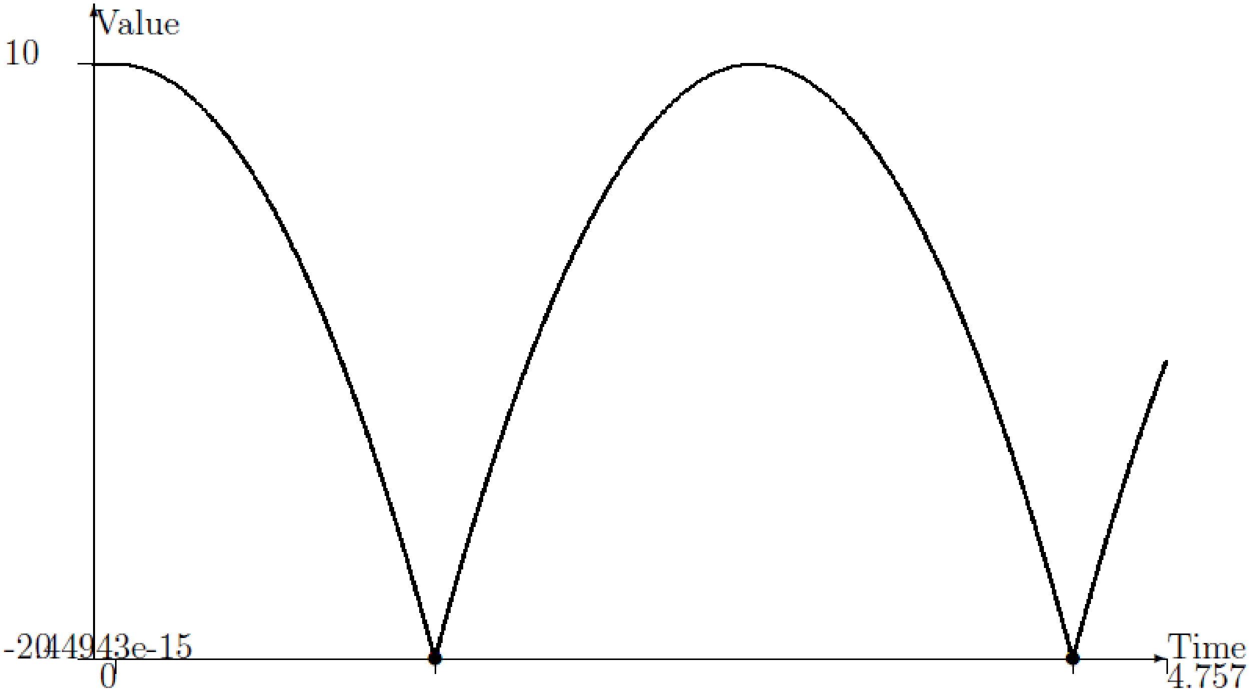


Figure 2.1: Graph of (height b1).

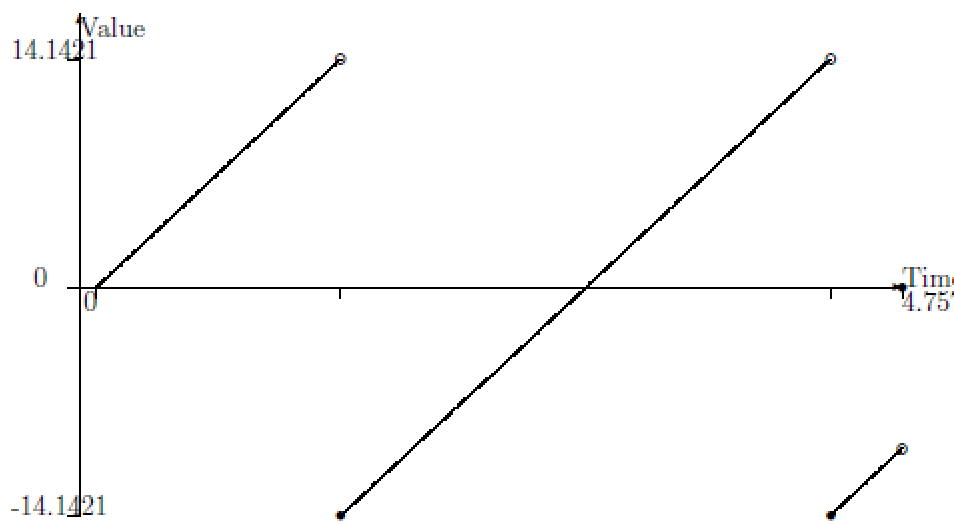


Figure 2.2: Graph of (velocity b1).

Some PDDL+ Planners

- **UPMurphi** (Della Penna et al.) [ICAPS'09]

  Based on Discretise and Validate

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

- **DiNo** (Piotrowski et al.) [IJCAI'16]

  Extend UPMurphi with TRPG heuristic for hybrid domains

- **SMTPlan** (Cashmore et al.) [ICAPS'16]

  Based on SMT encoding of PDDL+ domains

- **ENHSP** (Scala et al.) [IJCAI'16]

  Expressive numeric heuristic planning

- **dReach/dReal** (Bryce et al.) [ICAPS-15]

  Combine SMT encoding with dReal solver

- **POPF** (Coles et al.) [ICAPS-10]

  Combine Forward Search and Linear Programming

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))
)
```

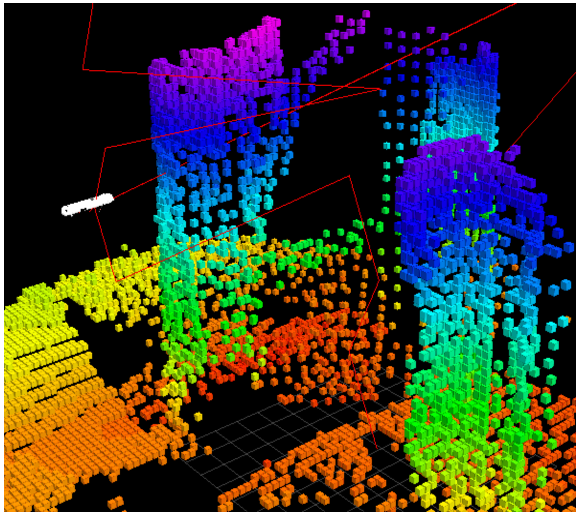
```
(:process wind
 :parameters ()
 :precondition (and (not (crashed)) (ascending) )
 :effect (and (increase (altitude) (* #t (wind_y) 1)
                    (increase (distance) (* #t (wind_x) 1)))
```

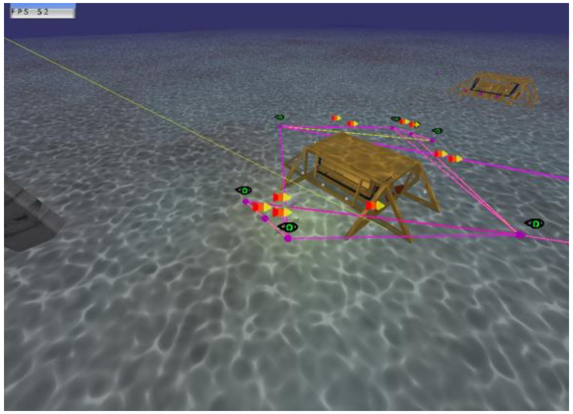
Outline

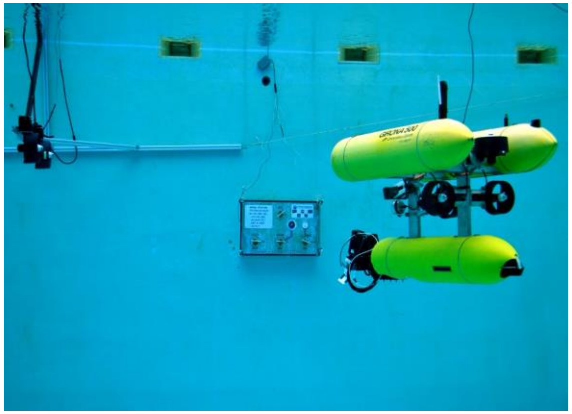
- Why PDDL Planning for Robotics and HRI?
  - Expressive Planning
  - Opportunistic Planning
  - Strategic Planning
  - eXplainable Planning (XAIP)
  - Planning with Uncertainty

Opportunistic Planning

- Very important in persistent autonomy
- Use case: PANDORA (EU funded project)



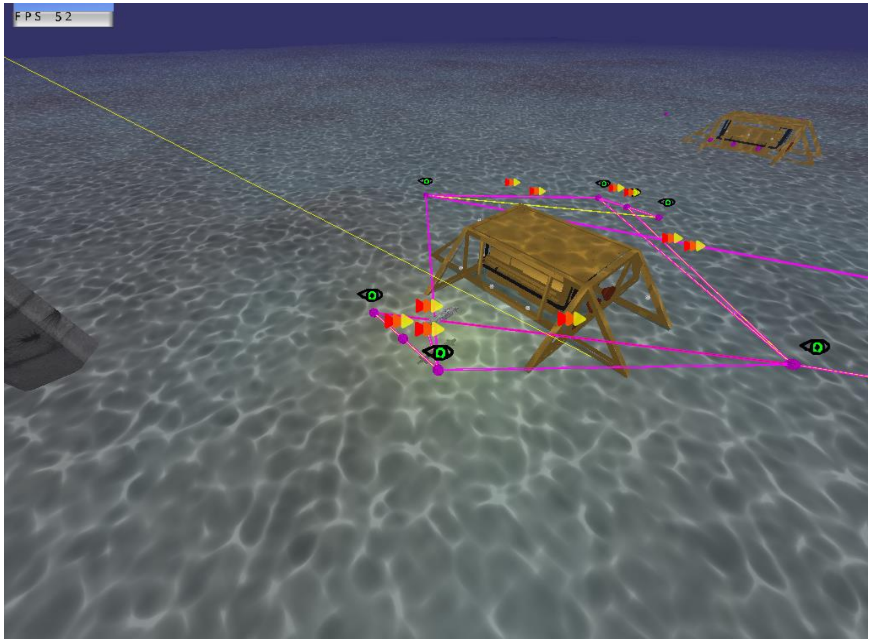




Persistent Autonomy (AUVs)

Inspection and maintenance of a seabed facility:

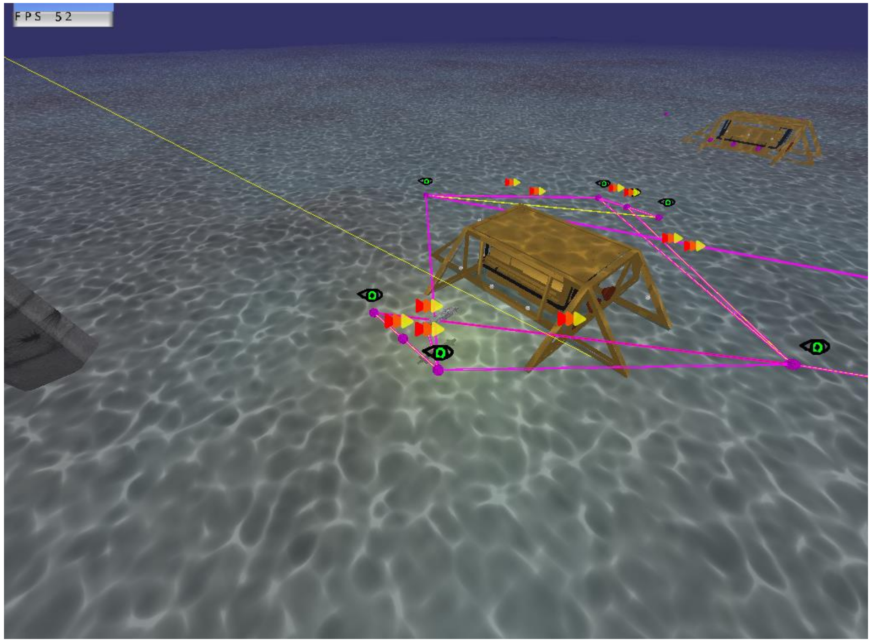
- without human intervention
- inspecting manifolds
- cleaning manifolds
- manipulation valves
- **opportunistic tasks**



Persistent Autonomy (AUVs)

Inspection and maintenance of a seabed facility:

- without human intervention
- inspecting manifolds
- cleaning manifolds
- manipulation valves
- **opportunistic tasks**



AUV mission, many tasks at scattered locations.

- long horizon plans
- large amount of uncertainty
- **discovery**

High utility, low-probability opportunities for new tasks.

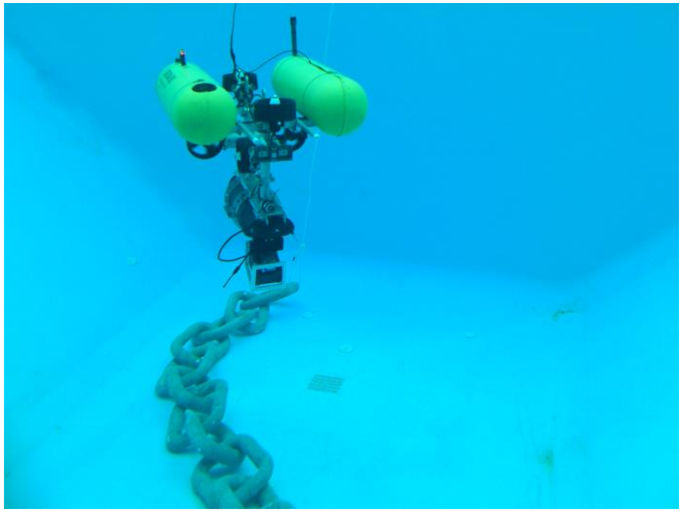
Persistent Autonomy (AUVs)

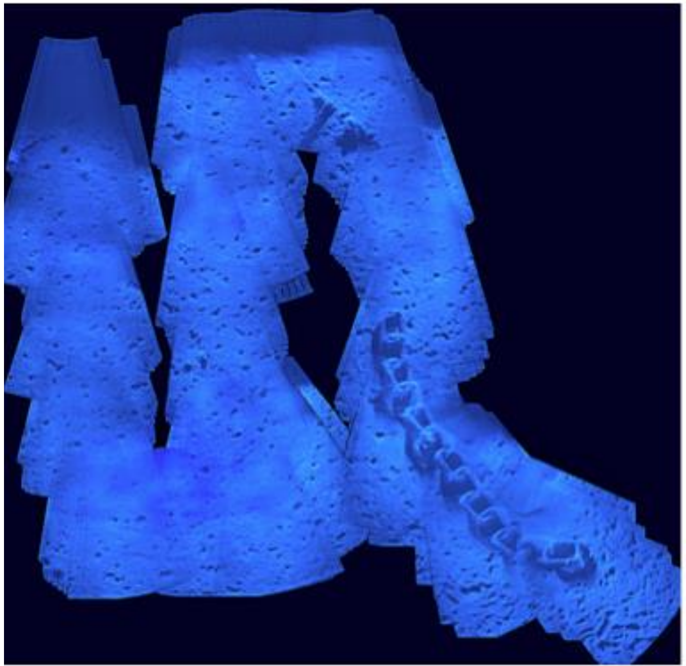
High Impact Low-Probability Events (HILPs)

- 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...

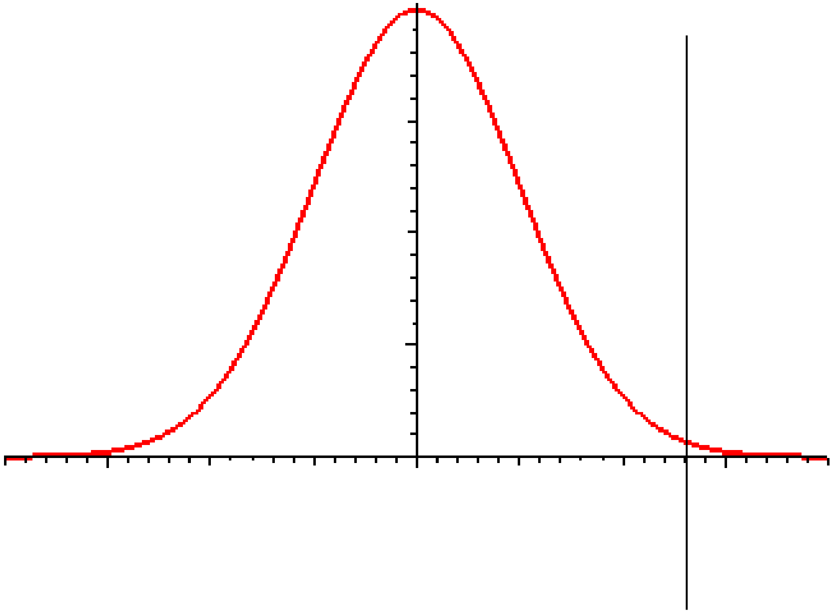
2011 Banff	5 of 10 lines parted.
2011 Volve	2 of 9 lines parted
2011 Gryphon Alpha	4 of 10 lines parted, vessel drifted a distance, riser broken
2010 Jubarte	3 lines parted between 2008 and 2010.
2009 Nan Hai Fa Xian	4 of 8 lines parted; vessel drifted a distance, riser broken
2009 Hai Yang Shi You	Entire 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)





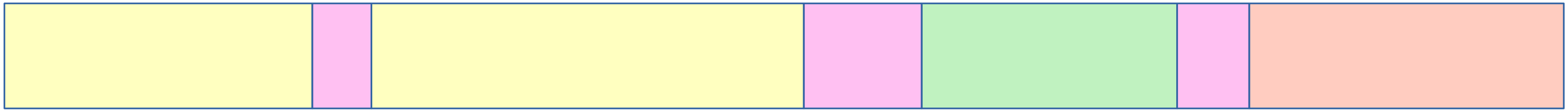
Opportunistic Planning

In PANDORA we plan and execute missions over long-term horizons (days or weeks)



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

To build a robust plan we therefore use estimated durations for the actions that are longer than the mean.
(95th percentile of the normal distribution)

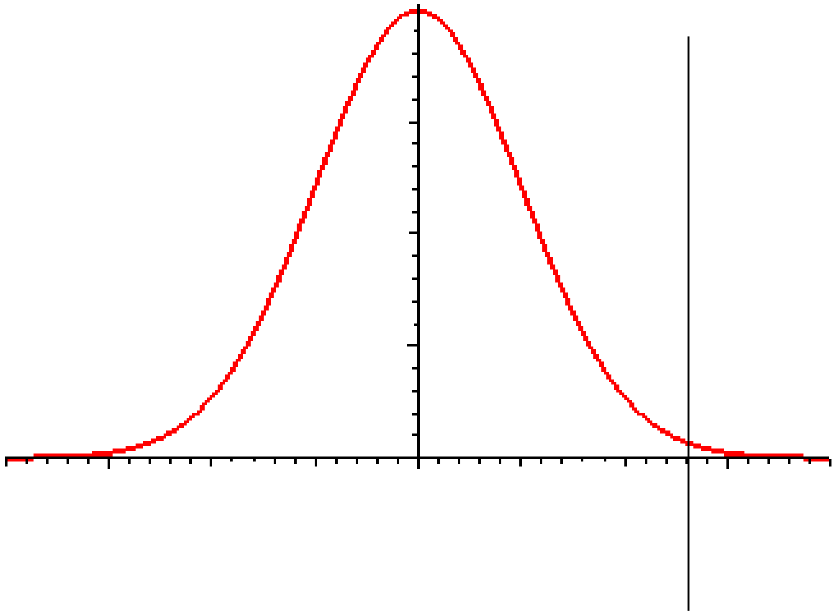


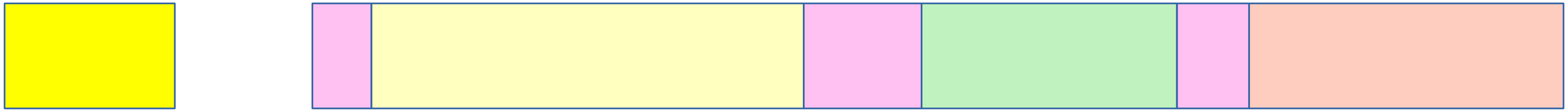
Opportunistic Planning

In PANDORA we plan and execute missions over long-term horizons (days or weeks)

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

To build a robust plan we therefore use estimated durations for the actions that are longer than the mean.
(95th percentile of the normal distribution)



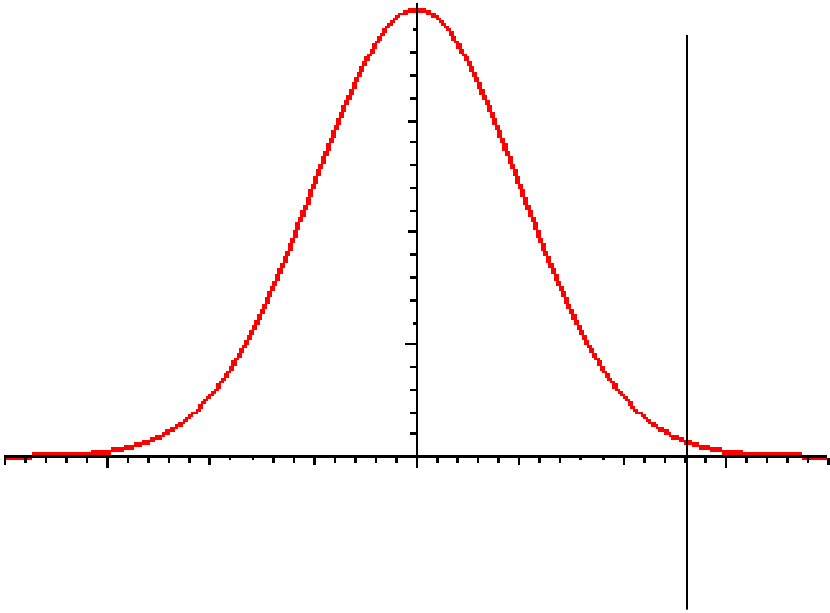


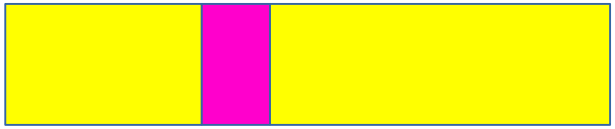
Opportunistic Planning

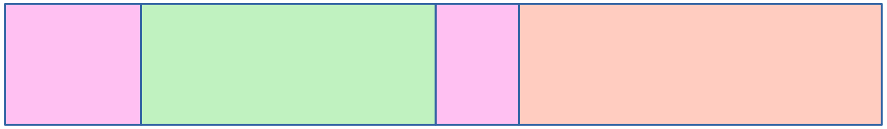
In PANDORA we plan and execute missions over long-term horizons (days or weeks)

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

To build a robust plan we therefore use estimated durations for the actions that are longer than the mean.
(95th percentile of the normal distribution)







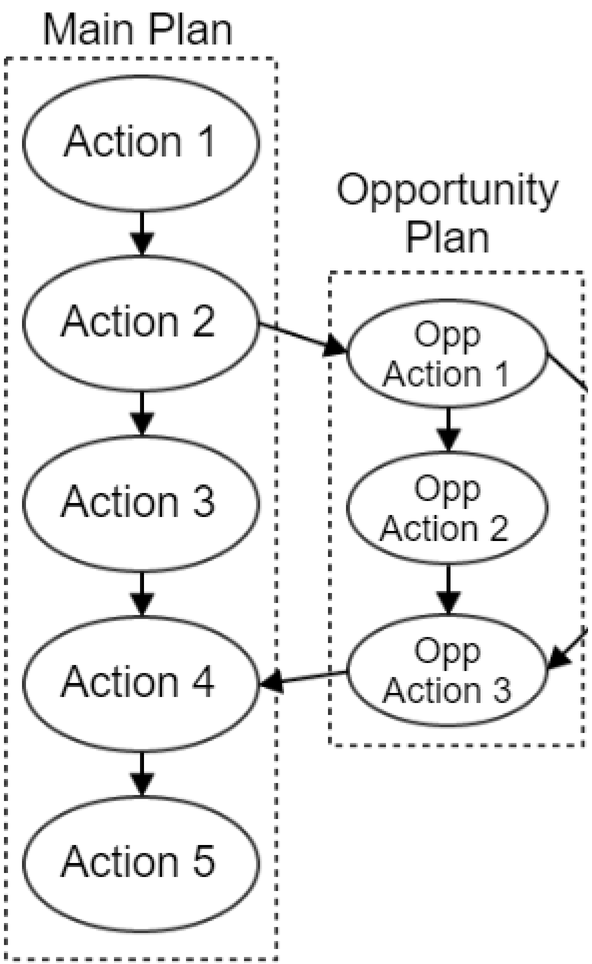
Opportunistic Planning

We use an execution stack (of goals & plans)

The current plan tail can be pushed onto the stack

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.



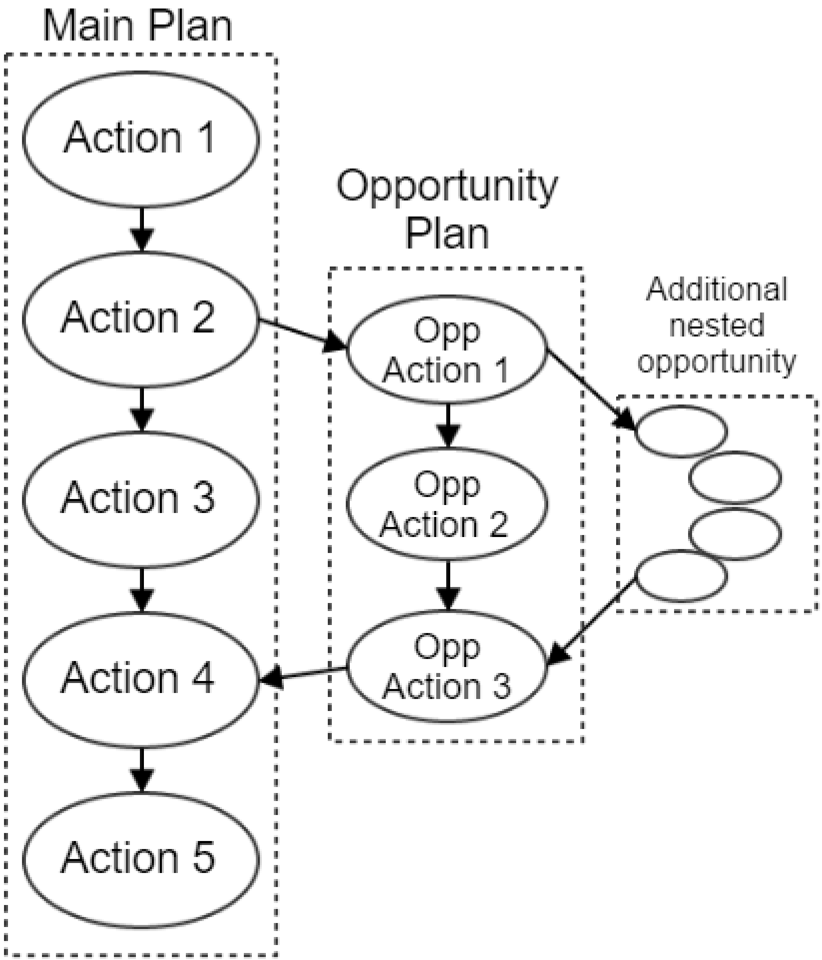
Opportunistic Planning

We use an execution stack (of goals & plans)

The current plan tail can be pushed onto the stack

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.



Why not just replan?

We compare the opportunistic approach against replanning the mission when an opportunity is discovered. When an opportunity is discovered a new initial state is generated.

Replanning:

- the problem is more difficult to solve
- the planning time can be increased

+ the opportunity can be ordered later in the plan
+ the existing plan can be reordered to make more time for exploiting the opportunity
+ the resulting plan can be more efficient

We examine situations where we have just discovered an opportunity:
10 second bound on planning for the opportunity alone
30 minute bound for replanning

Why not just replan?

Mission		Opp plan	Full replan	Plan duration		
Main	Opp	time	time	Opp Mission	Complete Opp Plan	Replanned plan
V2_400	I_16	0.36	38.18	851.384	1265.032	2437.496
V2_500	I_16	5.54	7.46	1541.168	2076.155	2596.156
V2_600	I_16	5.34	7.28	1541.168	2117.136	2269.701
V2_700	I_16	5.32	9.56	1541.168	2117.136	2283.134
V2_800	I_16	5.38	6.24	1541.168	2117.136	**2048.833**
V2_900	I_16	5.4	9.16	1541.168	2117.136	**1900.069**
V2_1000	I_16	0.38	21.42	851.384	1265.032	2615.245
V2_1100	I_16	0.34	7.28	888.554	1302.202	2048.833
V2_1200	I_16	2.4	11.9	1440.568	1854.216	2511.960
V2_1300	I_16	0.36	6.34	851.384	1265.032	2772.985
V2_1400	I_16	0.42	6.28	851.384	1265.032	2772.985
V2_1500	I_16	0.34	7.82	851.384	1265.032	2946.391
V2_1600	I_16	0.38	14.54	851.384	1265.032	2175.901
V2_1700	I_16	0.4	15.6	851.384	1265.032	2897.665
V2_1800	I_16	0.42	6.24	851.384	1265.032	2772.985
V2_1900	I_16	0.38	6.44	851.384	1265.032	2772.985
V2_2000	I_16	0.36	2.62	851.384	1265.032	2490.490
V2_400	I_32	5.08	148.17	2233.961	2564.254	3531.784
V2_500	I_32	2.2	165.62	1768.98	2129.213	5332.514
V2_600	I_32	3.7	78.19	1777.177	2137.41	3623.974
V2_700	I_32	4.08	272.84	1815.849	2176.082	4877.45
V2_1000	I_32	4.66	104.04	2686.638	3093.992	4263.605
V2_2000	I_32	4.32	100.16	2457.922	2865.276	3778.601

Why not just replan?

Mission		Opp plan	Full replan	Plan duration		
Main	Opp	time	time	Opp Mission	Complete Opp Plan	Replanned plan
V2_400	I_16	0.36	38.18	851.384	1265.032	2437.496
V2_500	I_16	5.54	7.46	1541.168	2076.155	2596.156
V2_600	I_16	5.34	7.28	1541.168	2117.136	2269.701
V2_700	I_16	5.32	9.56	1541.168	2117.136	2283.134
V2_800	I_16	5.38	6.24	1541.168	2117.136	**2048.833**
V2_900	I_16	5.4	9.16	1541.168	2117.136	**1900.069**
V2_1000	I_16	0.38	[illegible]	[illegible]	[illegible]	2615.245
V2_1100	I_16	0.34	[illegible]	[illegible]	[illegible]	2048.833
V2_1200	I_16	2.4	11.9	1440.568	1854.216	2511.960
V2_1300	I_16	0.36	6.34	851.384	1265.032	2772.985
V2_1400	I_16	0.42	6.28	851.384	1265.032	2772.985
V2_1500	I_16	0.34	7.82	851.384	1265.032	2946.391
V2_1600	I_16	0.38	14.54	851.384	1265.032	2175.901
V2_1700	I_16	0.4	15.6	851.384	1265.032	2897.665
V2_1800	I_16	0.42	6.24	851.384	1265.032	2772.985
V2_1900	I_16	0.38	6.44	851.384	1265.032	2772.985
V2_2000	I_16	0.36	2.62	851.384	1265.032	2490.490
V2_400	I_32	5.08	148.17	2233.961	2564.254	3531.784
V2_500	I_32	2.2	165.62	1768.98	2129.213	5332.514
V2_600	I_32	3.7	78.19	1777.177	2137.41	3623.974
V2_700	I_32	4.08	272.84	1815.849	2176.082	4877.45
V2_1000	I_32	4.66	104.04	2686.638	3093.992	4263.605
V2_2000	I_32	4.32	100.16	2457.922	2865.276	3778.601

Better plan quality by replanning

Why not just replan?

Mission		Opp plan	Full replan	Plan duration		
Main	Opp	time	time	Opp Mission	Complete Opp Plan	Replanned plan
V2_400	I_16	0.36	38.18	851.384	1265.032	2437.496
V2_500	I_16	5.54	7.46	1541.168	2076.155	2596.156
V2_600	I_16	5.34	7.28	1541.168	2117.136	2269.701
V2_700	I_16	5.32	9.56	1541.168	2117.136	2283.134
V2_800	I_16	5.38	6.24	1541.168	2117.136	**2048.833**
V2_900	I_16	5.4	9.16	1541.168	2117.136	**1900.069**
V2_1000	I_16	0.38	[illegible]	[illegible]	[illegible]	2615.245
V2_1100	I_16	0.34	[illegible]	[illegible]	[illegible]	2048.833
V2_1200	I_16	2.4	11.9	1440.568	1854.216	2511.960
V2_1300	I_16	0.36	6.34	851.384	1265.032	2772.985
V2_1400	I_16	0.42	6.28	851.384	1265.032	2772.985
V2_1500	I_16	0.34	7.82	851.384	1265.032	2946.391
V2_1600	I_16	0.38	14.54	851.384	1265.032	2175.901
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Better plan quality by replanning

In **228** total missions:
5 replanning plans were more efficient than the opportunistic approach.

We examine situations where we have just discovered an opportunity:
10 second bound on planning for the opportunity alone
30 minute bound for replanning

Opportunistic Planning

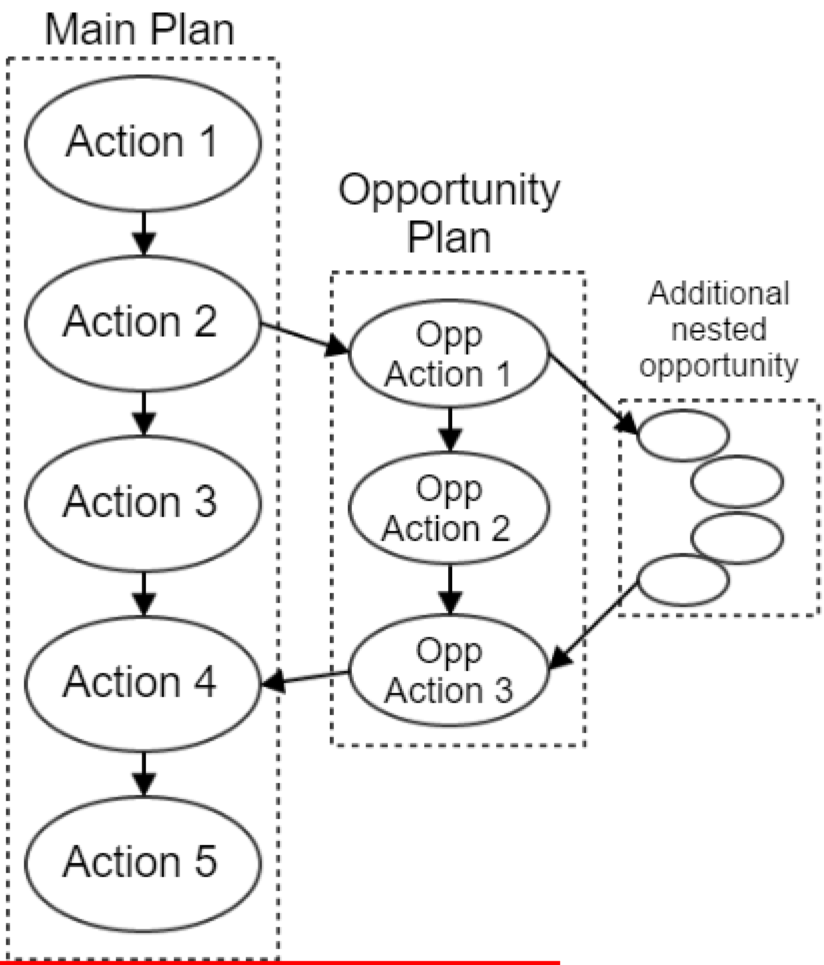
We use an execution stack (of goals & plans)

The current plan tail can be pushed onto the stack

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.

NOTE: Opportunities can also arise for supervisor requests!



More details on Friday morning
(Paper on Opportunistic Planning at the Journal Track)

Outline

- Why PDDL Planning for Robotics and HRI?
  - Expressive Planning
  - Opportunistic Planning
  - Strategic Planning
  - eXplainable Planning (XAIP)
  - Planning with Uncertainty

Strategic Planning for Persistent Autonomy

Planning over long horizons (days, weeks)

Missions with strict deadlines and time windows in which goals need to be accomplished.

Example in underwater robotics:
Seabed facilities need to be inspected at certain intervals.

Current planning systems struggle in generating complex plans over long horizons.

One possible solution:
Decompose into **Strategic/Tactical Layers**

Strategic/Tactical Planning

Cluster the *goals* into *tasks*

Strategic Layer: contains a high lever plan that achieves all tasks and manages the resource and time constraints.

Tactical Layer: contains a plan that solves a single task.

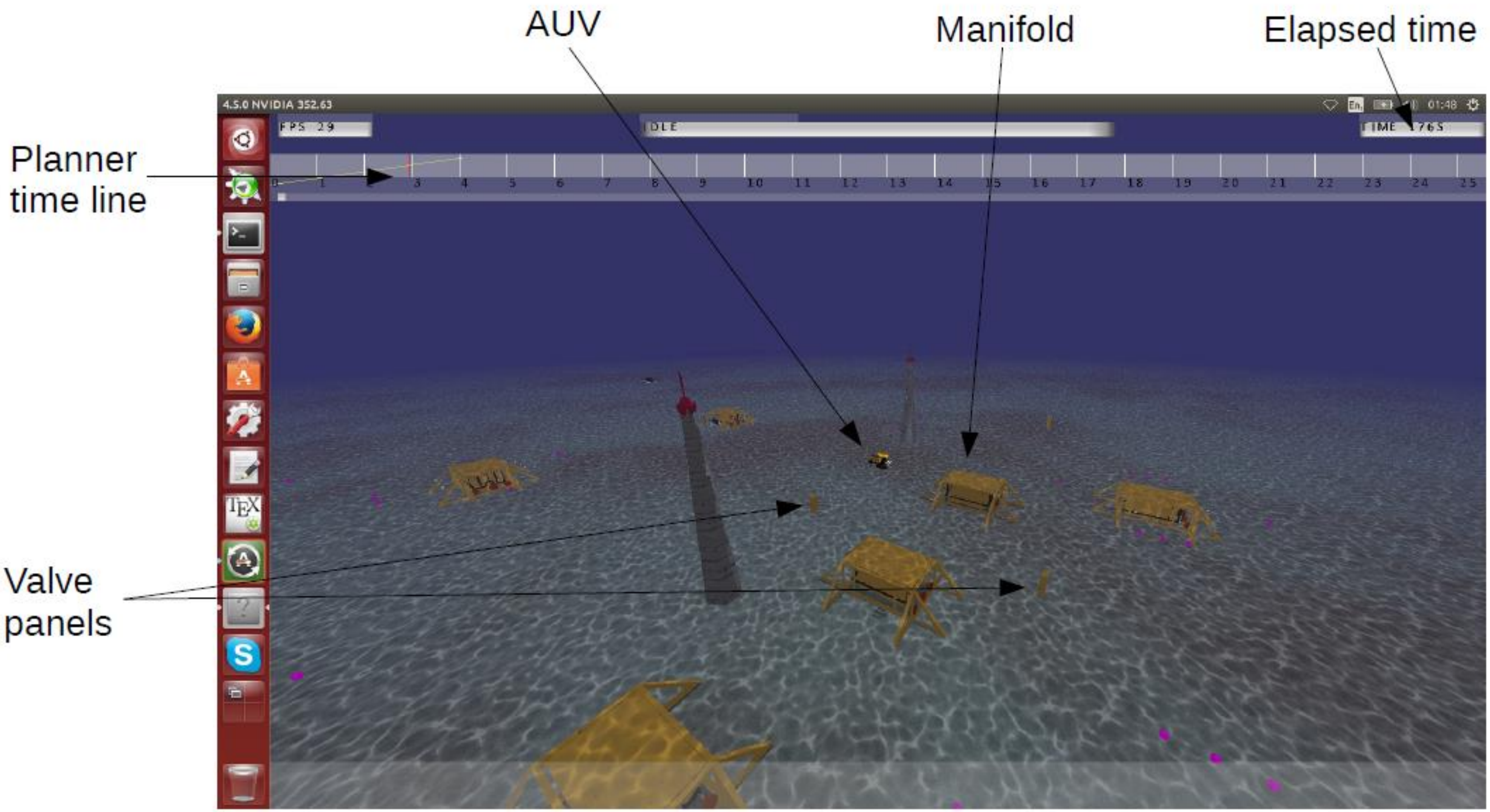
Example from underwater robotics.

Long term maintenance of seabed facility includes

-Inspecting the structures are regular intervals.
-Changing the configuration of the site by interacting with interfaces within specific time windows.
-Recharging the AUVs.

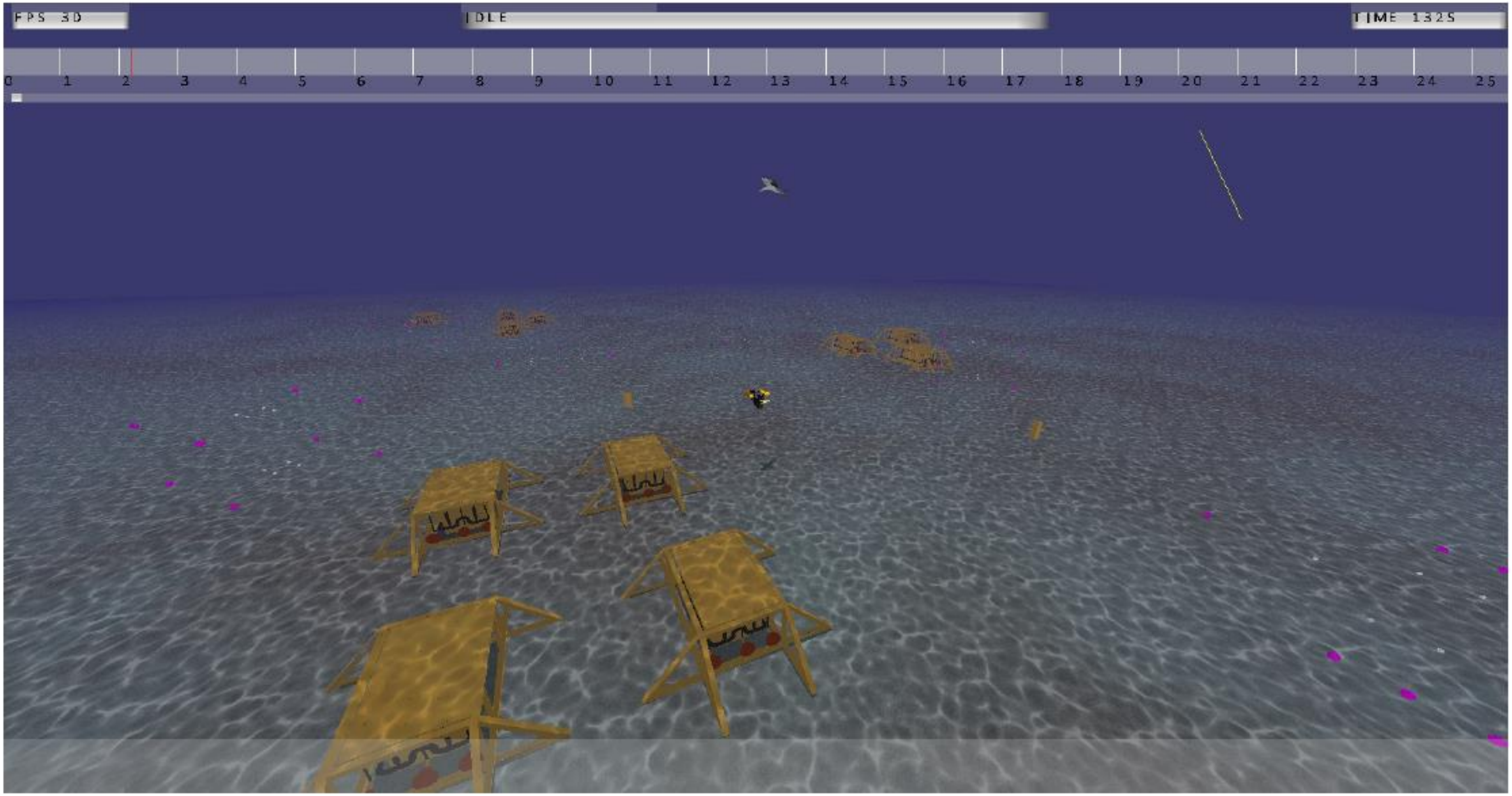
Additional challenges:
-Ever changing environment (currents, visibility)
-Wildlife

Strategic/Tactical Planning



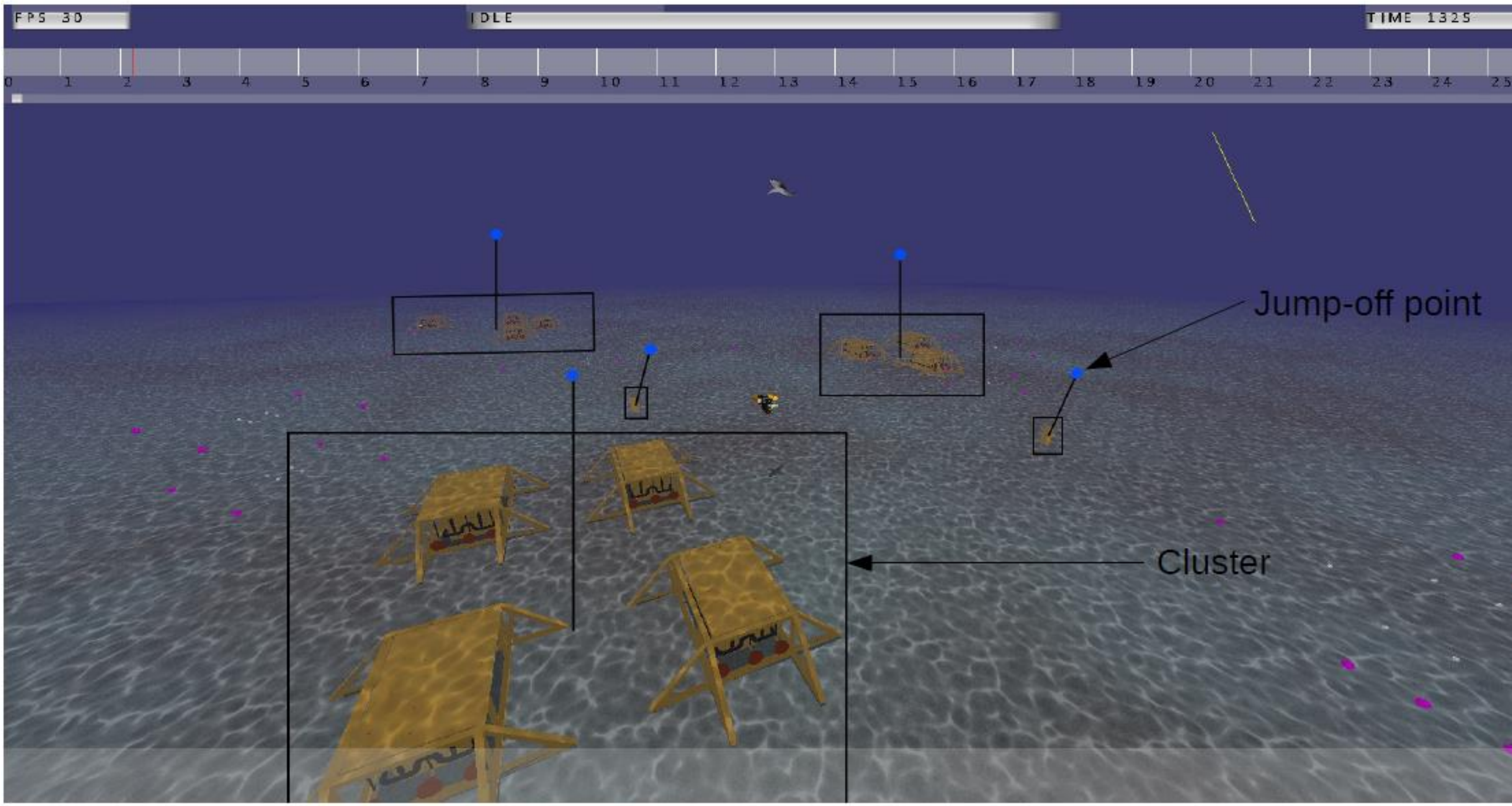
Strategic/Tactical Planning

Clustering



Strategic/Tactical Planning

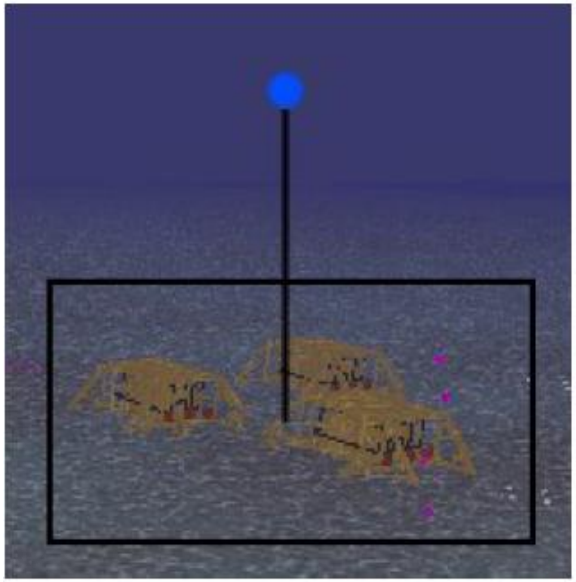
Clustering



Strategic/Tactical Planning

Tactical Layer

For each Task the planner generates a plan and stores:
-duration
-resource constraints



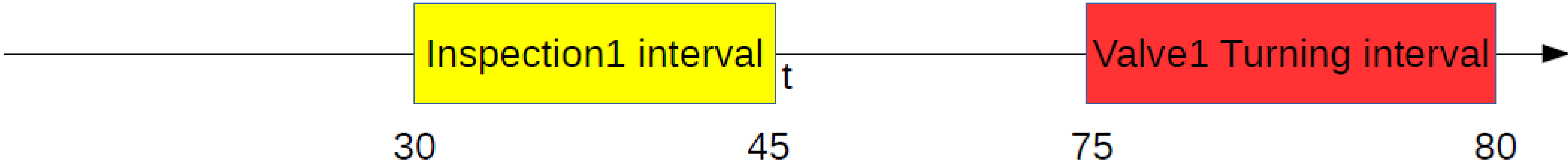
```
0.000: (correct_position auv0 wp_auv0) [3.000]
3.001: (do_hover_fast auv0 wp_auv0 strategic_location_7)
[11.403]
14.405: (correct_position auv0_strategic_location_78)
[3.000]
17.406: (observe_inspection_point auv0 strategic_location_7
inspection_point_2) [10.000]
27.407: (correct_position auv0 strategic_location_7)
[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

Strategic/Tactical Planning

Strategic Layer

On the strategic layer the planner constructs a plan that conforms to the time and resource constraints.



Strategic/Tactical Planning

Strategic Layer

On the strategic layer the planner constructs a plan that conforms to the time and resource constraints.

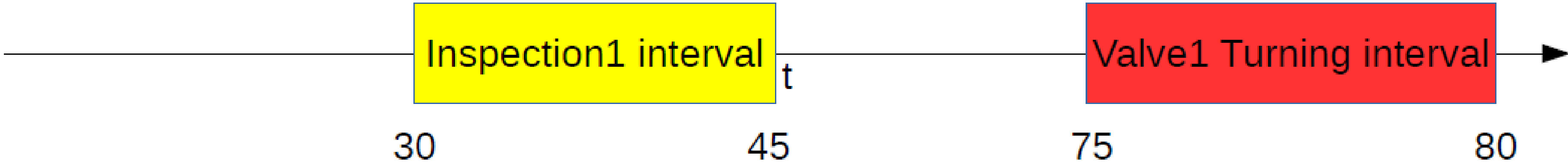
All the tactical plans are collected.

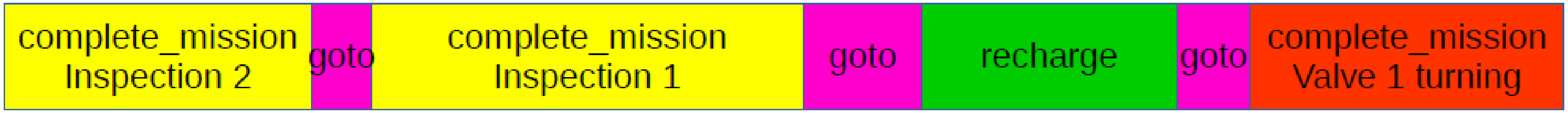
complete_mission Inspection 1

complete_mission Inspection 2

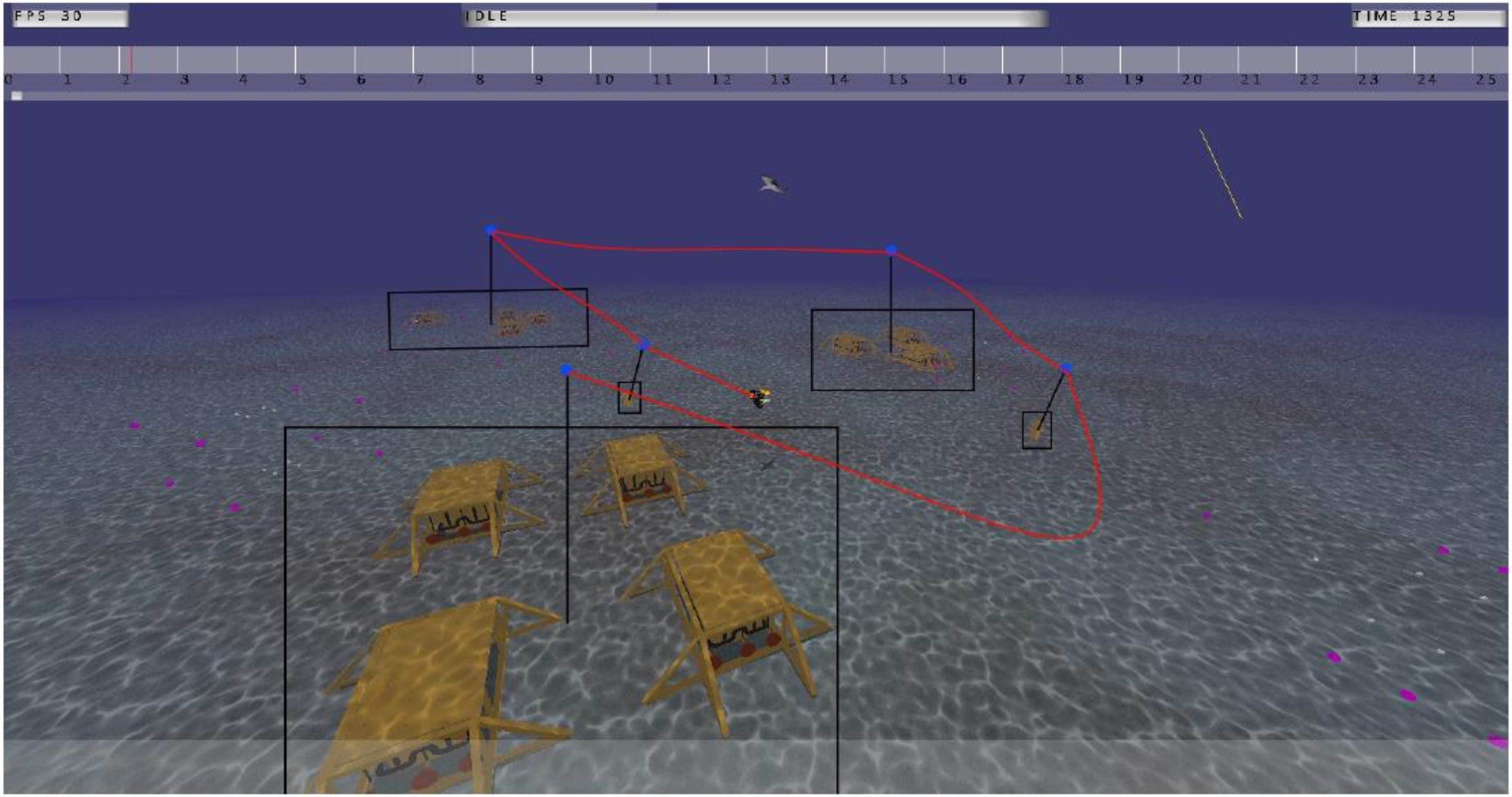
complete_mission Valve 1 turning

And the strategic plan is generated, not violating resource/time constraints





Strategic/Tactical Planning



Outline

- Why PDDL Planning for Robotics and HRI?
  - Expressive Planning
  - Opportunistic Planning
  - Strategic Planning
  - eXplainable Planning (XAIP)
  - Planning with Uncertainty

eXplainable Planning (XAIP)

Planners can be trusted

Planners can allow an easy interaction with humans

Planners are transparent
(at least, the process by which the decisions are made are understood by their programmers)

To note: entirely trustworthy and theoretically well-understood algorithms can still yield decisions that are hard to explain.
Ex: Linear Programming ….

To note: XAI and the need to explain machine/deep learning remain of critical importance!
XAIP is important in domains where learning is not an option.

What eXplainable Planning is NOT !

XAIP is **not** explaining what is **obvious** !

Many planners select actions in their plan-construction process by minimising a heuristic distance to goal (relaxed plan)

Q: *Why did the planner do that* ?

A: *Because it got me closer to the goal* !

What eXplainable Planning is NOT !

XAIP is **not** explaining what is **obvious** !

Many planners select actions in their plan-construction process by minimising a heuristic distance to goal (relaxed plan)

Q: *Why did the planner do that* ?

~~A: *Because it got me closer to the goal* !~~

What eXplainable Planning is NOT !

XAIP is **not** explaining what is **obvious** !

Many planners select actions in their plan-construction process by minimising a heuristic distance to goal (relaxed plan)

Q: *Why did the planner do that* ?

A: ~~*Because it got me closer to the goal* !~~

A request for an explanation is an attempt to uncover a piece of knowledge that the questioner believes must be available to the system and that the questioner does not have.

Towards XAIP

- Plan explanation
  - Translate PDDL in forms that humans can understand [Sohrabi et al. 2012]
  - Design interfaces that help this understanding [Bidot et al. 2012]
  - Describe causal/temporal relations for plan steps [Seegebarth et al. 2012]
  - Explaining observed behaviours [Sohrabi, Baier, McIlraith, 2011]
  - Understanding the past [Molineaux et al., 2012]
  - … ... …
- Plan Explicability
  - Focus on human's interpretation of plans [Seegebarth et al. 2012]
- Verbalization and *transparency* in autonomy
  - Generate narrations for autonomous robot navigations [Veloso et al. 2016]
- Explainable Agency [Langley et al. 2017]
- Model Reconciliation (Sreedharan et al.)
  - Identify/reconcile different human/robot models [Chakraborti et al 2017]

Transparency in Autonomy

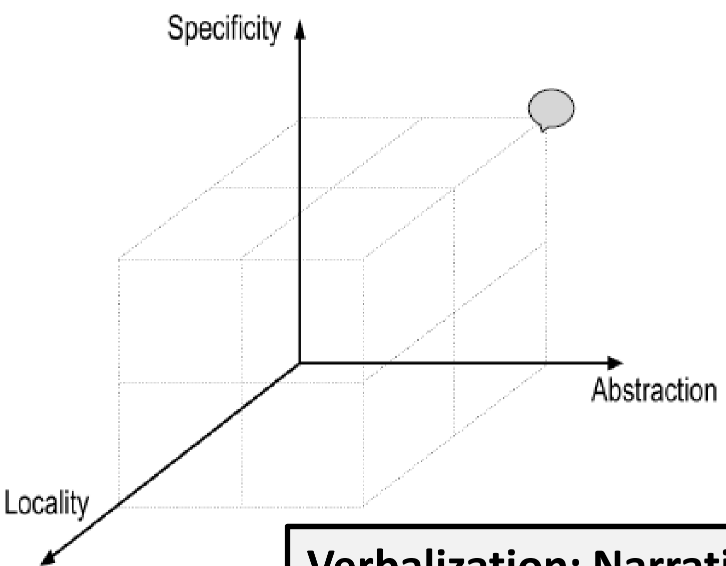
(Manuela Veloso et al.)

Verbalization: the process by which an autonomous robots converts its own experience into language

Verbalization space: to capture different nature of explanations.

And to learn to correctly infer an explanation level in the verbalization space.

Specificity – Locality - Abstraction



"Please tell me exactly how you got here"

"OK, now only tell me what happened near the room 7004"

"Can you only give me a brief summary?"

Verbalization: Narration of Autonomous Mobile Robot Experience.
Rosenthal, Selvaraj, Veloso. IJCAI 2016.

Things to Be Explained (*some*)

- Q1: Why did you do that?
- Q2: Why didn't you do *something else*? (that I would have done)
- Q3: Why is what you propose to do more efficient/safe/cheap than something else? (that I would have done)
- Q4: Why can't you do that ?
- Q5: Why do I need to replan at this point?
- Q6: Why do I not need to replan at this point?

Illustrative Example

`Rover Time` domain from IPC-4 (problem 3)

```
0.000: (navigate r1 wp3 wp0)  [5.0]
0.000: (navigate r0 wp1 wp0)  [5.0]
5.001: (calibrate r1 camera1 obj0 wp0)  [5.0]
5.001: (sample_rock r0 r0store wp0)  [8.0]
10.002: (take_image r1 wp0 obj0 camera1 col)  [7.0]
13.001: (navigate r0 wp0 wp1)  [5.0]
17.002: (navigate r1 wp0 wp3)  [5.0]
18.001: (comm_rock_data r0 general wp0 wp1 wp0)  [10.0]
22.003: (navigate r1 wp3 wp2)  [5.0]
27.003: (sample_soil r1 r1store wp2)  [10.0]
28.002: (comm_image_data r1 general obj0 col wp2 wp0) [15.0]
43.003: (comm_soil_data r1 general wp2 wp2 wp0)  [10.0]

[Duration = 53.003]
```

Q1: *why did you use Rover0 to take the rock sample at waypoint0* ?

NA: *so that I can communicate_data from Rover0 later (at 18.001)*

Illustrative Example

`Rover Time` domain from IPC-4 (problem 3)

```
0.000: (navigate r1 wp3 wp0)   [5.0]
0.000: (navigate r0 wp1 wp0)   [5.0]
5.001: (calibrate r1 camera1 obj0 wp0)   [5.0]
5.001: (sample_rock r0 r0store wp0)   [8.0]
10.002: (take_image r1 wp0 obj0 camera1 col)   [7.0]
13.001: (navigate r0 wp0 wp1)   [5.0]
17.002: (navigate r1 wp0 wp3)   [5.0]
18.001: (comm_rock_data r0 general wp0 wp1 wp0)   [10.0]
22.003: (navigate r1 wp3 wp2)   [5.0]
27.003: (sample_soil r1 r1store wp2)   [10.0]
28.002: (comm_image_data r1 general obj0 col wp2 wp0) [15.0]
43.003: (comm_soil_data r1 general wp2 wp2 wp0)   [10.0]

[Duration = 53.003]
```

Q1: *why did you use Rover0 to take the rock sample at waypoint0 ?*

NA: *so that I can communicate_data from Rover0 later (at 18.001)*

Illustrative Example

`Rover Time` domain from IPC-4 (problem 3)

```
0.000: (navigate r1 wp3 wp0)   [5.0]
0.000: (navigate r0 wp1 wp0)   [5.0]
5.001: (calibrate r1 camera1 obj0 wp0)   [5.0]
5.001: (sample_rock r0 r0store wp0)   [8.0]
10.002: (take_image r1 wp0 obj0 camera1 col)   [7.0]
13.001: (navigate r0 wp0 wp1)   [5.0]
17.002: (navigate r1 wp0 wp3)   [5.0]
18.001: (comm_rock_data r0 general wp0 wp1 wp0)   [10.0]
22.003: (navigate r1 wp3 wp2)   [5.0]
27.003: (sample_soil r1 r1store wp2)   [10.0]
28.002: (comm_image_data r1 general obj0 col wp2 wp0) [15.0]
43.003: (comm_soil_data r1 general wp2 wp2 wp0)   [10.0]

[Duration = 53.003]
```

Q1: *why did you use Rover0 to take the rock sample at waypoint0 ?*

why didn't Rover1 take the rock sample at waypoint0 ?

Illustrative Example

Q1: *why did you use Rover0 to take the rock sample at waypoint0 ?*

why didn't Rover1 take the rock sample at waypoint0 ?

We remove the ground action instance for Rover0 and re-plan

A: *Because not using Rover0 for this action leads to a longer plan*

```
0.000: (navigate r1 wp3 wp0)  [5.0]
5.001: (calibrate r1 camera1 obj0 wp0)  [5.0]
10.002: (take_image r1 wp0 obj0 camera1 col)  [7.0]
10.003: (sample_rock r1 r1store wp0)  [8.0]
18.003: (navigate r1 wp0 wp3)  [5.0]
18.004: (drop r1 r1store)  [1.0]
23.004: (navigate r1 wp3 wp2)  [5.0]
28.004: (comm_image_data r1 general obj0 col wp2 wp0) [15.0]
28.005: (sample_soil r1 r1store wp2)  [10.0]
43.005: (comm_soil_data r1 general wp2 wp2 wp0)  [10.0]
53.006: (comm_rock_data r1 general wp0 wp2 wp0)  [10.0]

[Duration = 63.006]
```

```
0.000: (naviga
0.000: (naviga
5.001: (calibr
5.001: (sample
10.002: (take_
13.001: (navig
17.002: (navig
18.001: (comm_
22.003: (navig
27.003: (sampl
28.002: (comm_image_data r1 general obj0 col wp2 wp0) [15.0]
43.003: (comm_soil_data r1 general wp2 wp2 wp0)  [10.0]

[Duration = 53.003]
```

Illustrative Example

Q1: *why did you use Rover0 to take the rock sample at waypoint0 ?*

why didn't Rover1 take the rock sample at waypoint0 ?

We remove the ground action instance for Rover0 and re-plan

A: *Because not using Rover0 for this action leads to a longer plan*

Q2: But why does Rover1 do everything in this plan?

```
0.000: (navigate r1 wp3 wp0)  [5.0]
5.001: (calibrate r1 camera1 obj0 wp0)  [5.0]
10.002: (take_image r1 wp0 obj0 camera1 col)  [7.0]
10.003: (sample_rock r1 r1store wp0)  [8.0]
18.003: (navigate r1 wp0 wp3)  [5.0]
18.004: (drop r1 r1store)  [1.0]
23.004: (navigate r1 wp3 wp2)  [5.0]
28.004: (comm_image_data r1 general obj0 col wp2 wp0) [15.0]
28.005: (sample_soil r1 r1store wp2)  [10.0]
43.005: (comm_soil_data r1 general wp2 wp2 wp0)  [10.0]
53.006: (comm_rock_data r1 general wp0 wp2 wp0)  [10.0]

[Duration = 63.006]
```

Illustrative Example

Q1: *why did you use Rover0 to take the rock sample at waypoint0 ?*

why didn’t Rover1 take the rock sample at waypoint0 ?

We remove the ground action instance for Rover0 and re-plan

A: *Because not using Rover0 for this action leads to a longer plan*

Q2: But why does Rover1 do everything in this plan?

We require the plan to contain at least one action that has Rover0 as argument (add dummy effect to all actions using Rover0 and put into the goal)

```
0.000:  (na
5.001:  (ca
10.002:  (t
10.003:  (s
18.003:  (r
18.004:  (d
23.004:  (r
28.004:  (c
28.005:  (s
43.005:  (c
53.006:  (c

[Duration
```

```
0.000:  (navigate r0 wp1 wp0)   [5.0]
0.000:  (navigate r1 wp3 wp0)   [5.0]
5.001:  (calibrate r1 camera1 obj0 wp0)   [5.0]
10.002:  (take_image r1 wp0 obj0 camera1 col)   [7.0]
10.003:  (sample_rock r1 r1store wp0)   [8.0]
18.003:  (navigate r1 wp0 wp3)   [5.0]
18.004:  (drop r1 r1store)   [1.0]
23.004:  (navigate r1 wp3 wp2)   [5.0]
28.004:  (comm_image_data r1 general obj0 col wp2 wp0)   [15.0]
28.005:  (sample_soil r1 r1store wp2)   [10.0]
43.005:  (comm_soil_data r1 general wp2 wp2 wp0)   [10.0]
53.006:  (comm_rock_data r1 general wp0 wp2 wp0)   [10.0]
```

Illustrative Example

Q1: *why did you use Rover0 to take the rock sample at waypoint0 ?*

why didn't Rover1 take the rock sample at waypoint0 ?

We remove the ground action instance for Rover0 and re-plan

A: ***Because not using Rover0 for this action leads to a longer plan***

Q2: But why does Rover1 do everything in this plan?

We require the plan to contain at least one action that has Rover0 as argument (add dummy effect to all actions using Rover0 and put into the goal)

A: ***There is no useful way to use Rover0 for improve this plan***

```
0.000: (navigate r0 wp1 wp0)   [5.0]
0.000: (navigate r1 wp3 wp0)   [5.0]
5.001: (calibrate r1 camera1 obj0 wp0)   [5.0]
10.002: (take_image r1 wp0 obj0 camera1 col)   [7.0]
10.003: (sample_rock r1 r1store wp0)   [8.0]
18.003: (navigate r1 wp0 wp3)   [5.0]
18.004: (drop r1 r1store)   [1.0]
23.004: (navigate r1 wp3 wp2)   [5.0]
28.004: (comm_image_data r1 general obj0 col wp2 wp0)   [15.0]
28.005: (sample_soil r1 r1store wp2)   [10.0]
43.005: (comm_soil_data r1 general wp2 wp2 wp0)   [10.0]
53.006: (comm_rock_data r1 general wp0 wp2 wp0)   [10.0]
```

eXplainable Planning
at execution time

- Q5: Why do I need to replan at this point?

In many real-world scenarios, it is not obvious that the plan being executed will fail. Often plain failures is discovered too late.

One possible approach is to use the “*Filter Violation*” (ROSPlan)

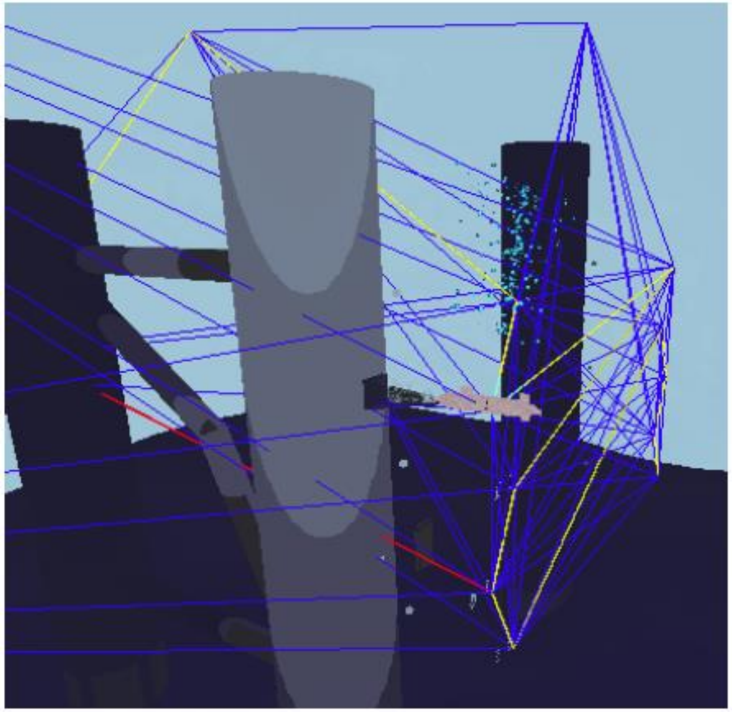
Once the plan is generated, ROSPlan creates a filter, by considering all the preconditions of the actions in the plan.

Ex: navigate (?from ?to - waypoint) has precondition (connected ?from ?to)
If the plan contains navigate (wp3 wp5),
then (connected wp3 wp5) is added to the filter.

Illustrative Example

AUV domain from (Cashmore et al, ICRA 2015)

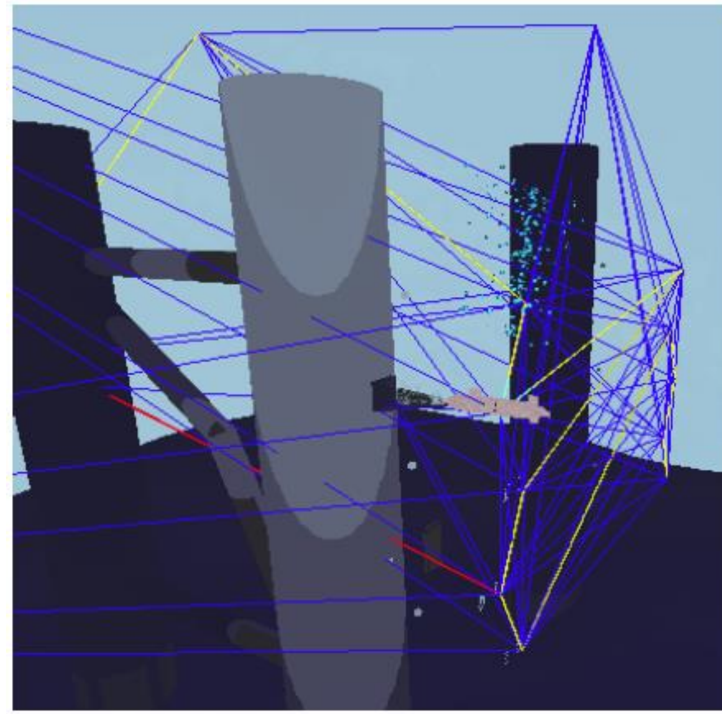
```
0.000: (observe auv wp1 ip3)  [10.000]
10.001: (correct_position auv wp1)  [10.000]
20.002: (do_hover auv wp1 wp2)  [71.696]
91.699: (observe auv wp2 ip4)  [10.000]
101.700: (correct_position auv wp2)  [10.000]
111.701: (do_hover auv wp2 wp23)  [16.710]
128.412: (observe auv wp23 ip5)  [10.000]
138.413: (correct_position auv wp23)  [10.000]
148.414: (observe auv wp23 ip1)  [10.000]
158.415: (correct_position auv wp23)  [10.000]
168.416: (do_hover auv wp23 wp22)  [16.710]
185.127: (do_hover auv wp22 wp26)  [30.201]
215.329: (observe auv wp26 ip7)  [10.000]
225.330: (correct_position auv wp26)  [10.000]
235.331: (do_hover auv wp26 wp21)  [23.177]
258.509: (observe auv wp21 ip2)  [10.000]
268.510: (correct_position auv wp21)  [10.000]
278.511: (do_hover auv wp21 wp27)  [21.255]
299.767: (observe auv wp27 ip8)  [10.000]
309.768: (correct_position auv wp27)  [10.000]
319.769: (observe auv wp27 ip6)  [10.000]
329.770: (correct_position auv wp27)  [10.000]
339.771: (do_hover auv wp27 wp17)  [23.597]
363.369: (do_hover auv wp17 wp25)  [21.413]
384.783: (do_hover auv wp25 wp32)  [16.710]
401.494: (do_hover auv wp32 wp36)  [21.451]
422.946: (observe auv wp36 ip9)  [10.000]
432.947: (correct_position auv wp36)  [10.000]
442.948: (observe auv wp36 ip15)  [10.000]
```



Illustrative Example

AUV domain from (Cashmore et al, ICRA 2015)

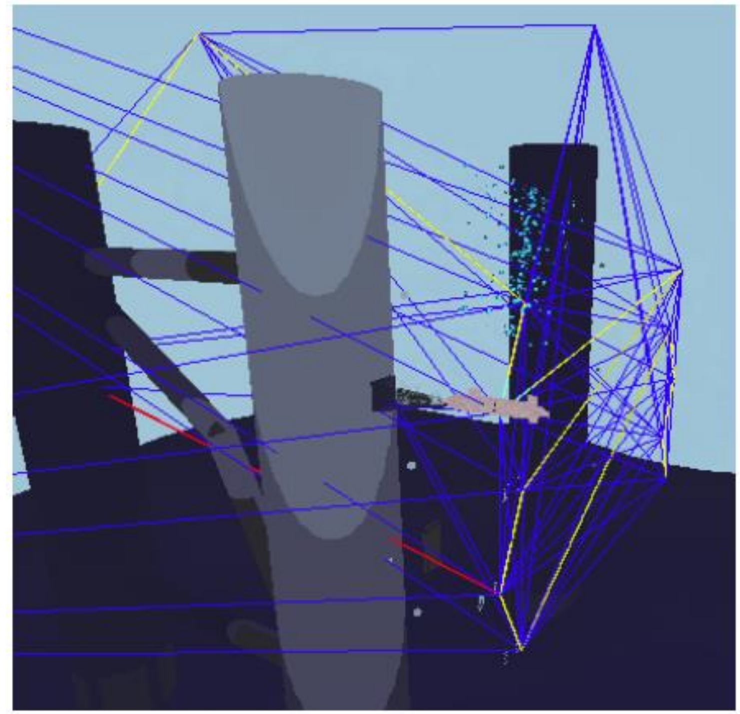
```
0.000: (observe auv wp1 ip3)  [10.000]
10.001: (correct_position auv wp1)  [10.000]
20.002: (do_hover auv wp1 wp2)  [71.696]
91.699: (observe auv wp2 ip4)  [10.000]
101.700: (correct_position auv wp2)  [10.000]
111.701: (do_hover auv wp2 wp23)  [16.710]
128.412: (observe auv wp23 ip5)  [10.000]
138.413: (correct_position auv wp23)  [10.000]
148.414: (observe auv wp23 ip1)  [10.000]
158.415: (correct_position auv wp23)  [10.000]
168.416: (do_hover auv wp23 wp22)  [16.710]
185.127: (do_hover auv wp22 wp26)  [30.201]
215.329: (observe auv wp26 ip7)  [10.000]
225.330: (correct_position auv wp26)  [10.000]
235.331: (do_hover auv wp26 wp21)  [23.177]
258.509: (observe auv wp21 ip2)  [10.000]
268.510: (correct_position auv wp21)  [10.000]
278.511: (do_hover auv wp21 wp27)  [21.255]
299.767: (observe auv wp27 ip8)  [10.000]
309.768: (correct_position auv wp27)  [10.000]
319.769: (observe auv wp27 ip6)  [10.000]
329.770: (correct_position auv wp27)  [10.000]
339.771: (do_hover auv wp27 wp17)  [23.597]
363.369: (do_hover auv wp17 wp25)  [21.413]
384.783: (do_hover auv wp25 wp32)  [16.710]
401.494: (do_hover auv wp32 wp36)  [21.451]
422.946: (observe auv wp36 ip9)  [10.000]
432.947: (correct_position auv wp36)  [10.000]
442.948: (observe auv wp36 ip15)  [10.000]
```



Illustrative Example

AUV domain from (Cashmore et al, ICRA 2015)

```
0.000: (observe auv wp1 ip3)  [10.000]
10.001: (correct_position auv wp1)  [10.000]
20.002: (do_hover auv wp1 wp2)  [71.696]
91.699: (observe auv wp2 ip4)  [10.000]
101.700: (correct_position auv wp2)  [10.000]
111.701: (do_hover auv wp2 wp23)  [16.710]
128.412: (observe auv wp23 ip5)  [10.000]
138.413: (correct_position auv wp23)  [10.000]
148.414: (observe auv wp23 ip1)  [10.000]
158.415: (correct_position auv wp23)  [10.000]
168.416: (do_hover auv wp23 wp22)  [16.710]
185.127: (do_hover auv wp22 wp26)  [30.201]
215.329: (observe auv wp26 ip7)  [10.000]
225.330: (correct_position auv wp26)  [10.000]
235.331: (do_hover auv wp26 wp21)  [23.177]
258.509: (observe auv wp21 ip2)  [10.000]
268.510: (correct_position auv wp21)  [10.000]
278.511: (do_hover auv wp21 wp27)  [21.255]
299.767: (observe auv wp27 ip8)  [10.000]
309.768: (correct_position auv wp27)  [10.000]
319.769: (observe auv wp27 ip6)  [10.000]
329.770: (correct_position auv wp27)  [10.000]
339.771: (do_hover auv wp27 wp17)  [23.597]
363.369: (do_hover auv wp17 wp25)  [21.413]
384.783: (do_hover auv wp25 wp32)  [16.710]
401.494: (do_hover auv wp32 wp36)  [21.451]
422.946: (observe auv wp36 ip9)  [10.000]
432.947: (correct_position auv wp36)  [10.000]
442.948: (observe auv wp36 ip15)  [10.000]
```



Outline

- **Why PDDL Planning for Robotics?**
    - **Expressive Planning**
    - **Opportunistic Planning**
    - **Strategic Planning**
    - **eXplainable Planning (XAIP)**
    - **Planning with Uncertainty**

Planning with Uncertainty

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.
- Humans are unpredictable.
- The environment is often initially full of unknowns.

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

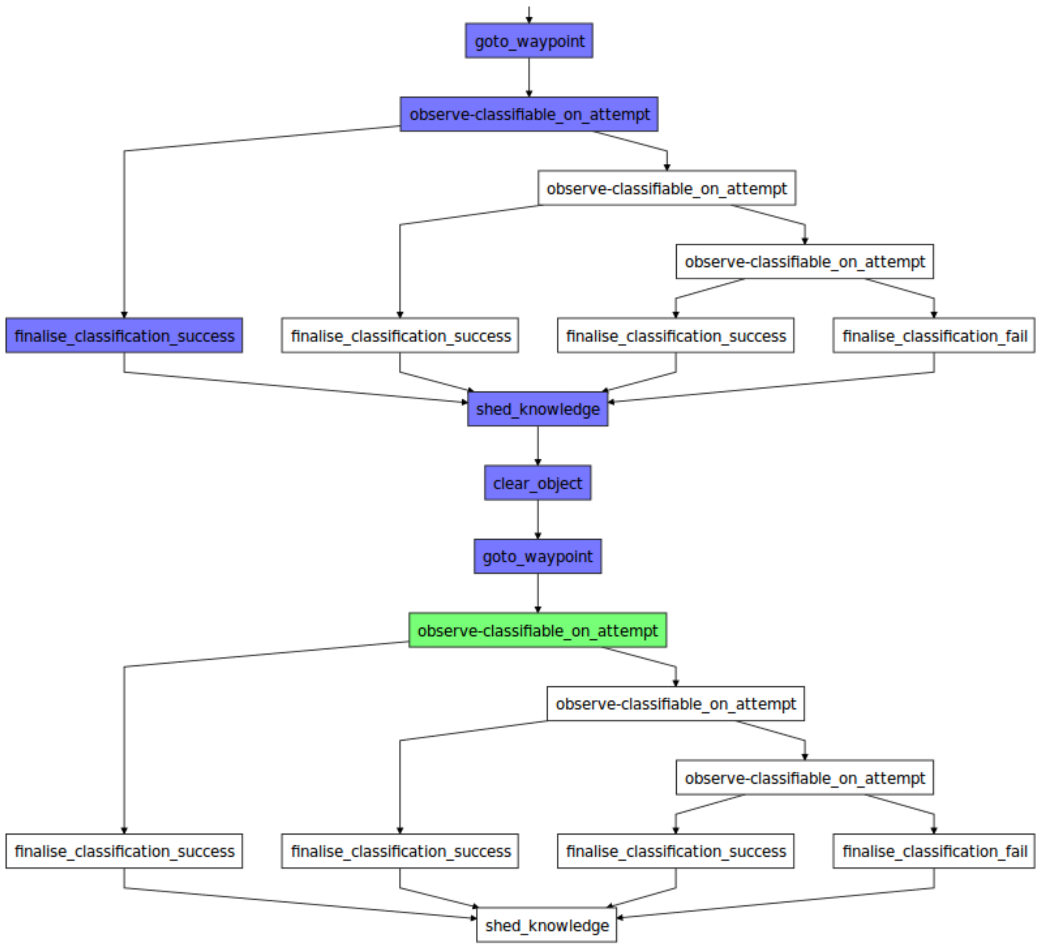
Uncertainty in AI Planning

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)



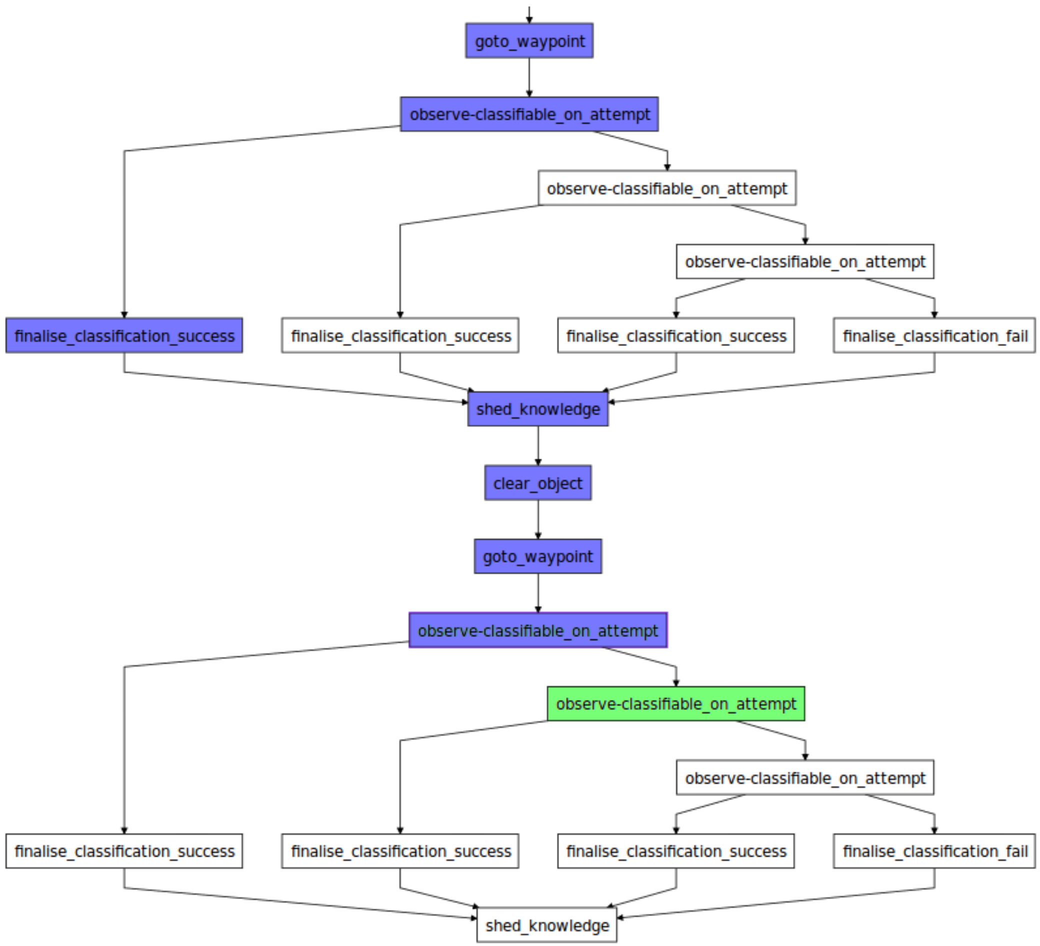
Uncertainty in AI Planning

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)



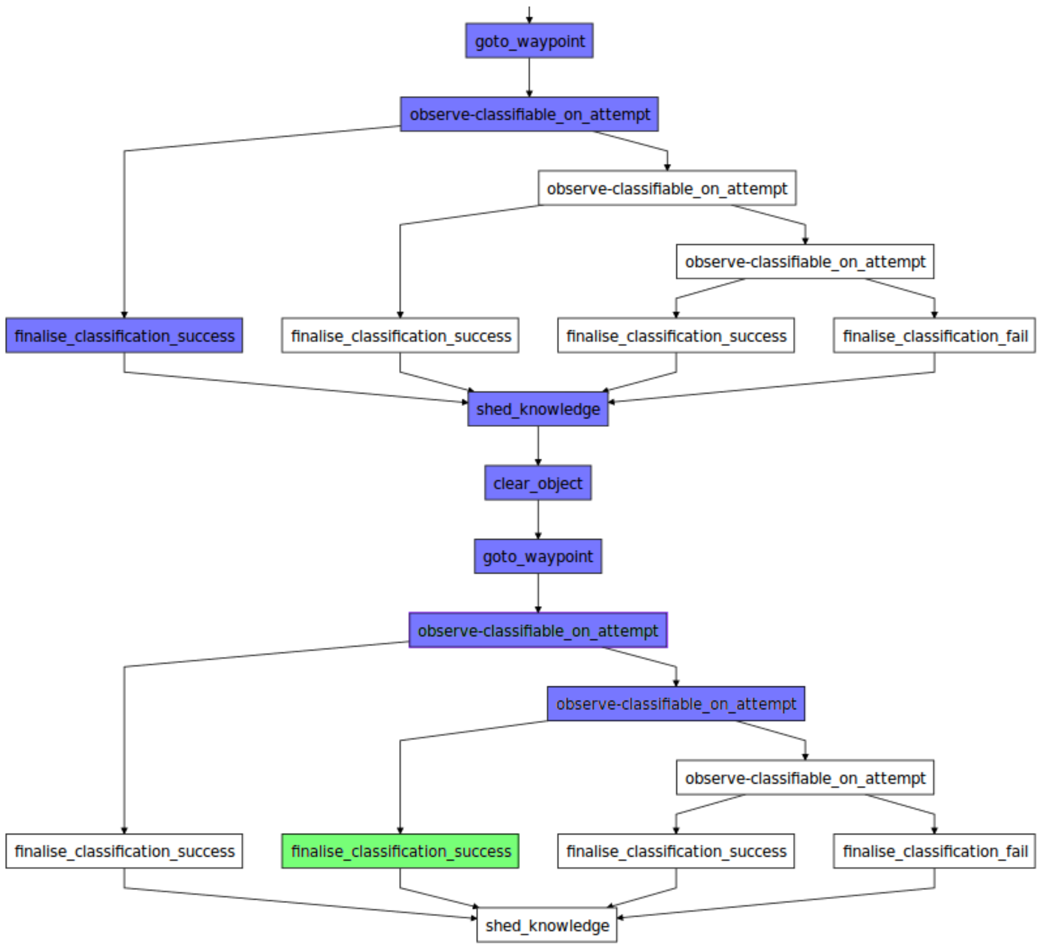
Uncertainty in AI Planning

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)



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)

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)

ROSPlan: Planning in the Robot Operating System

Outline

- ROS Basics
- Plan Execution
  - Very Simple Dispatch
  - Very Simple Temporal Dispatch
  - Conditional Dispatch
  - Temporal and Conditional Dispatch together
- Dispatching More than a Single Plan
  - Hierarchical and Recursive Planning
  - Opportunistic Planning

ROS Basics

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

A ROS system is composed of nodes, which pass messages, in two forms:

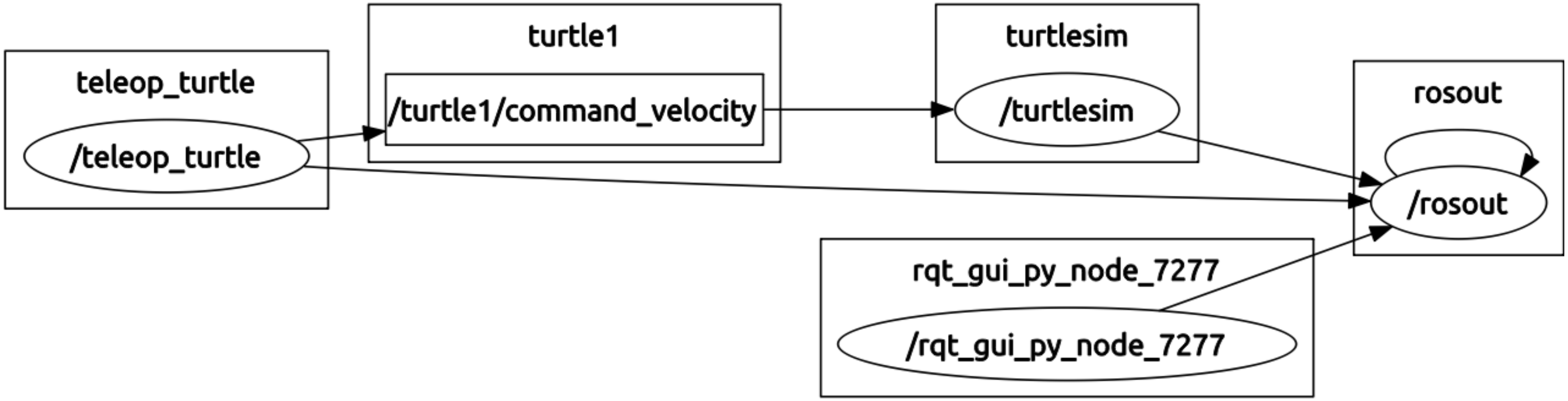
1. ROS messages are published on topics and are many-to-many.
2. ROS services are used for synchronous request/response.

ROS Basics

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

A ROS system is composed of nodes, which pass messages, 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, 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="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" />
    <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.

The actionlib package standardizes the interface for preemptable 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:

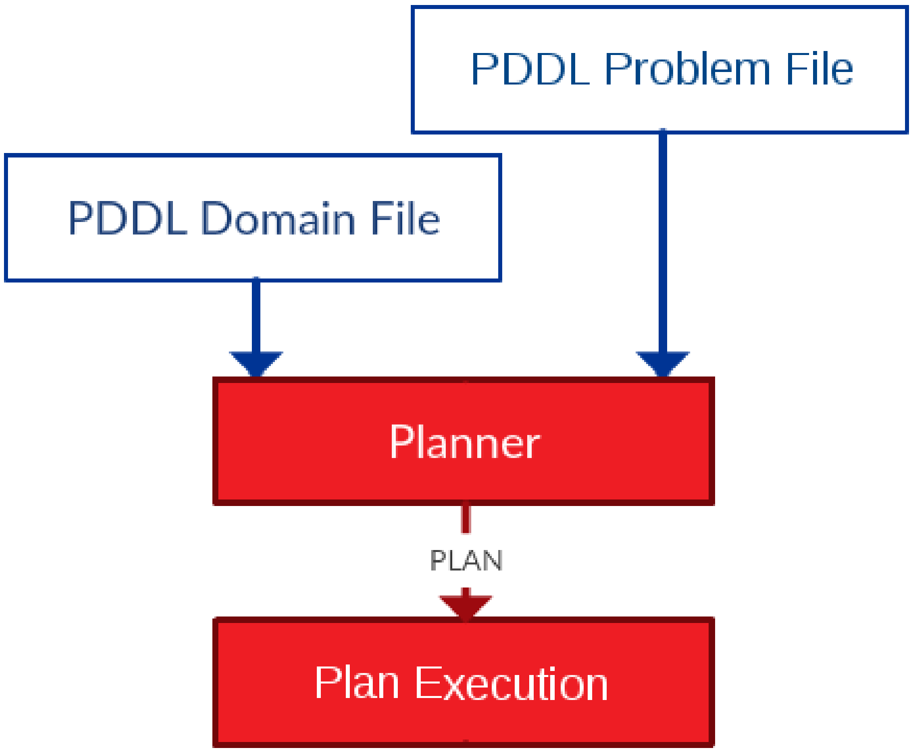
- goals
- feedback
- result

```
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
```

Plan Execution 1: Very simple Dispatch

The most basic structure.

- The plan is generated.
- The plan is executed.



Plan Execution 1: Very simple Dispatch

(Some) Related Work

McGann et el.C., Py, F., A deliberative architecture for AUV control. *In Proc. Int. Conf. on Robotics and Automation (ICRA),* 2008

Beetz & McDermott Improving Robot Plans During Their Execution. *In Proc. International Conference on AI Planning Systems (AIPS),* 1994

Ingrand et el. PRS: a high level supervision and control language for autonomous mobile robots. *In IEEE Int.l Conf. on Robotics and Automation,* 1996

Kortenkamp & Simmons Robotic Systems Architectures and Programming. *In Springer Handbook of Robotics, pp. 187–206,* 2008

Lemai-Chenevier & Ingrand Interleaving Temporal Planning and Execution in Robotics Domains. *In Proceedings of the National Conference on Artificial Intelligence (AAAI),* 2004

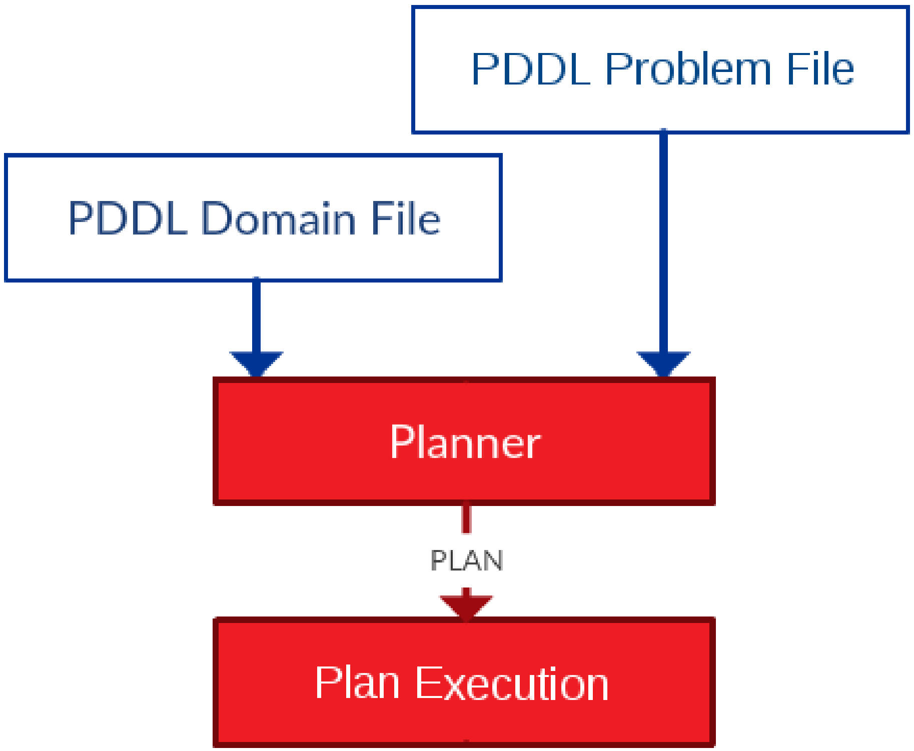
Baskaran, et el. Plan execution interchance language (PLEXIL) Version 1.0. *NASA Technical Memorandum,* 2007

Robertson et al. Autonomous Robust Execution of Complex Robotic Missions. *Proceedings of the 9th International Conference on Intelligent Autonomous Systems (IAS-9)*, 2006

Plan Execution 1: Very simple Dispatch

The most basic structure.

- The plan is generated.
- The plan is executed.



Plan Execution 1: Very simple Dispatch

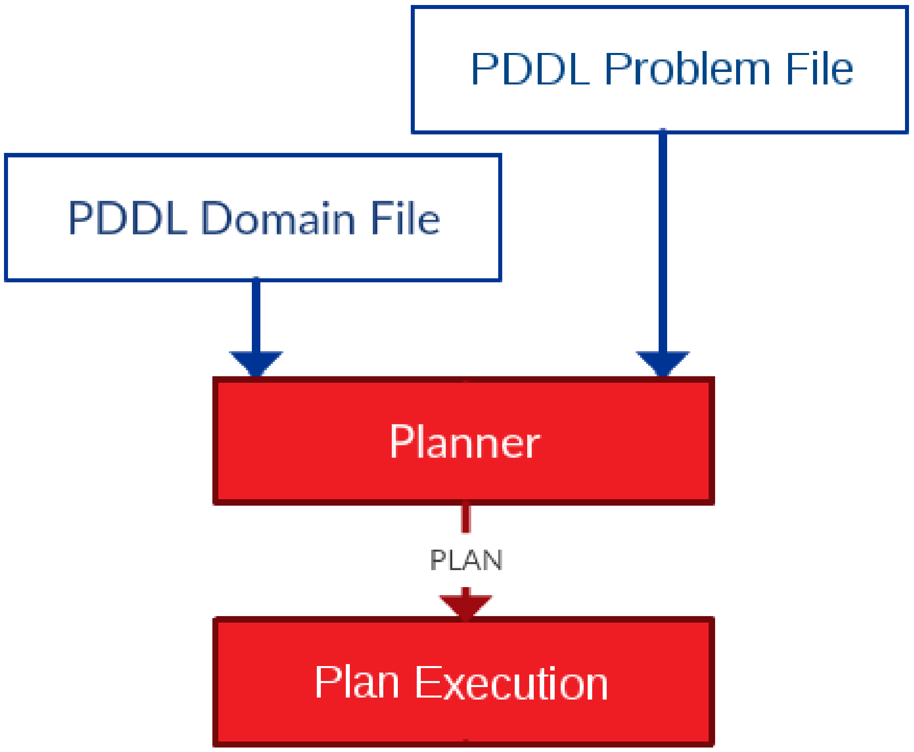
The most basic structure.

- The plan is generated.
- The plan is executed.

Red boxes are components of ROSPlan. They correspond to ROS nodes.

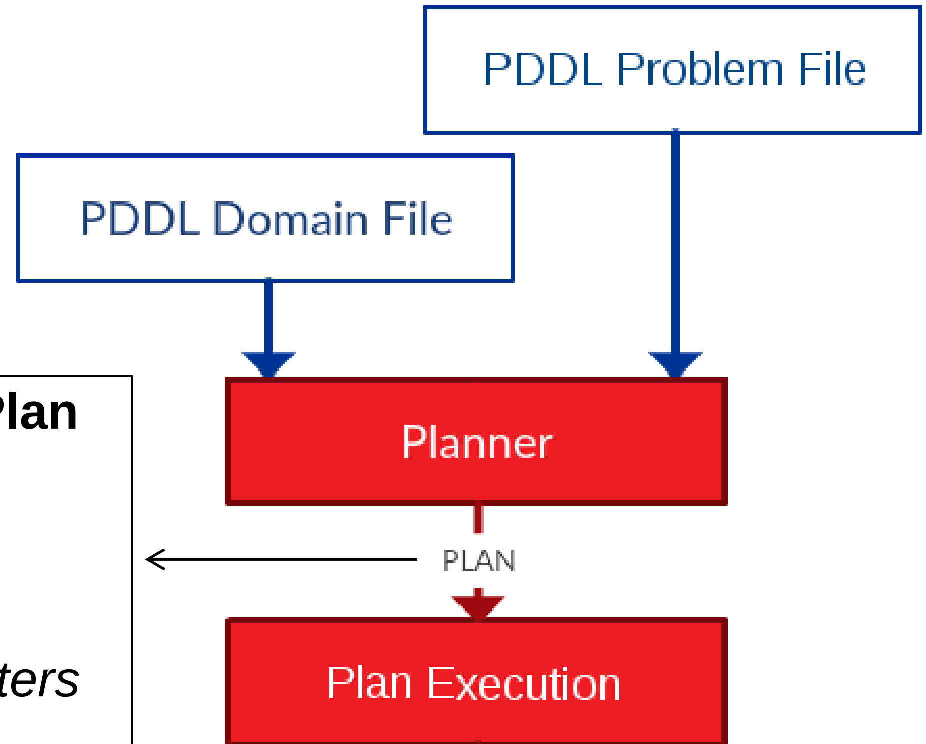
The domain and problem file can be supplied

- in launch parameters
- as ROS service parameters



Plan Execution 1: Very simple Dispatch

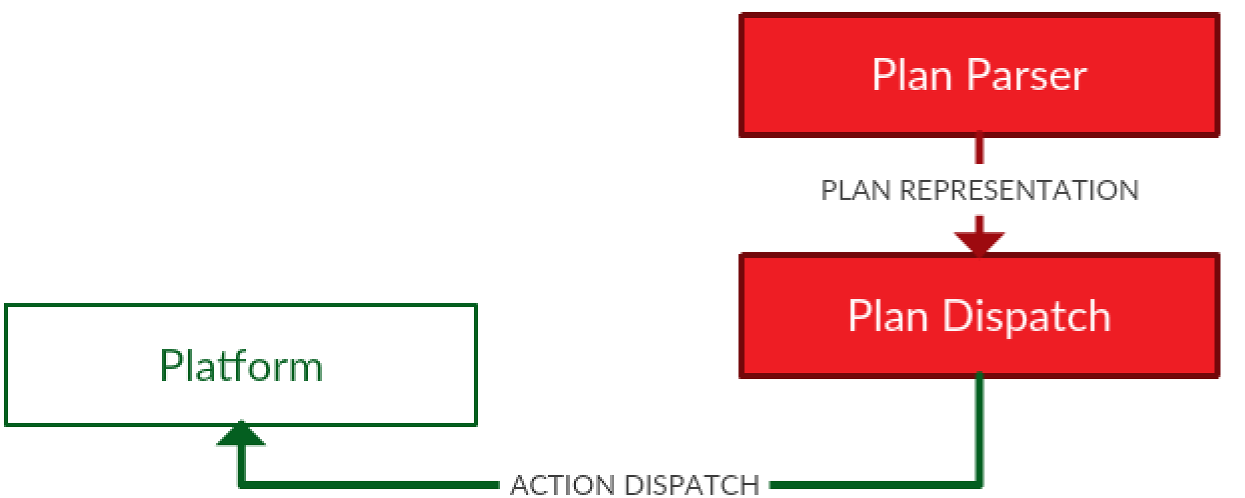
```
rosplan_dispatch_msgs/CompletePlan
ActionDispatch[] plan
  int32 action_id
  string name
  diagnostic_msgs/KeyValue[] parameters
    string key
    string value
  float32 duration
  float32 dispatch_time
```



A dispatch loop without feedback

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

- simple sequential execution
- timed execution
- Petri-Net plans
- Esterel Plans
- etc.

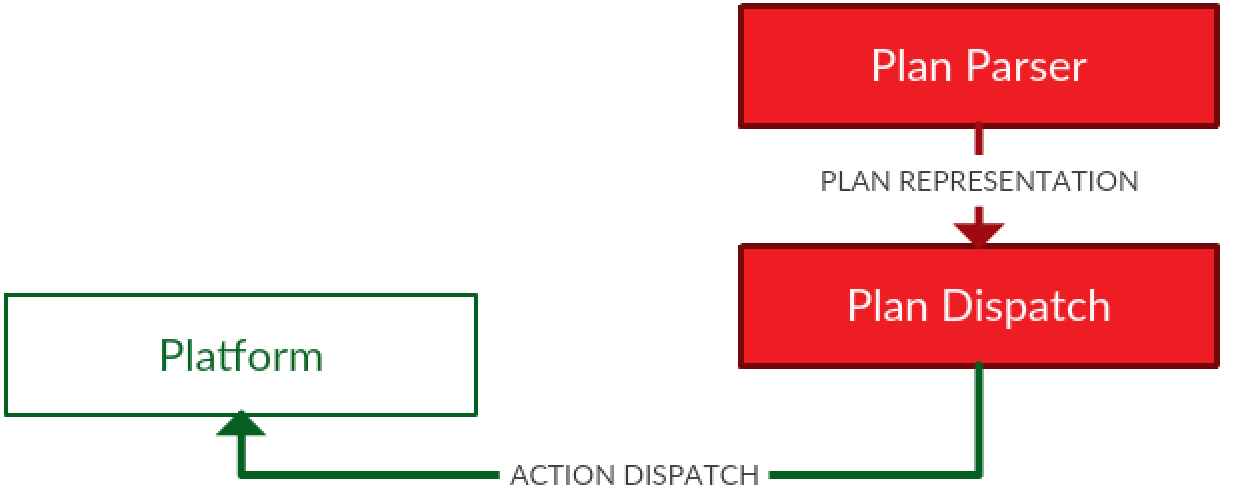


A dispatch loop without feedback

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.



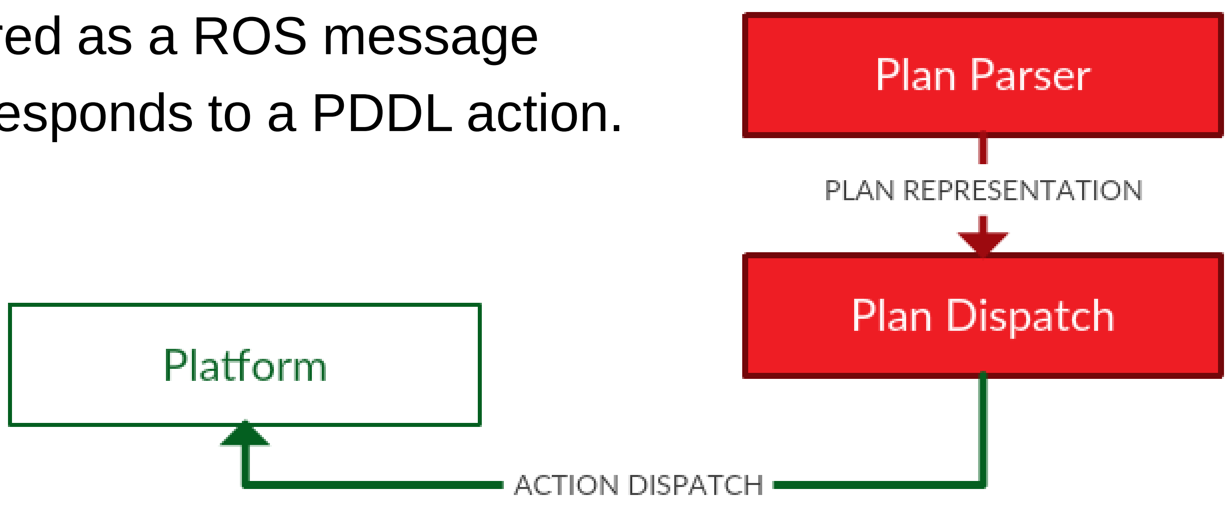
A dispatch loop without feedback

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.



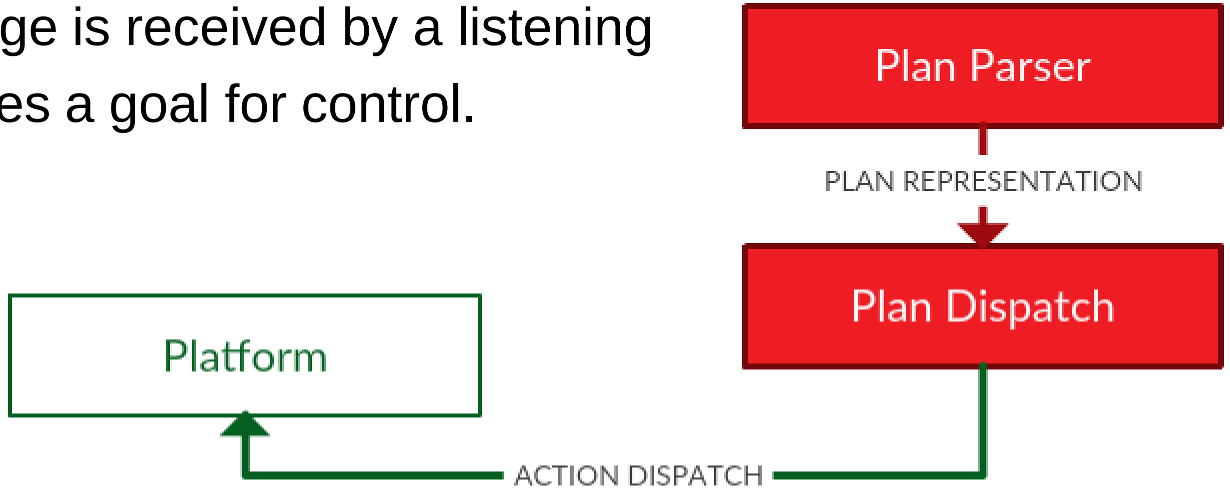
A dispatch loop without feedback

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.



A dispatch loop without feedback

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

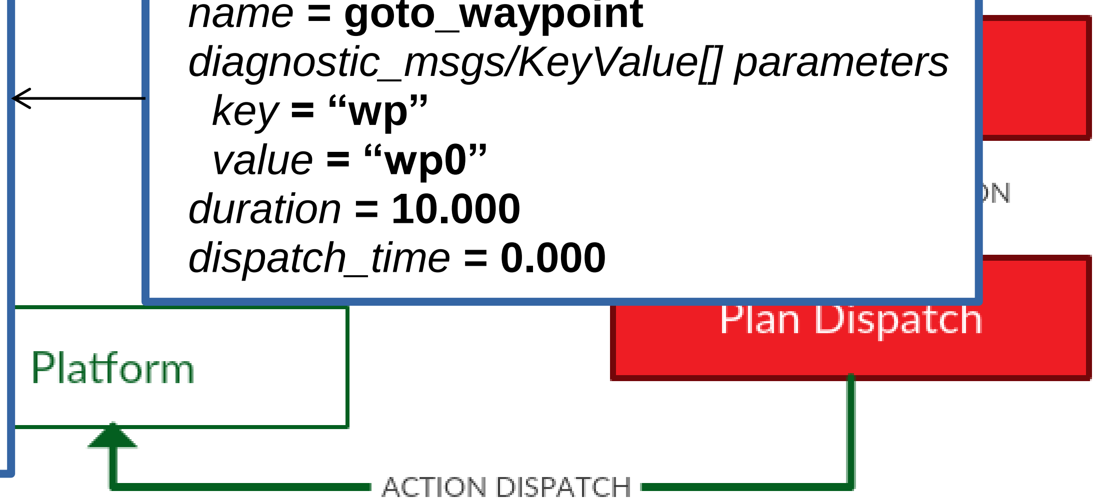
- simple sequential execution

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

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

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**



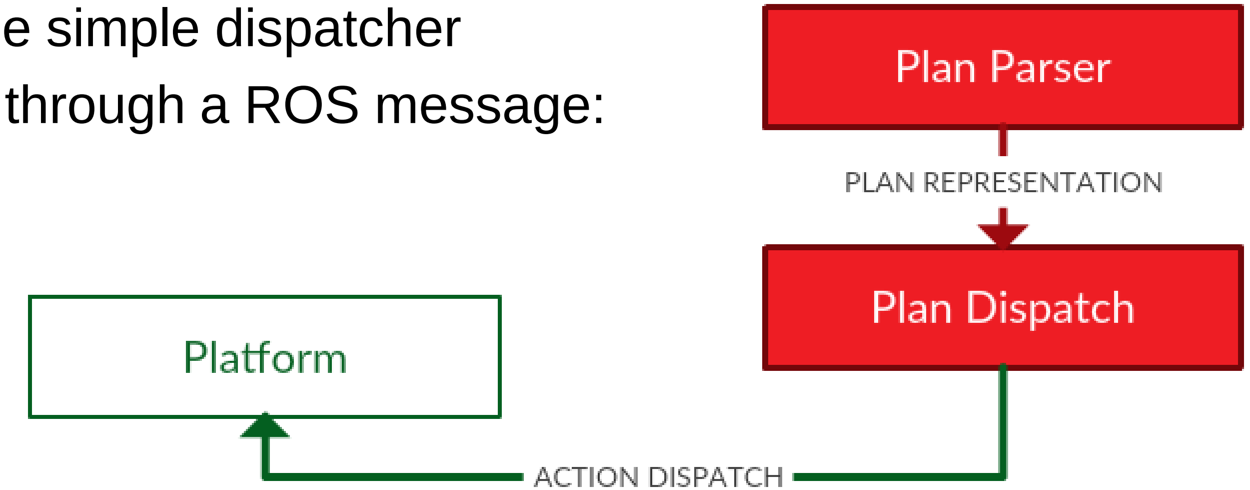
A dispatch loop without feedback

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.*

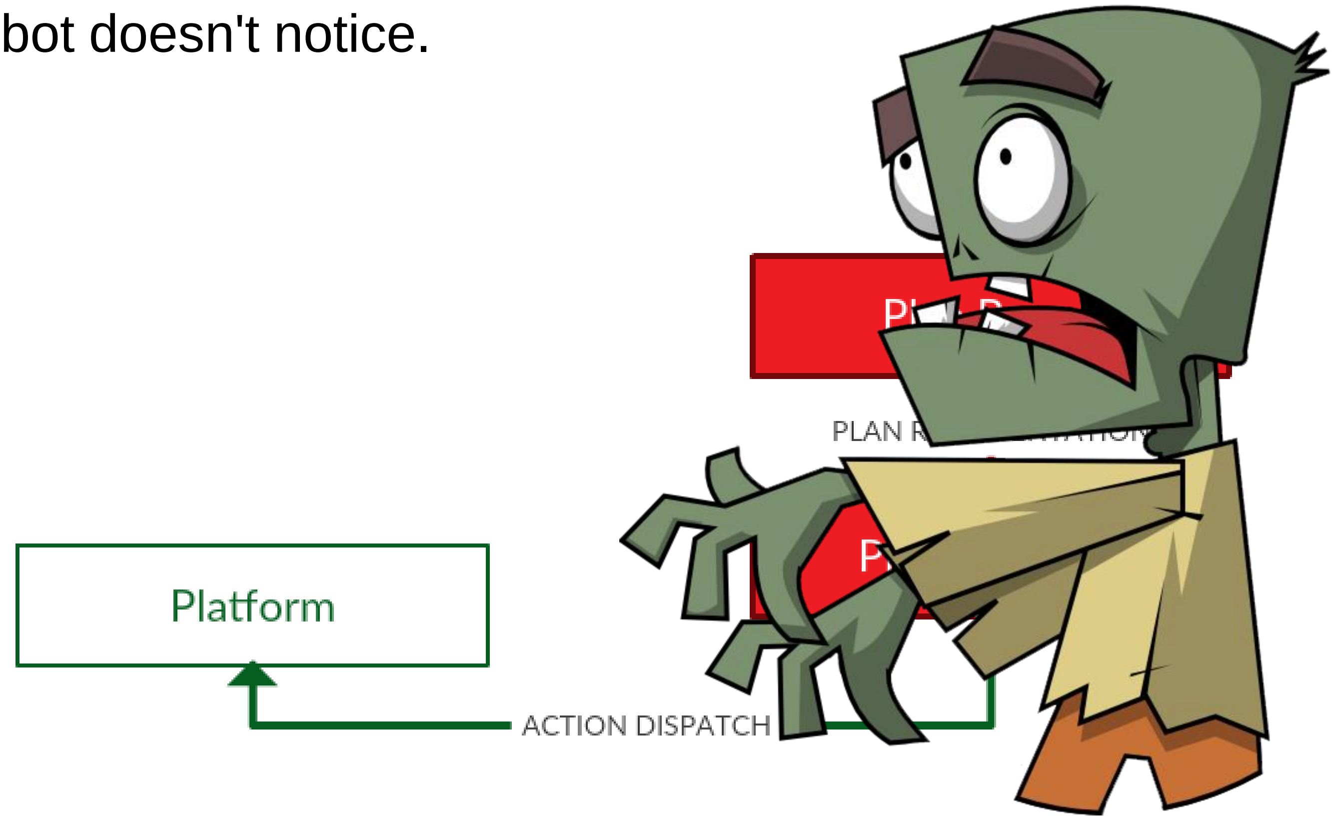


Plan Execution Failure

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.

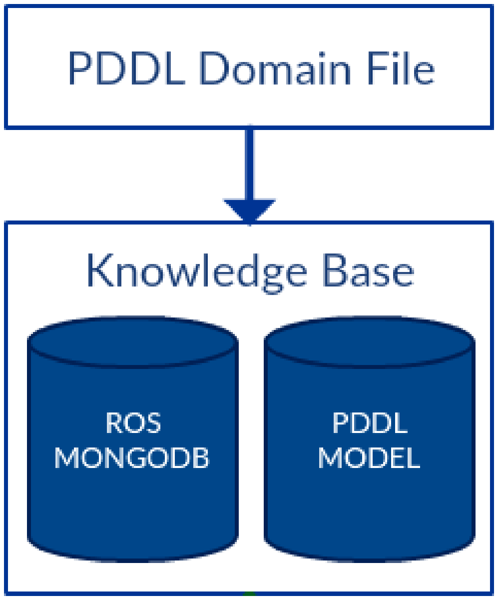
If we desire persistent autonomy, then the robot must be able to plan again, from the new current state, without human intervention.

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.

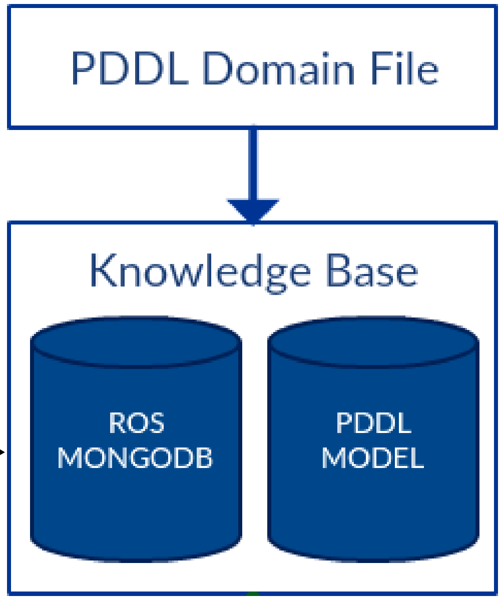


PDDL Model

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



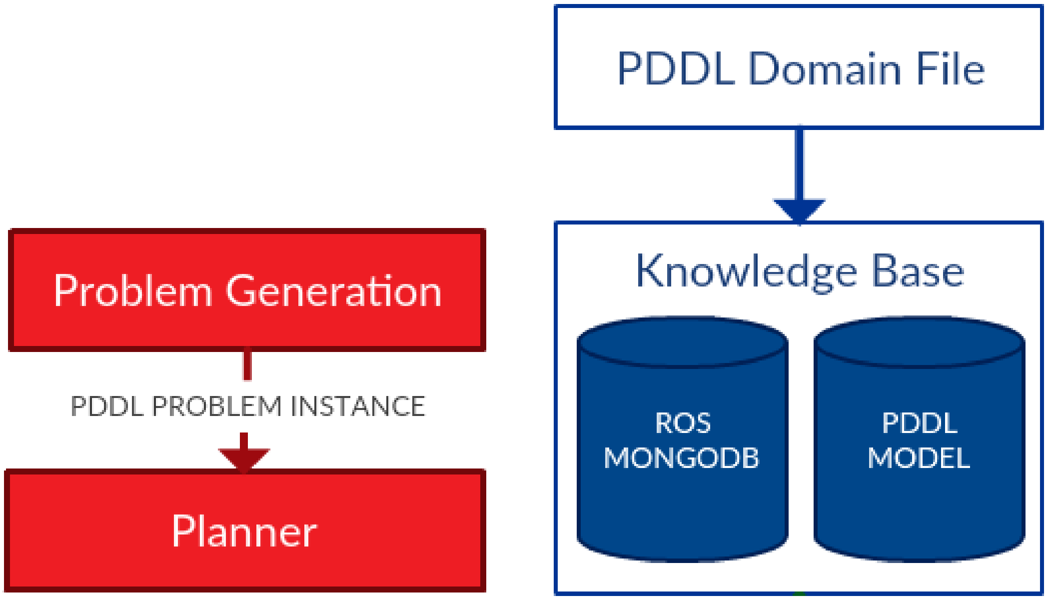
PDDL Model

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.

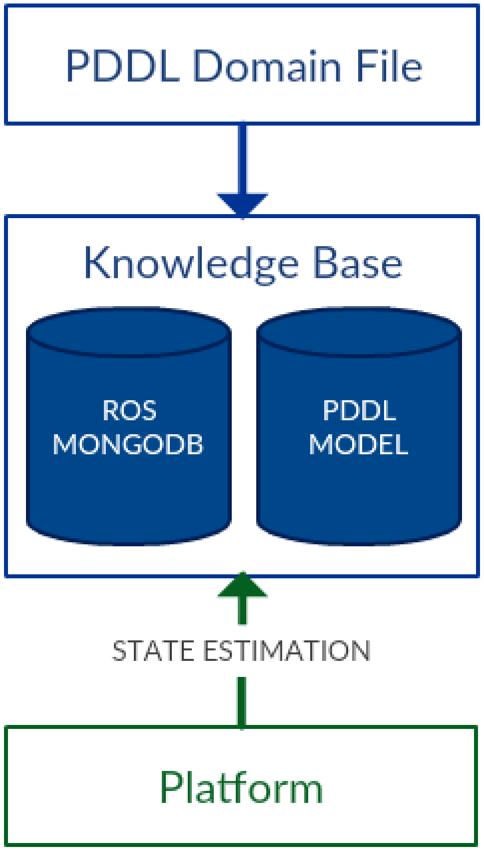


PDDL Model

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.

wp)

PDDL Domain File

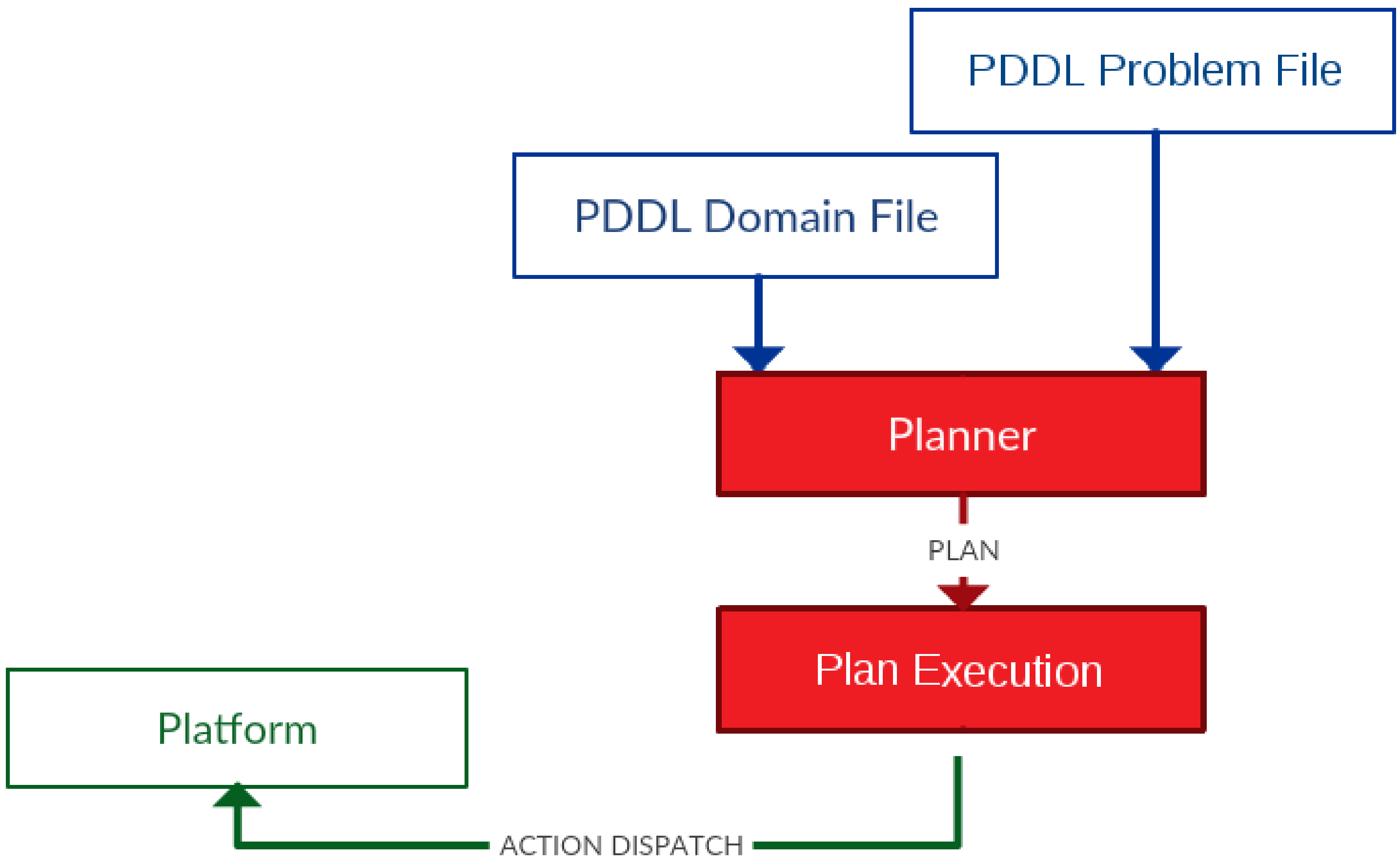
nav_msgs/Odometry

```
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
```

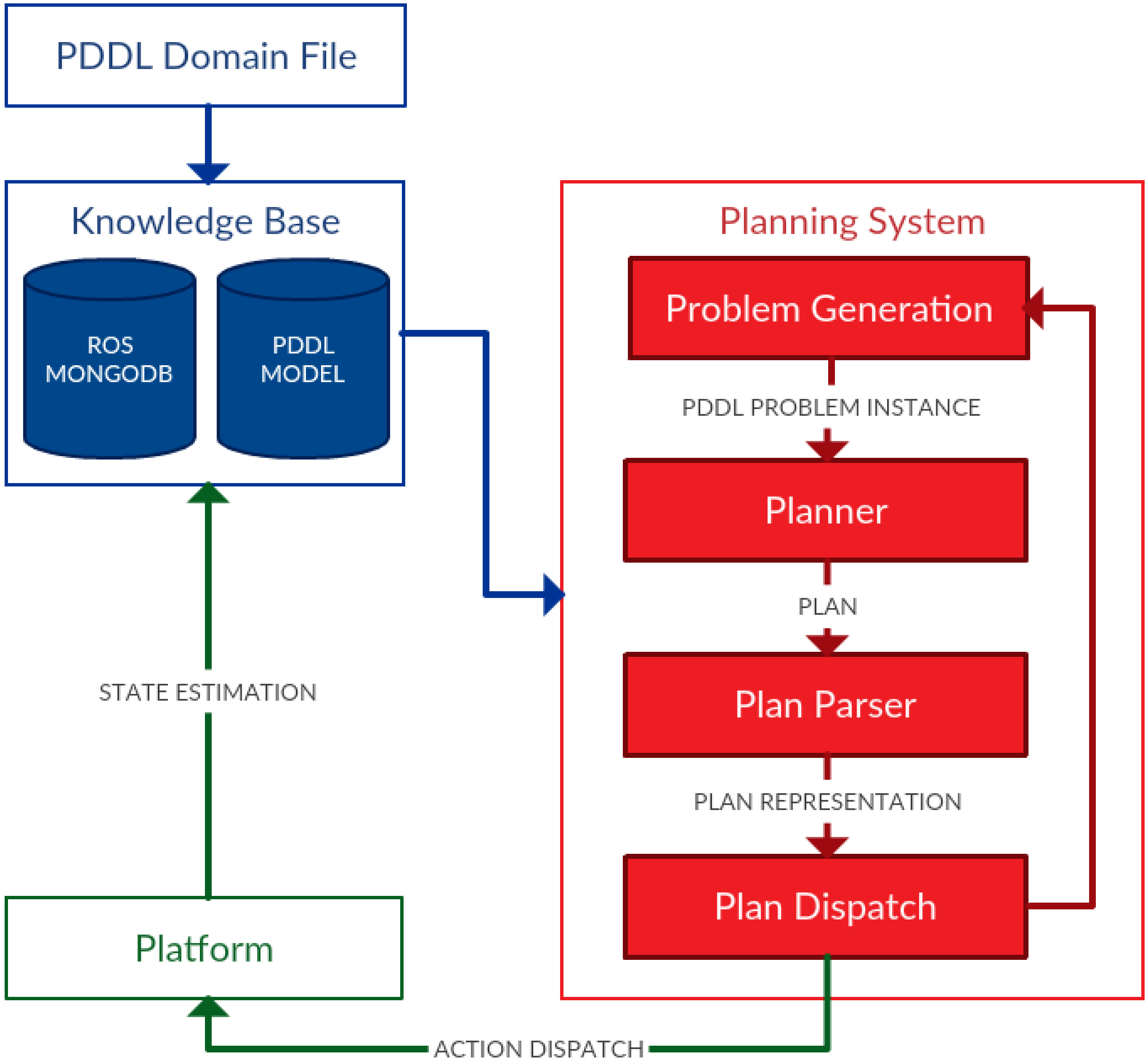
rosplan_knowledge_msgs/KnowledgeItem

```
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
```

ROSPlan components



ROSPlan components



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.

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.

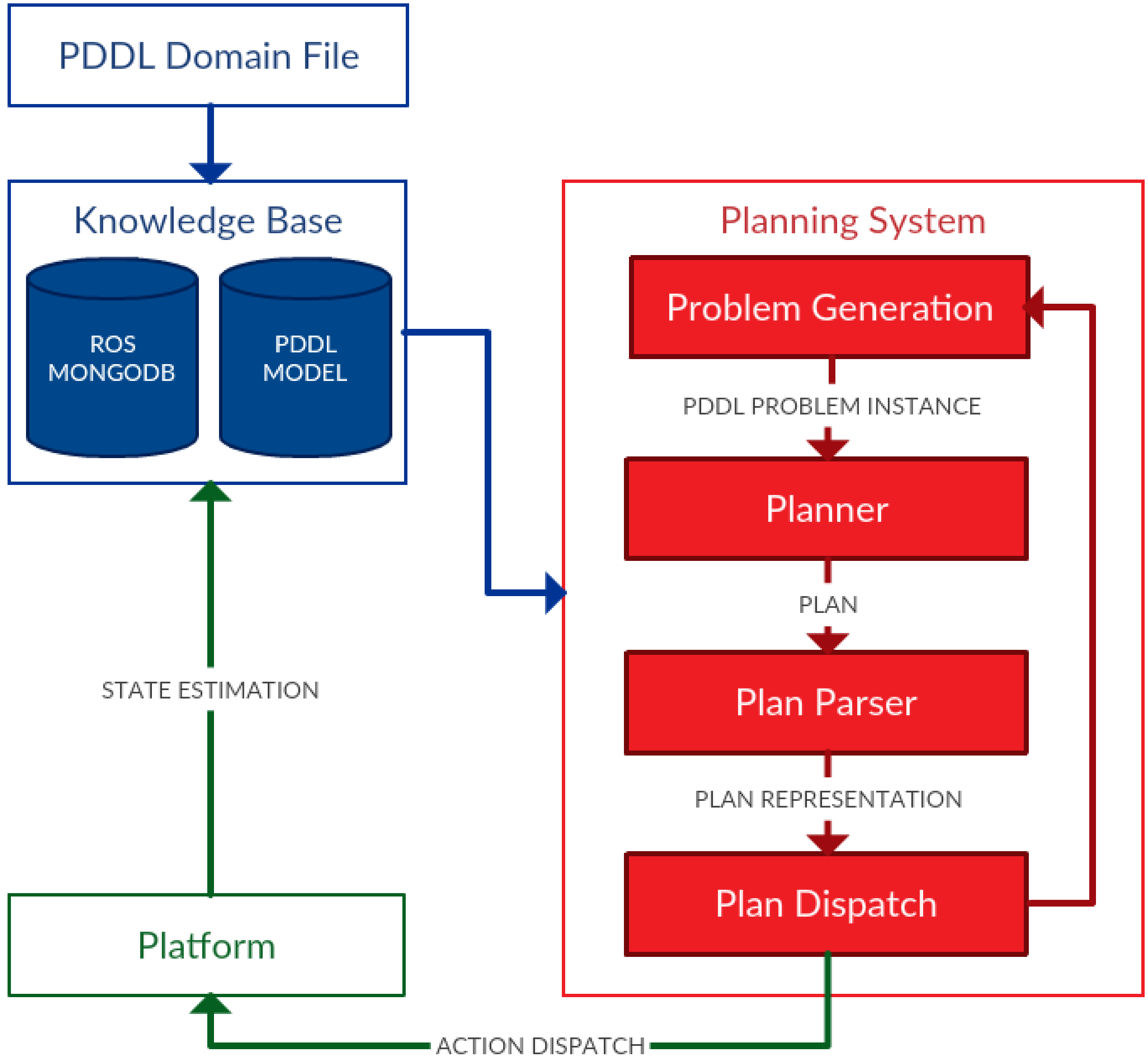
If the plan will fail in the future, the robot should not continue to execute the plan for a long time without purpose.

The success or failure of an action can sometimes not be understood outside of the context of the whole plan.

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.



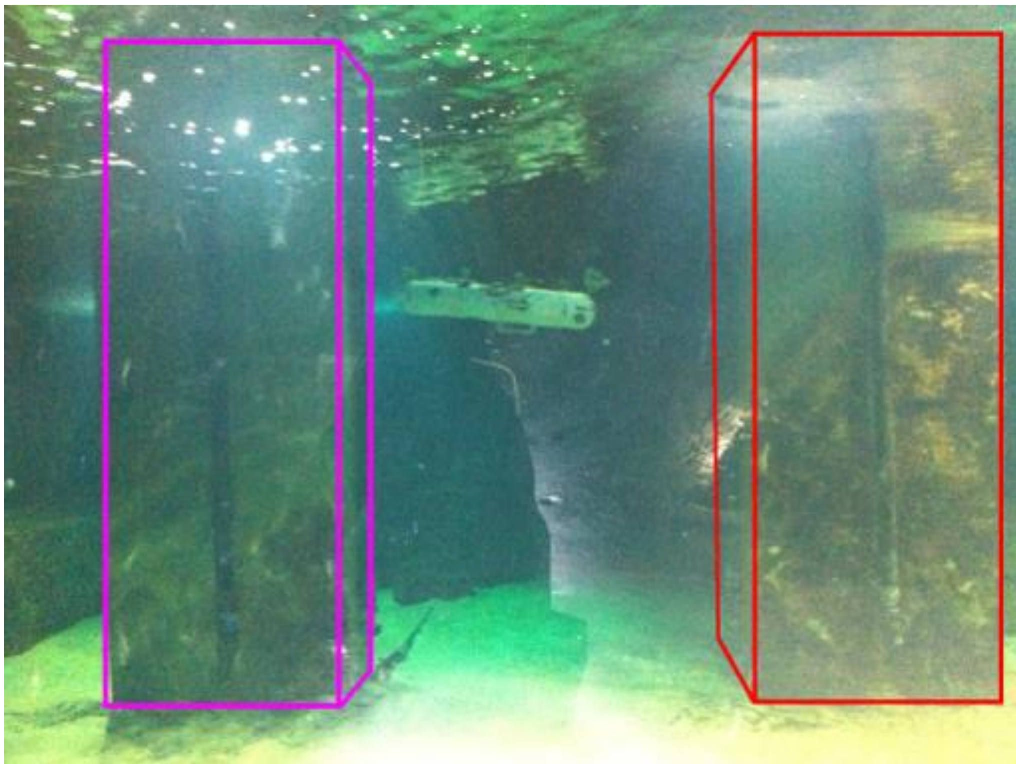
Bad Behaviour 2: Plan Failure

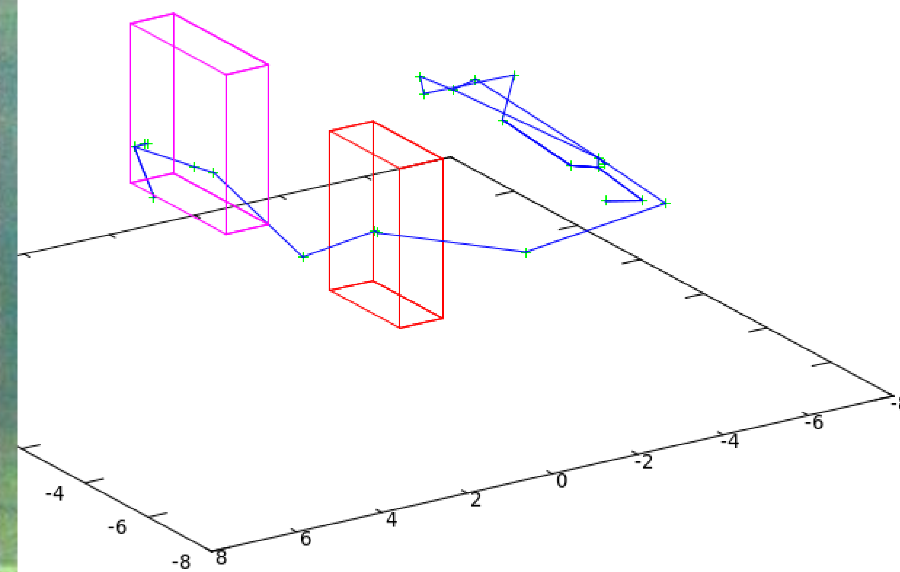
The AUV plans for inspection missions, recording images of pipes and welds.

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.



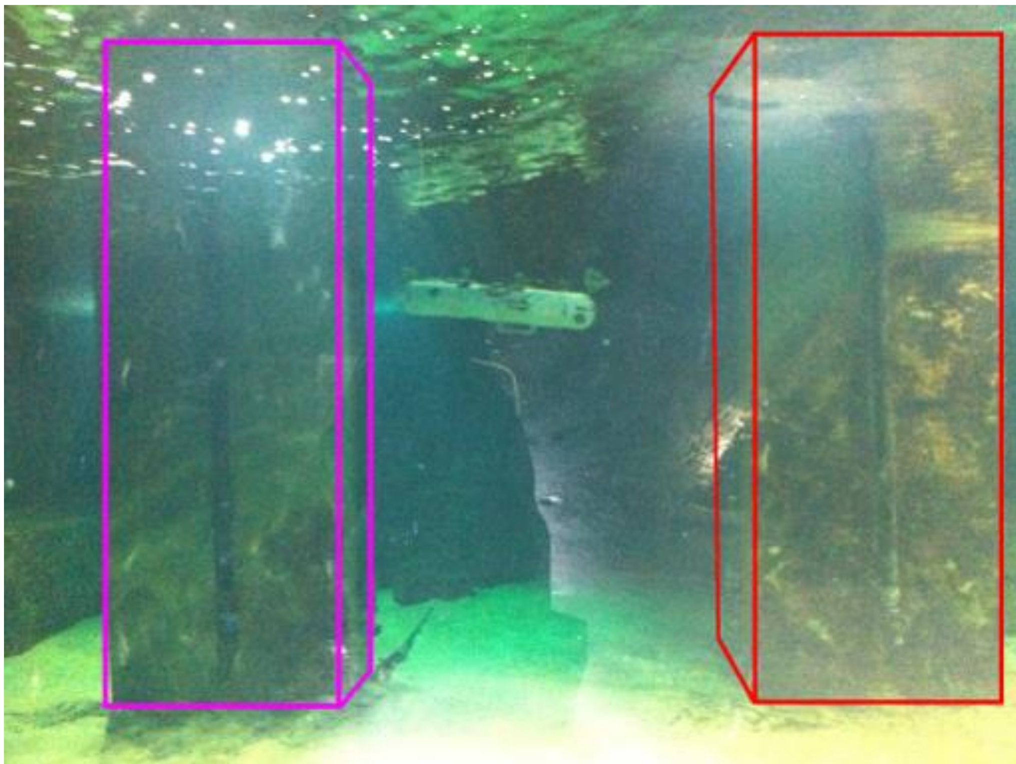


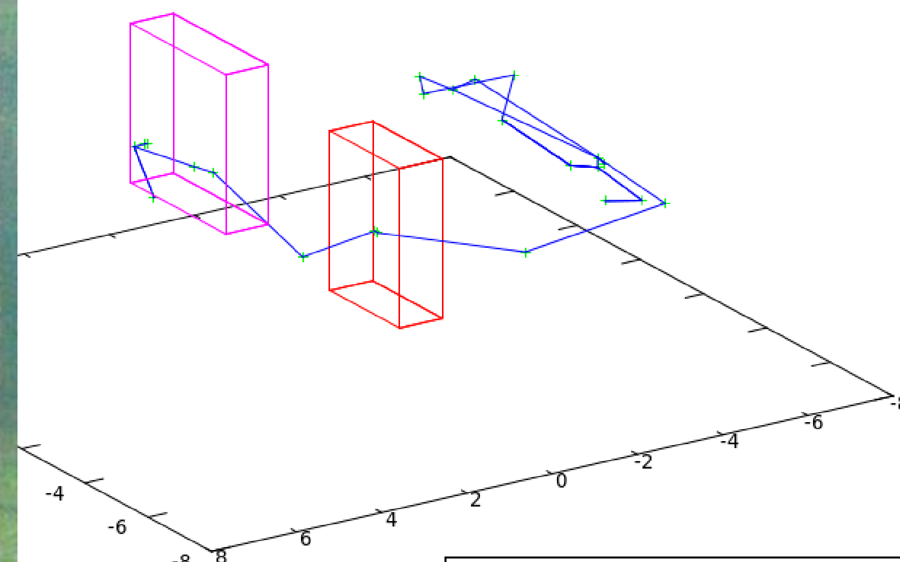
The planned inspection path is shown on the right. The AUV will move around to the other side of the pillars before inspecting the pipes on their facing sides.

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.





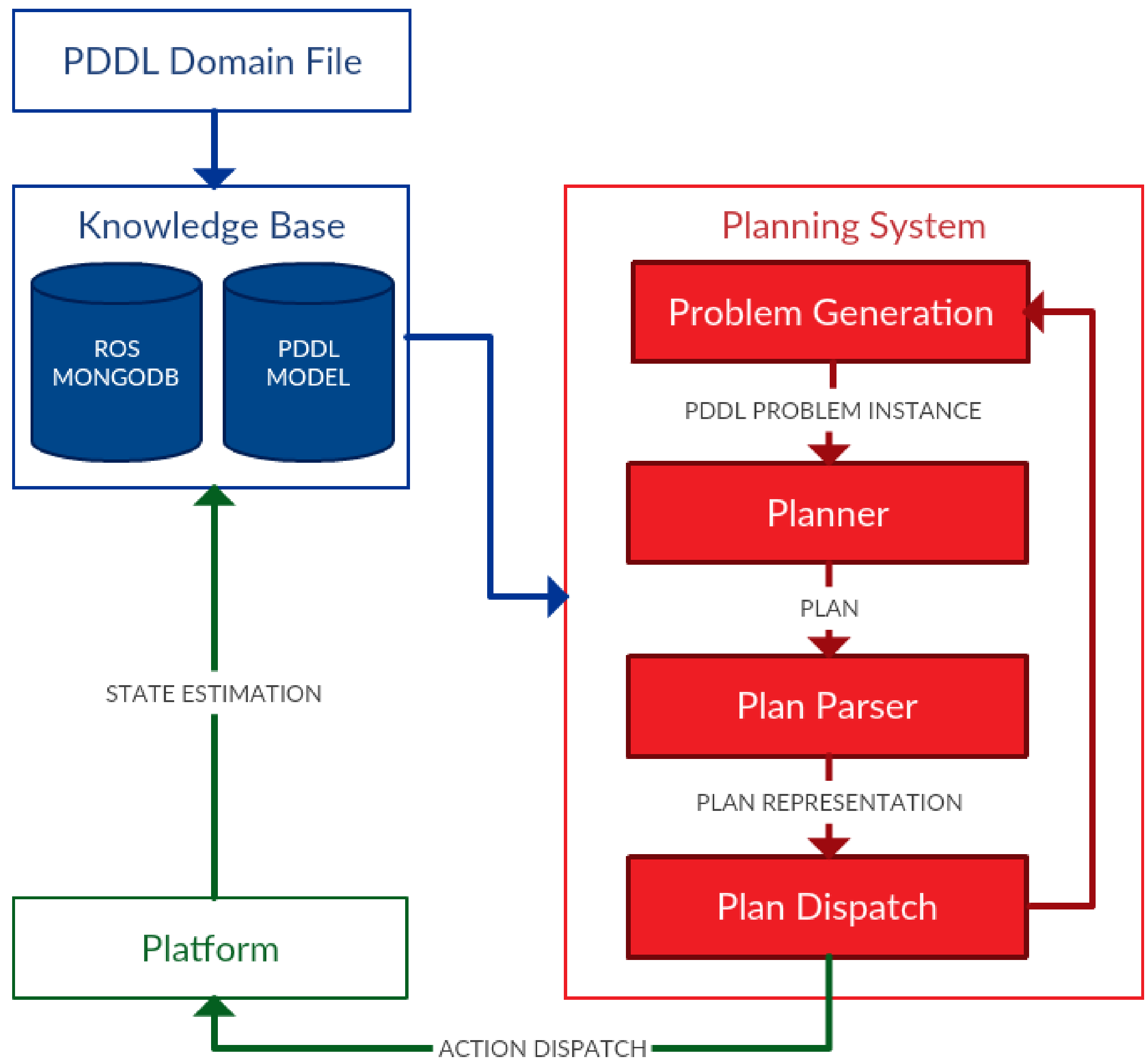
ROSPlan validates using VAL. [Fox et al. 2005]

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

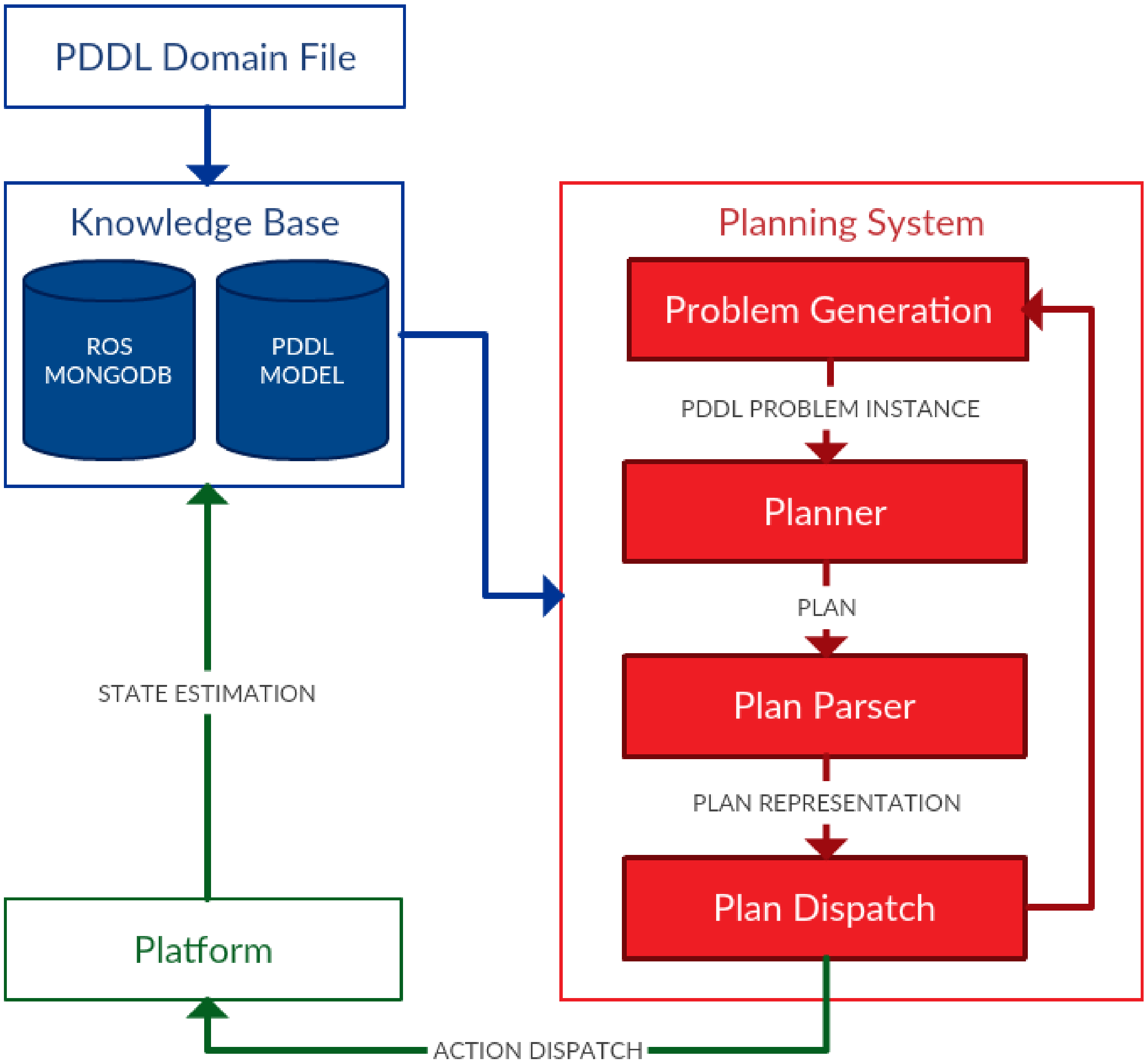
Now the system is more complex:

- PDDL model is continuously updated from sensor data.

- problem file is automatically generated.

- the planner generates a plan.

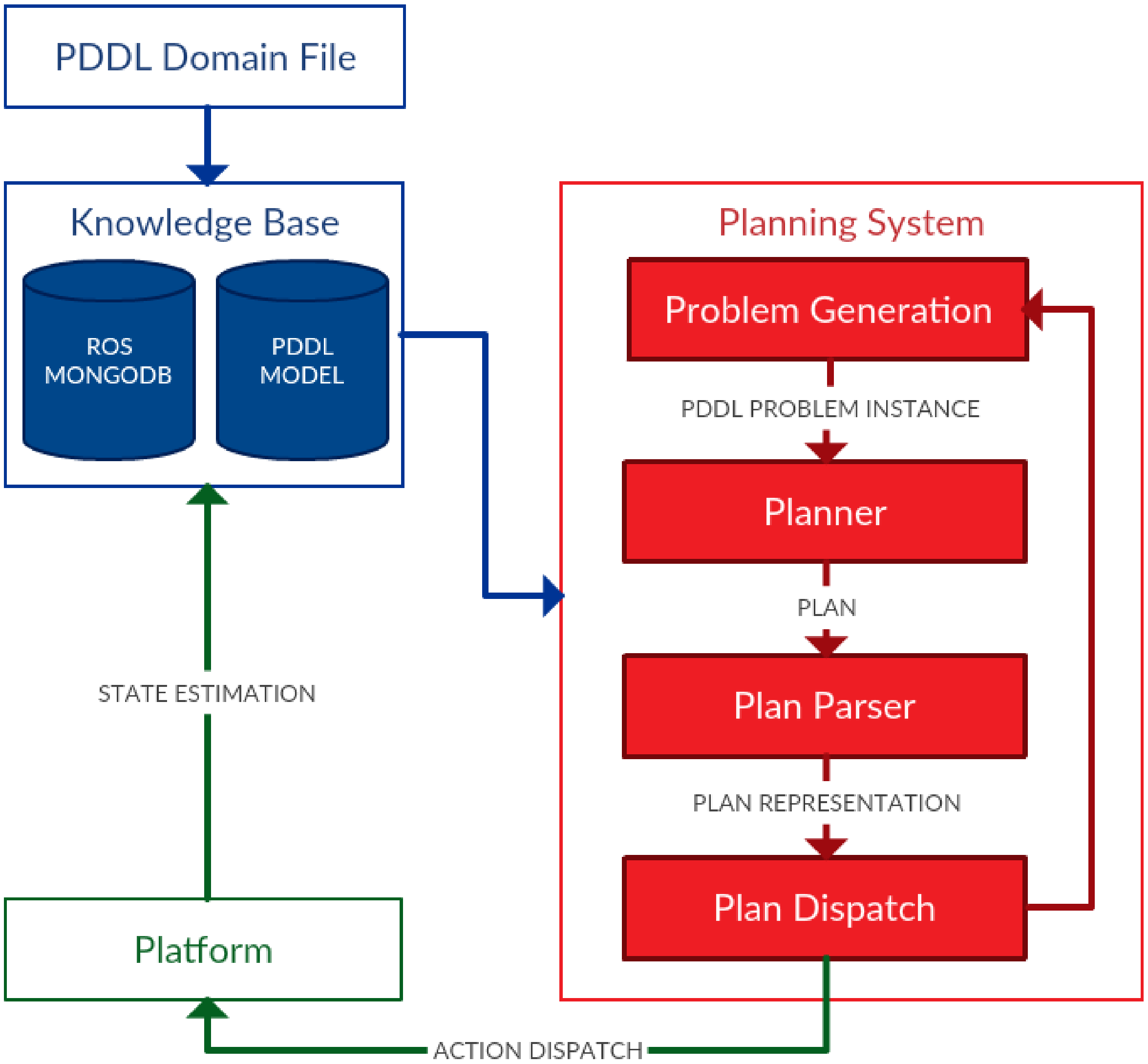
- the plan is dispatched action-by-action.



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.

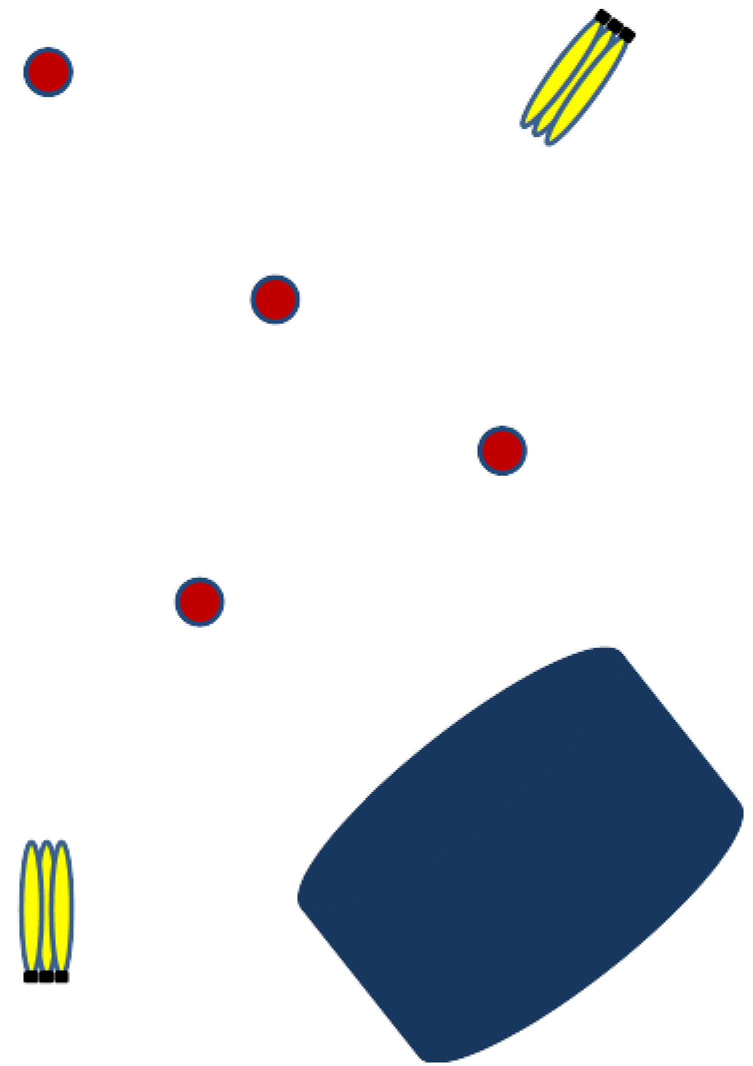


Plan Execution 2: Very Simple Temporal Dispatch

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?

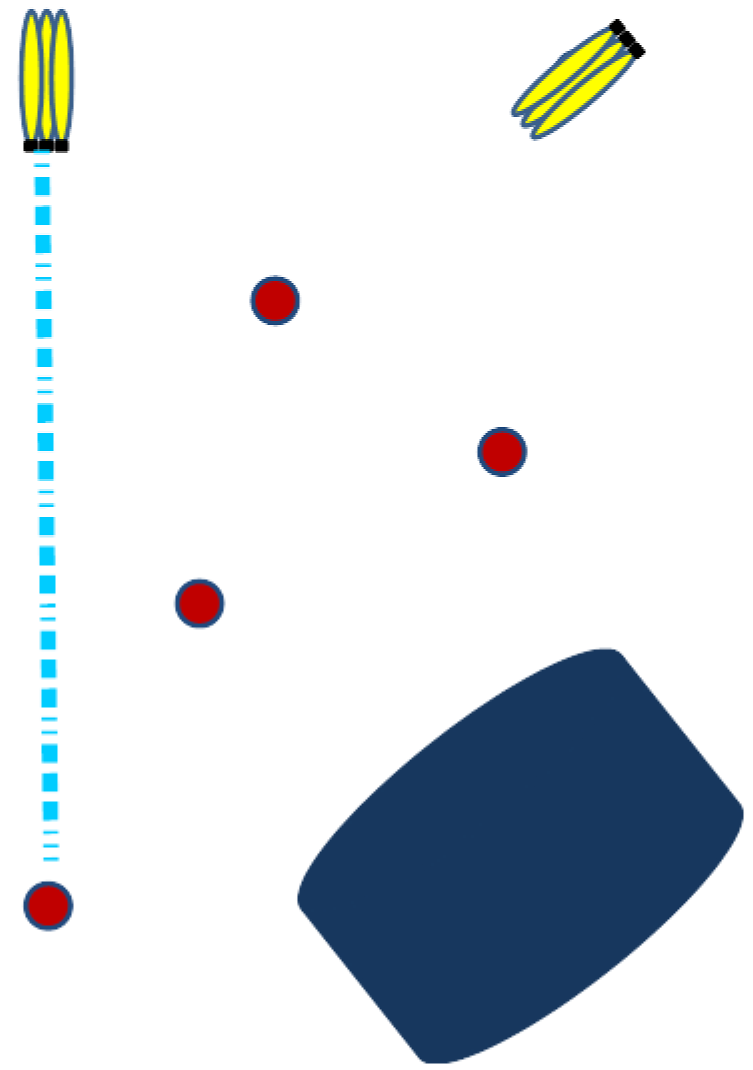


Plan Execution 2: Very Simple Temporal Dispatch

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?

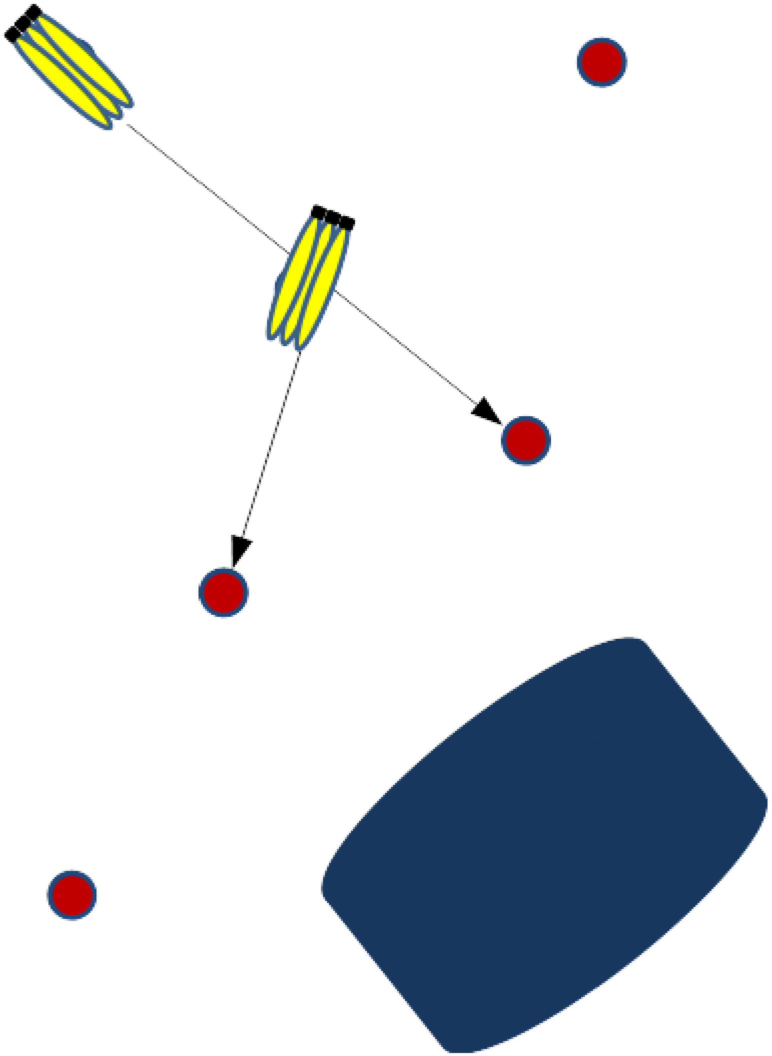


Plan Execution 2: Very Simple Temporal Dispatch

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?

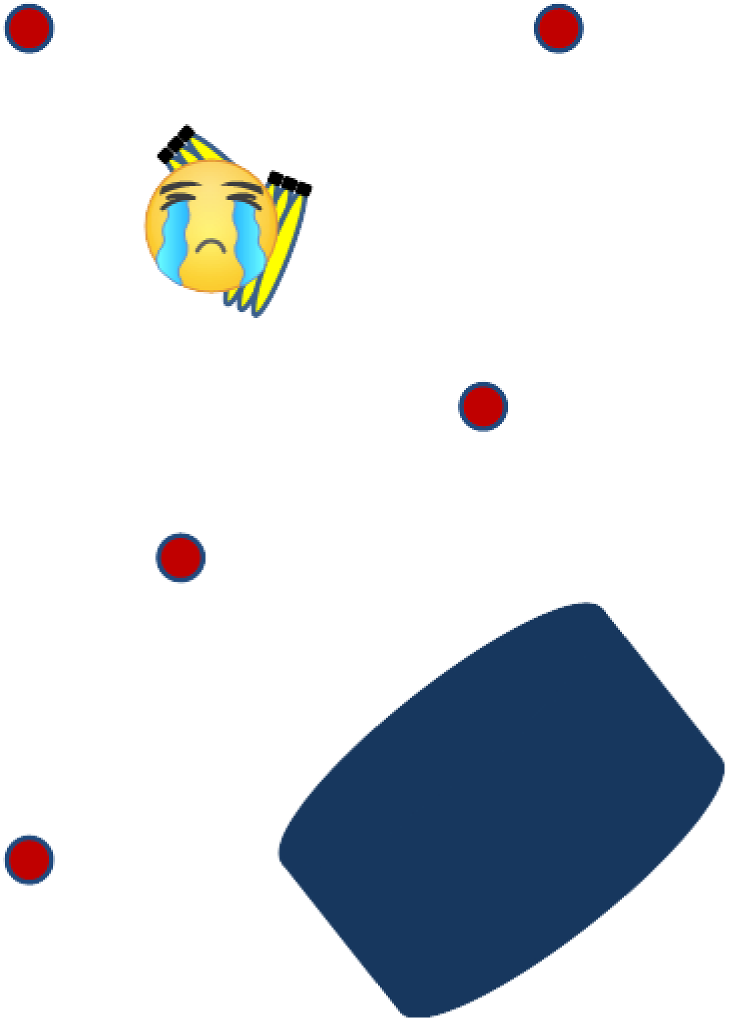


Plan Execution 2: Very Simple Temporal Dispatch

The real world requires a temporal and numeric model:

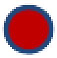
- time and deadlines,
- battery power and consumption,
- direction of sea current, or traffic flow.

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



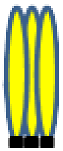
Temporal Constraints

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



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

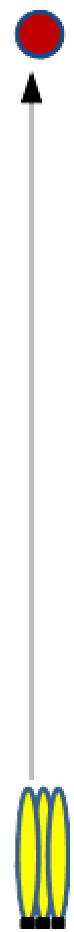
However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.



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

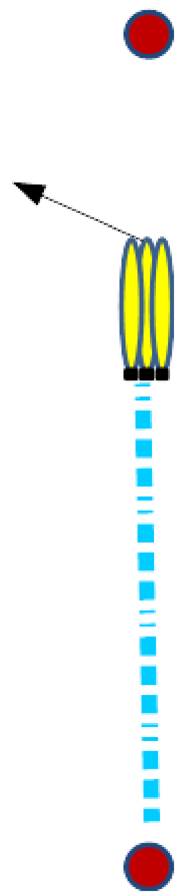
The plan execution loop could instead dispatch actions at their *estimated* timestamps.

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.



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

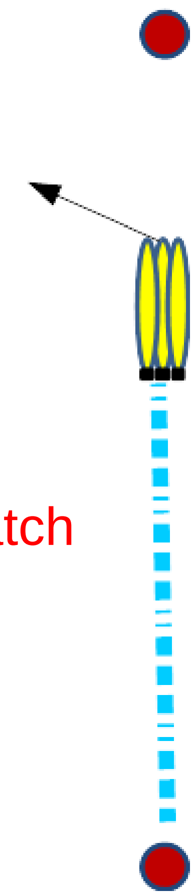
However, in the real world there are many uncontrollable durations and events. The estimated duration of actions is rarely accurate.

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]

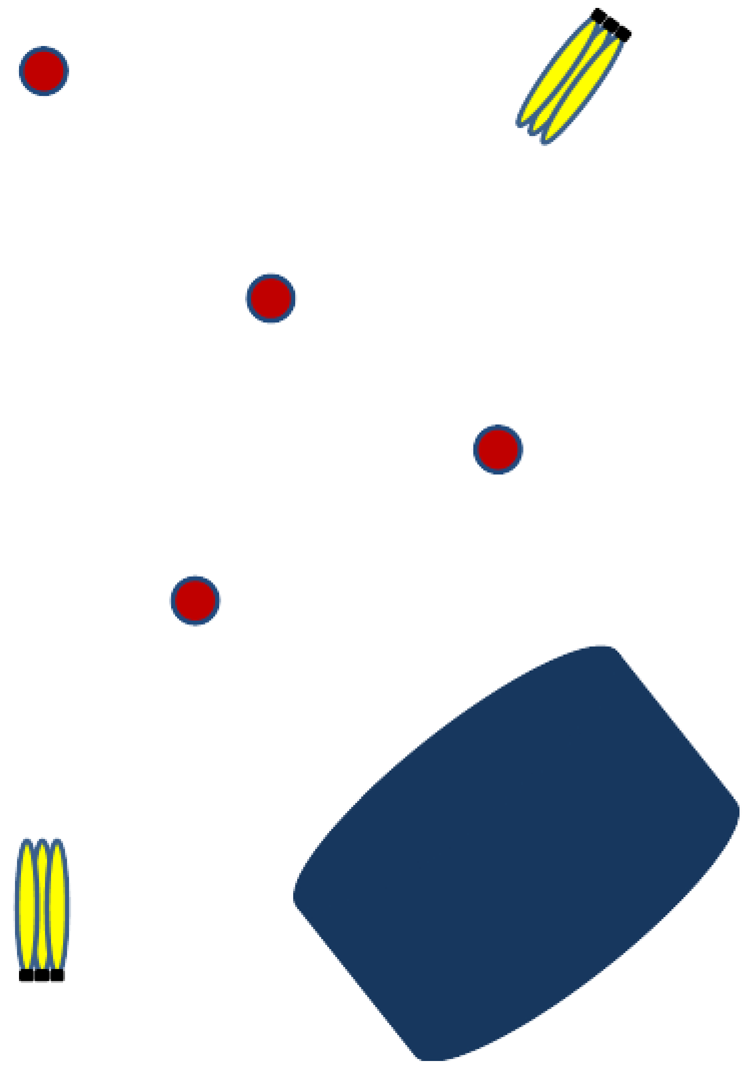
Temporal Constraints

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.



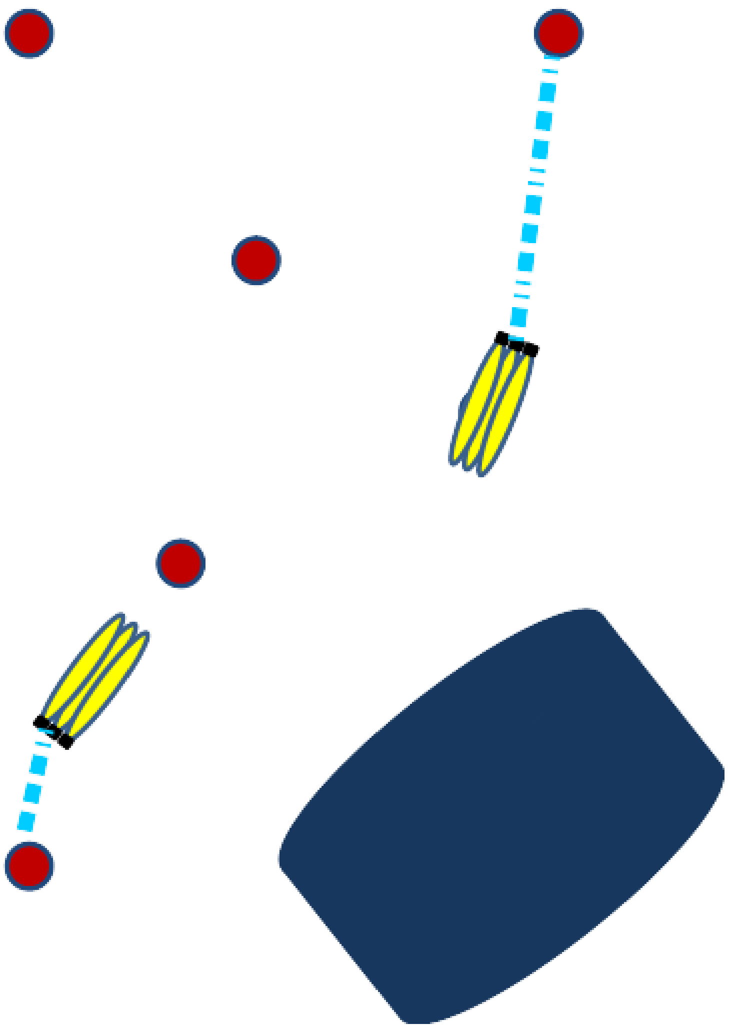
Temporal Constraints

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.



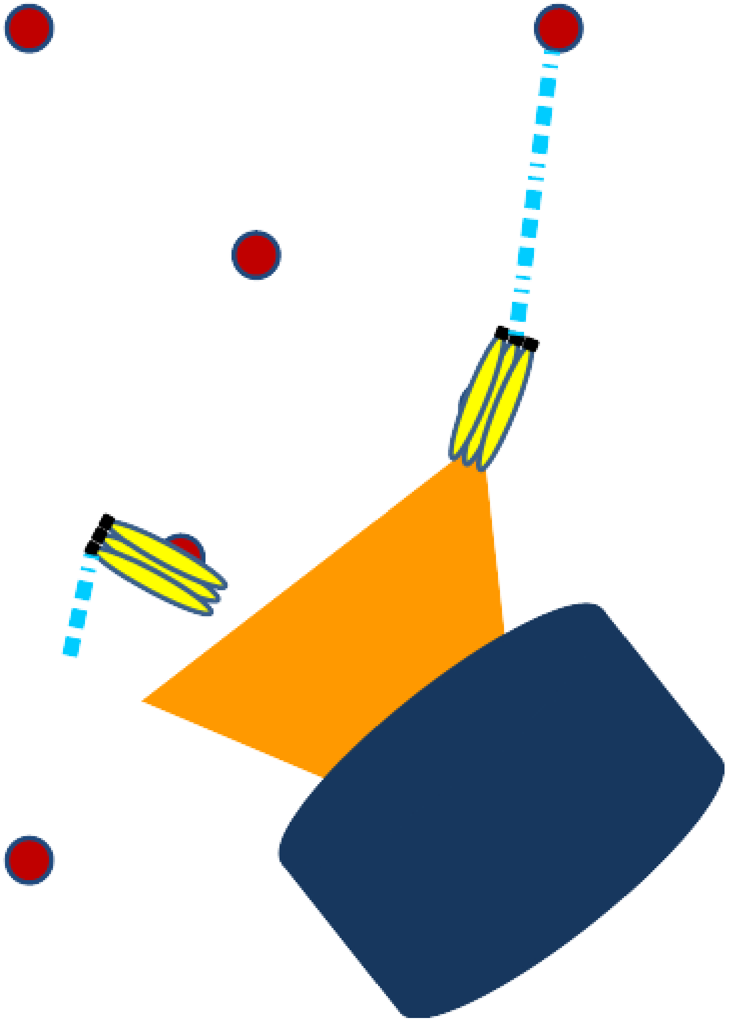
Temporal Constraints

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.



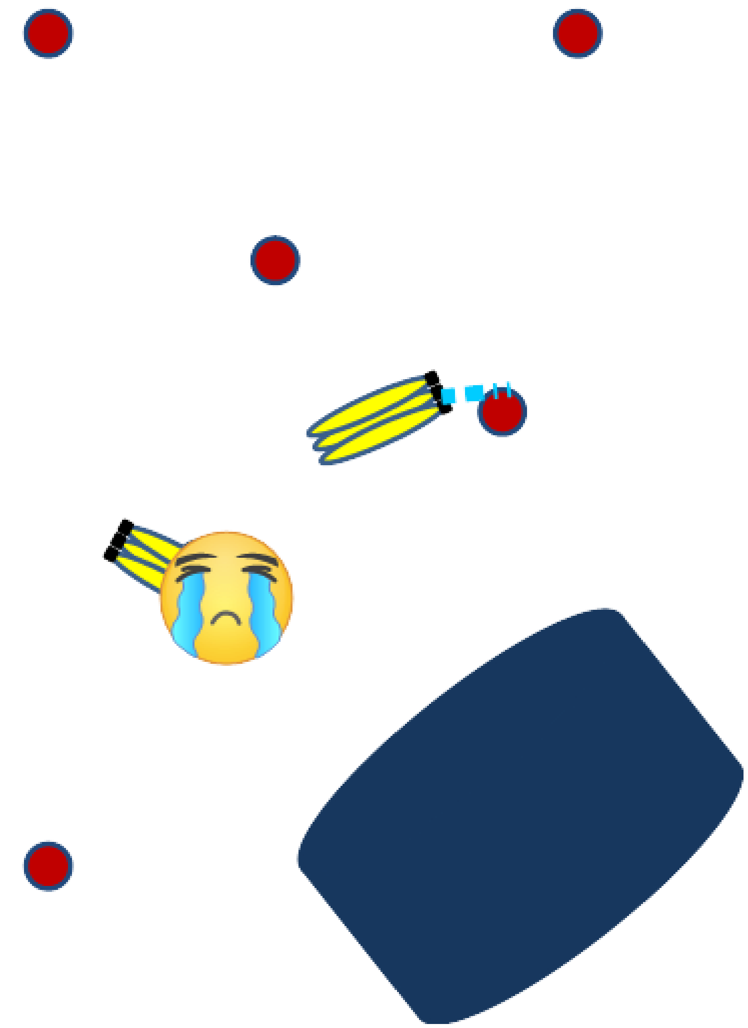
Temporal Constraints

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.

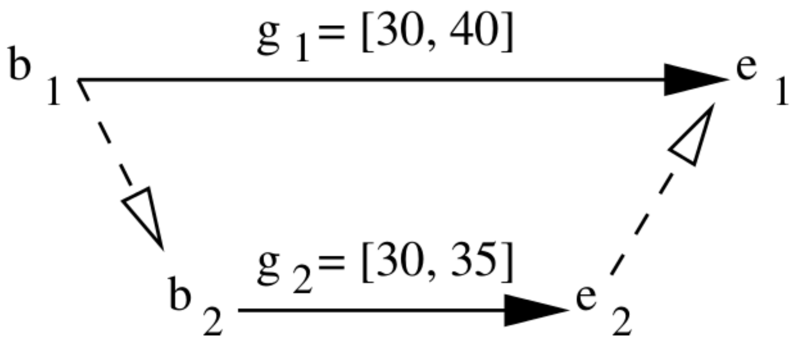


STPUs: Strong controllability

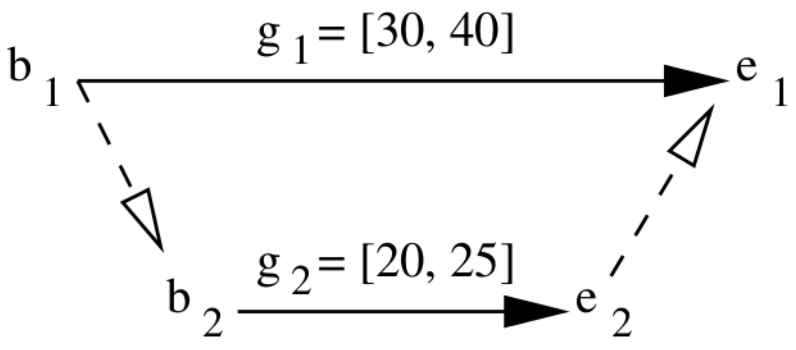
An STPU is strongly controllable iff:

- the agent can commit (in advance) to a time for all activated time-points,

- for any possible time for received time points, the temporal constraints are not violated.



(a)

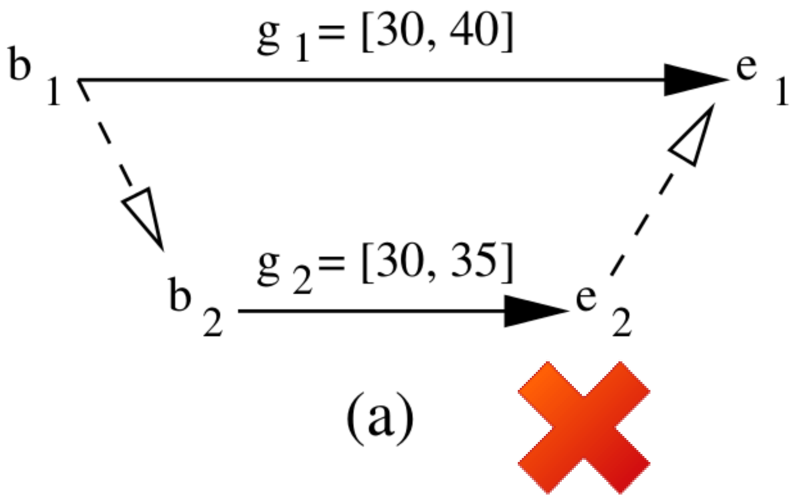


(b)

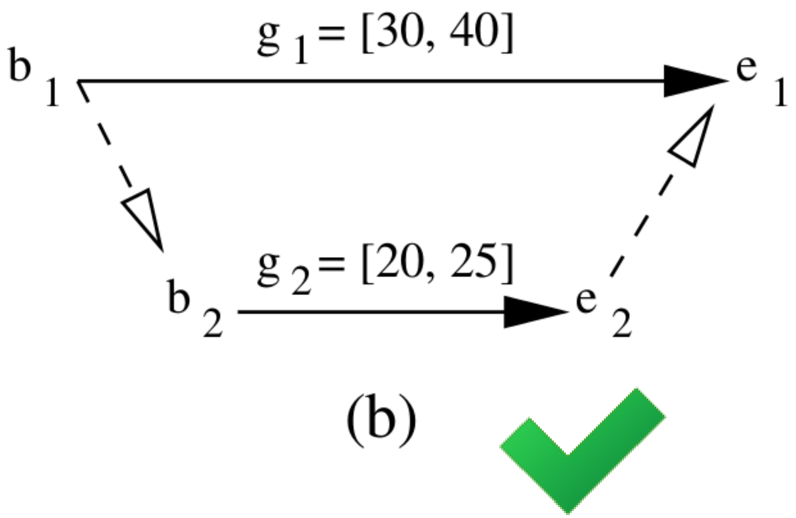
STPUs: Strong controllability

An STPU is strongly controllable iff:

- the agent can commit (in advance) to a time for all activated time-points,
- for any possible time for received time points, the temporal constraints are not violated.



(a)



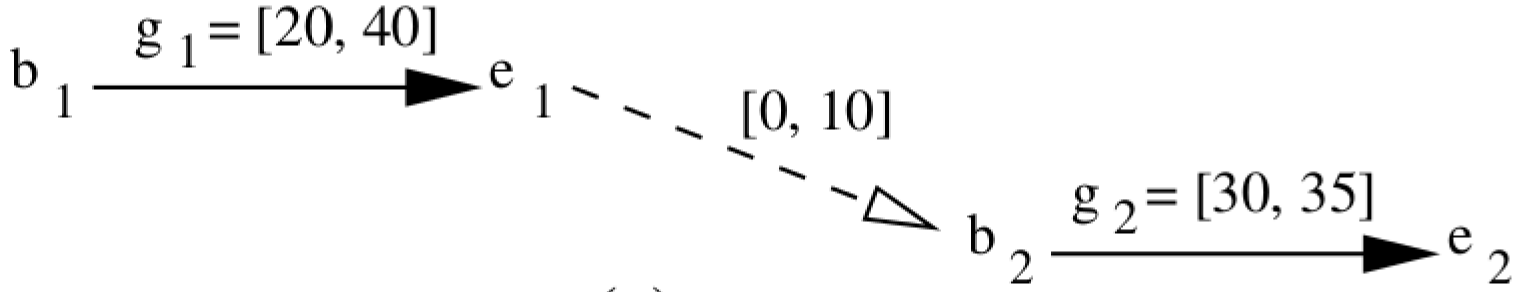
(b)

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

STPUs: Strong controllability

An STPU is strongly controllable iff:

- the agent can commit (in advance) to a time for all activated time-points,
- 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.
It is dynamically controllable.

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.

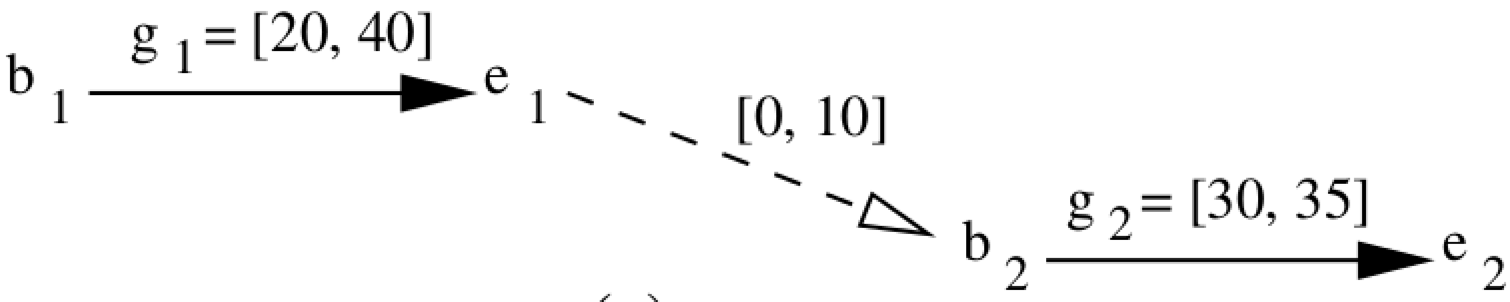
In this case, the agent does not have to commit to a time for any activated time points in advance.

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)

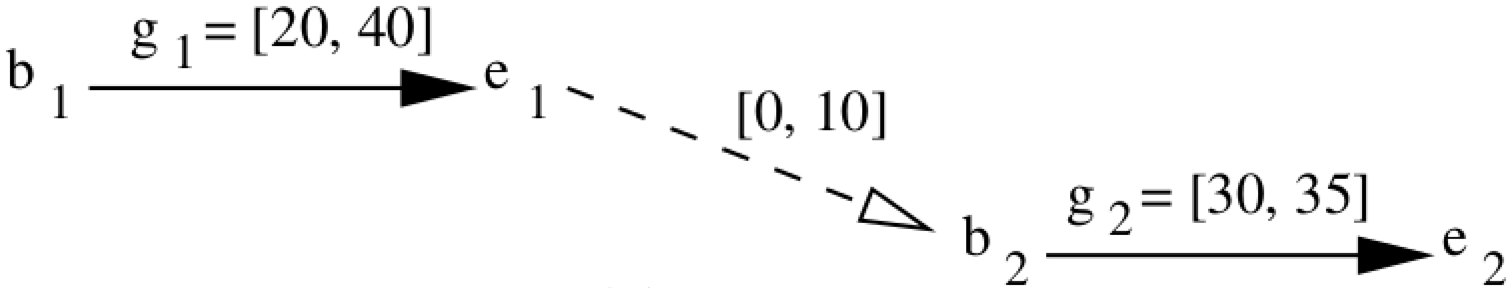
STPUs: Dynamic controllability

Not all problems will have solutions have any kind of controllability.

This does not mean they are impossible to plan or execute.

To reason about these kinds of issues we need to use a plan representation sufficient to capture

- the difference between controllable and uncontrollable durations,
- causal orderings, and
- temporal constraints.



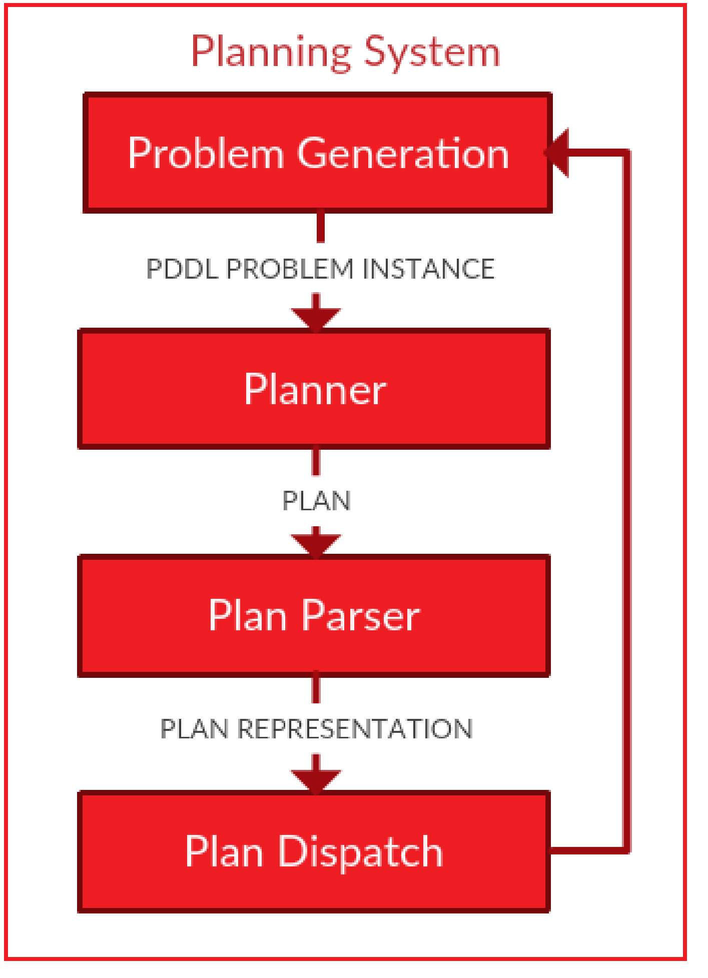
(c)

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.
- Humans are unpredictable.
- The environment is often initially full of unknowns.

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

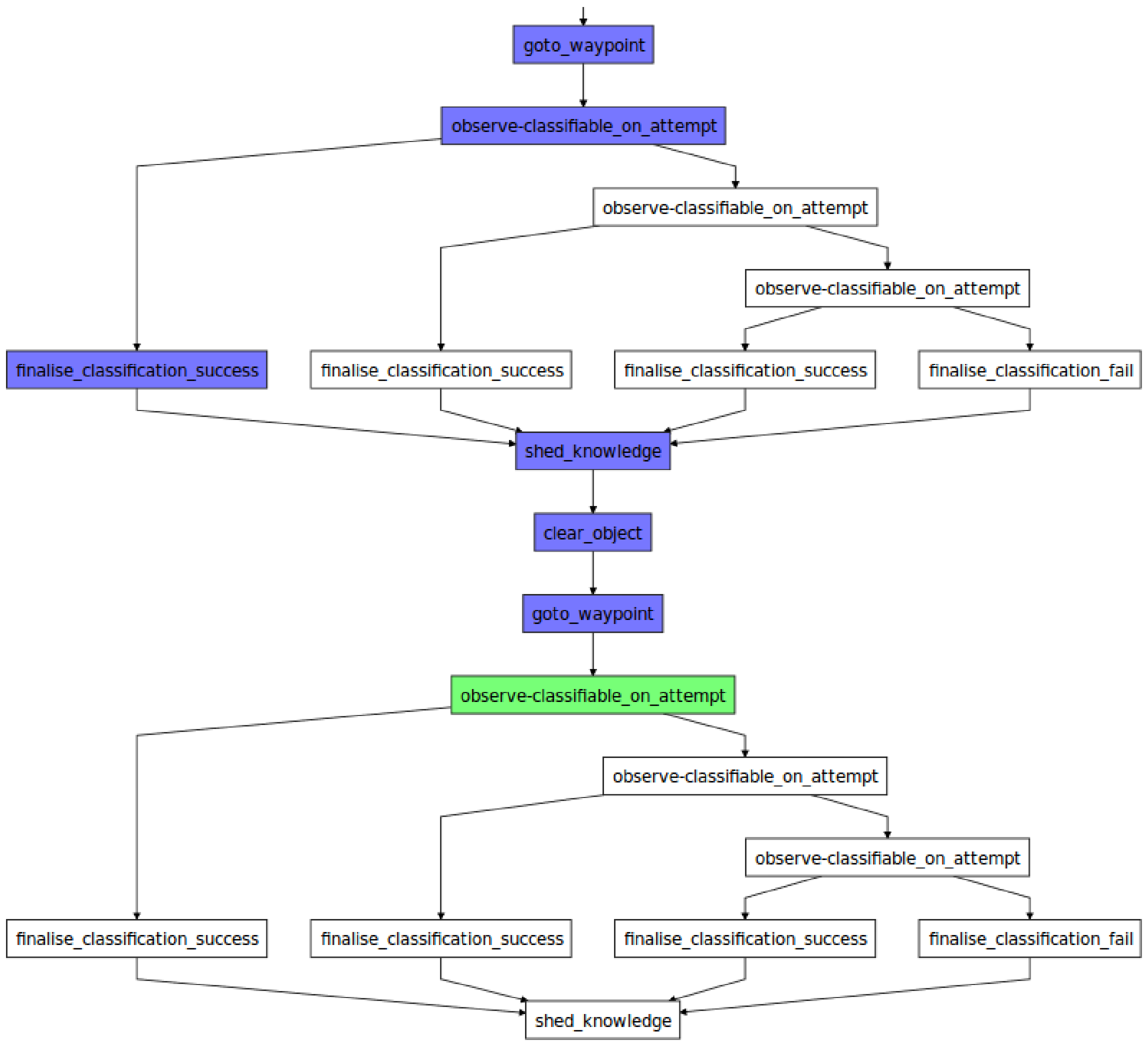
Uncertainty in AI Planning

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)



Plan Execution 4: Temporal and Conditional Dispatch together

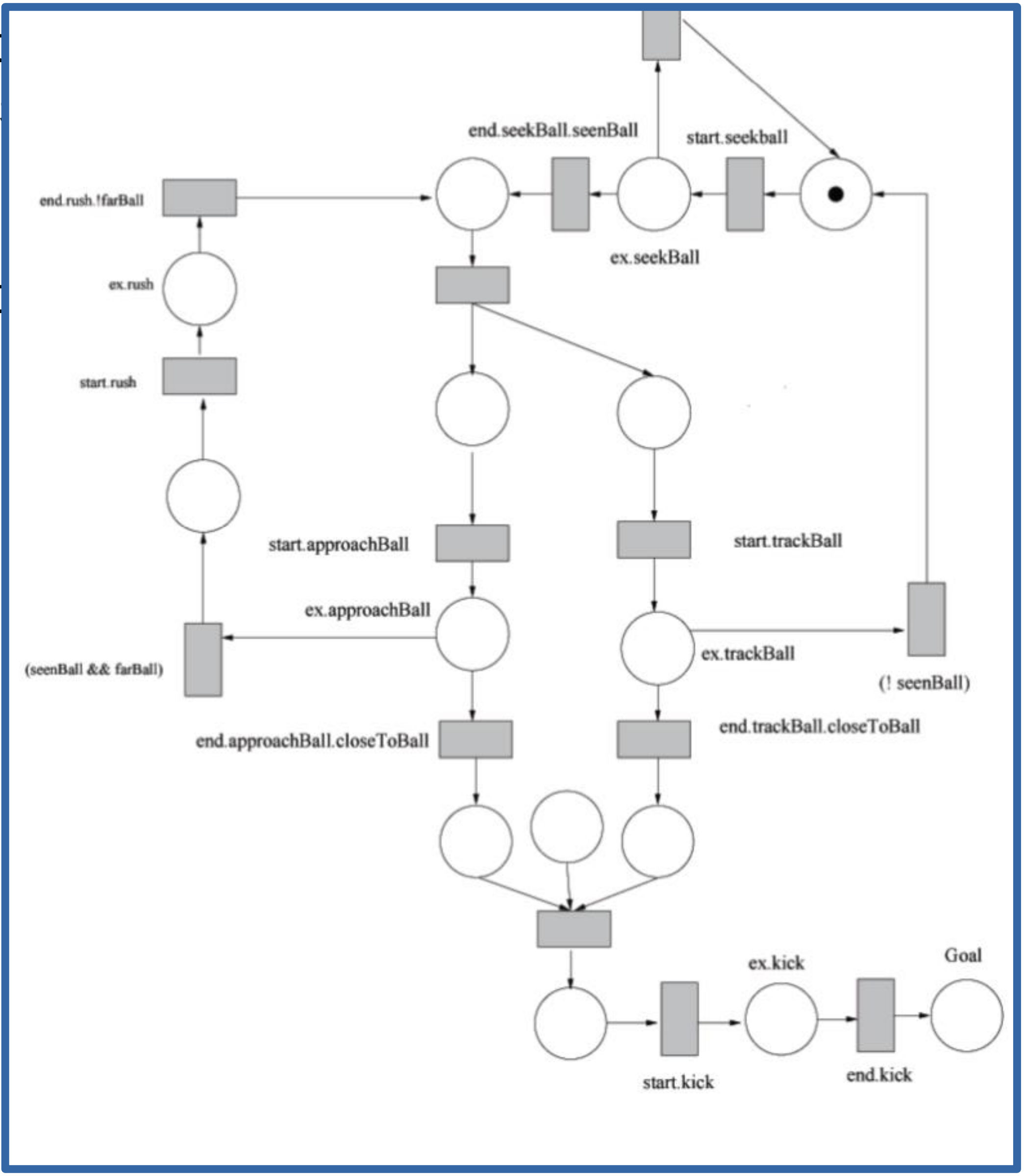
Robotics domains require a combination of temporal and conditional reasoning. Combining these two kinds of uncertainty can result in very complex structures.

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 combinat
reasoning. Combining these two kind
complex structures.

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



Plan Execution 4: Temporal and Conditional Dispatch together

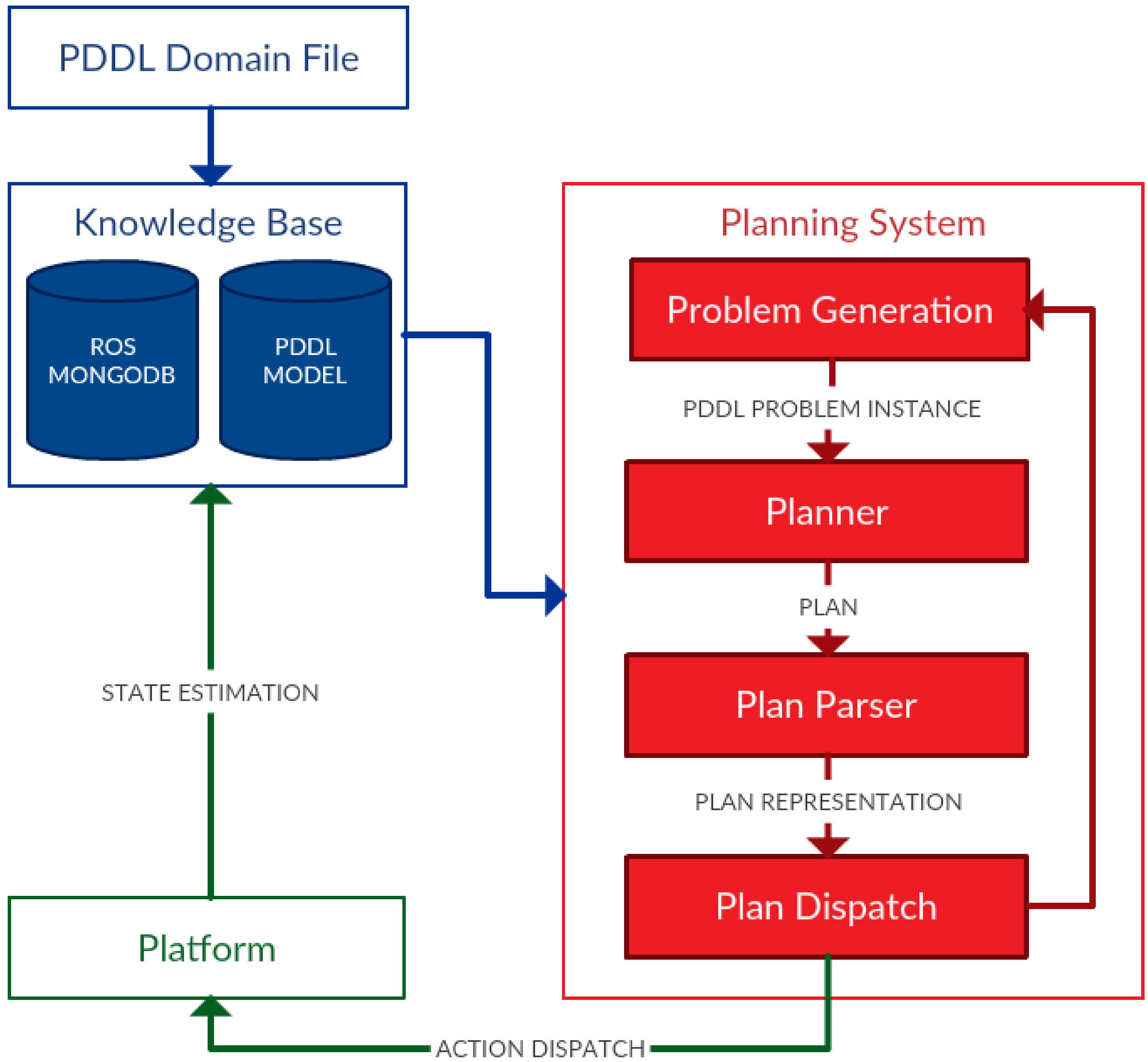
Robotics domains require a combination of temporal and conditional reasoning. Combining these two kinds of uncertainty can result in very complex structures.

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, Cashmore, Magazzeni, and Iocchi; 2017]*

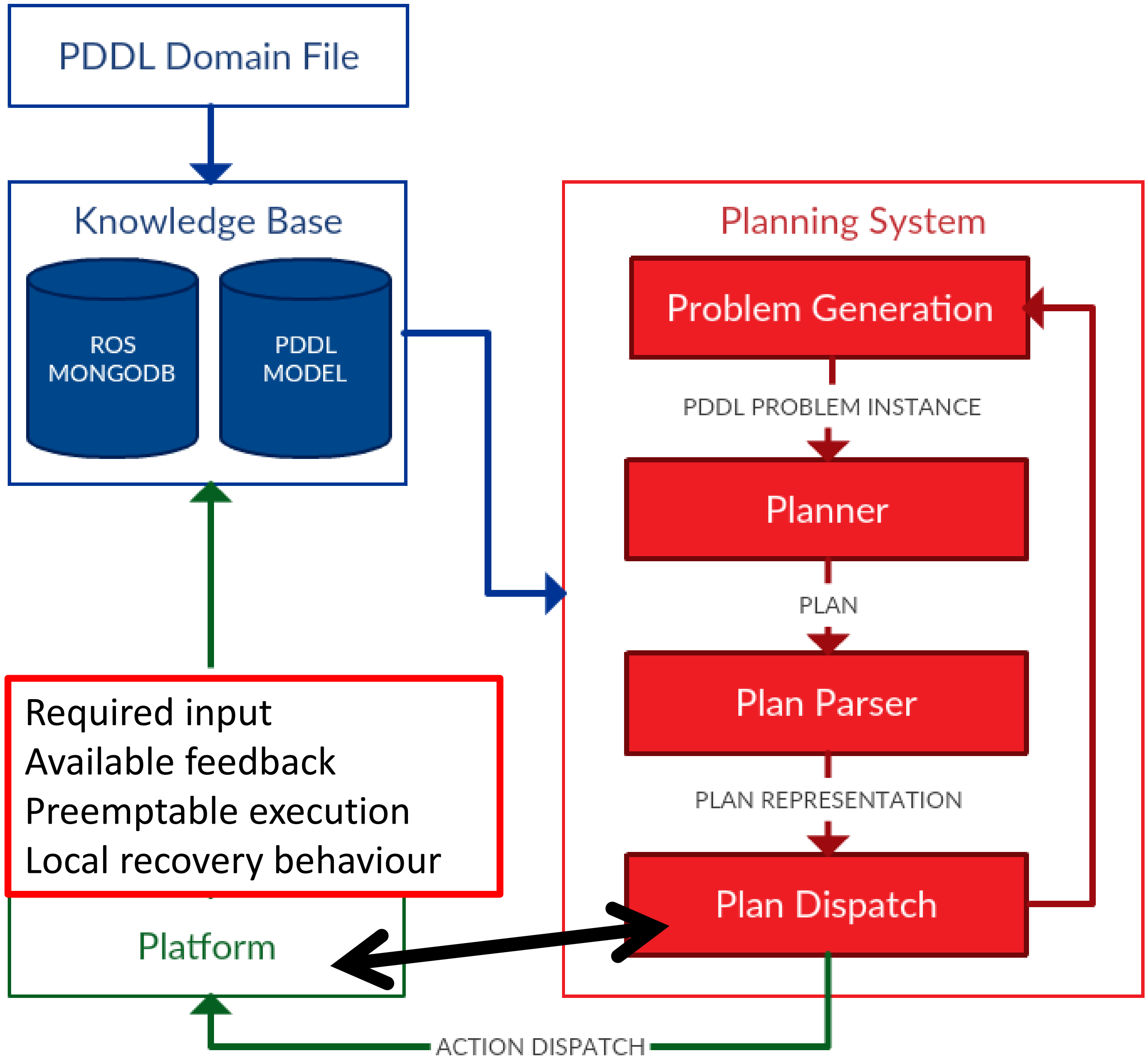
Summary of Very Simple Plan Execution

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.



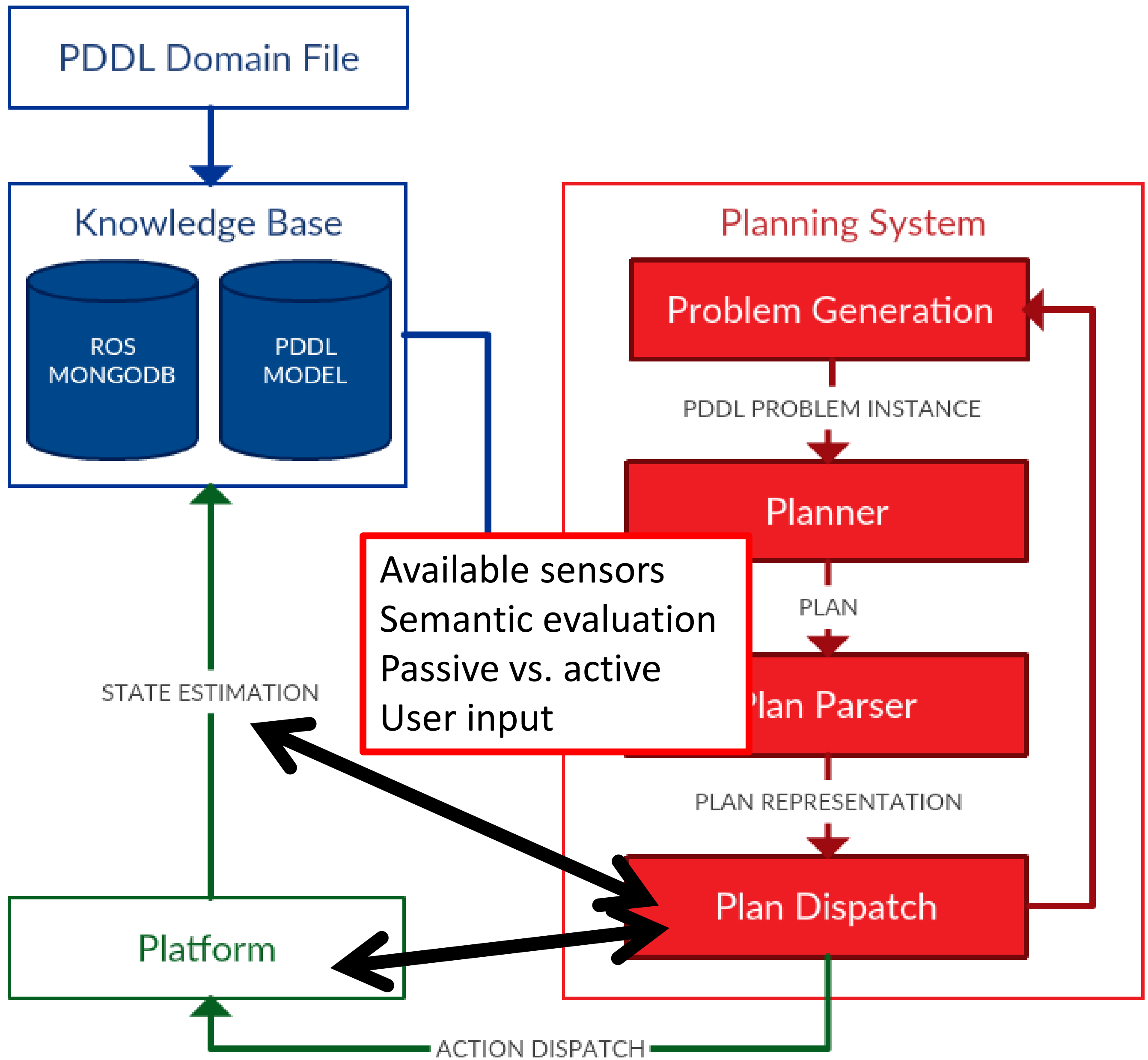
Summary of Very Simple Plan Execution

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.



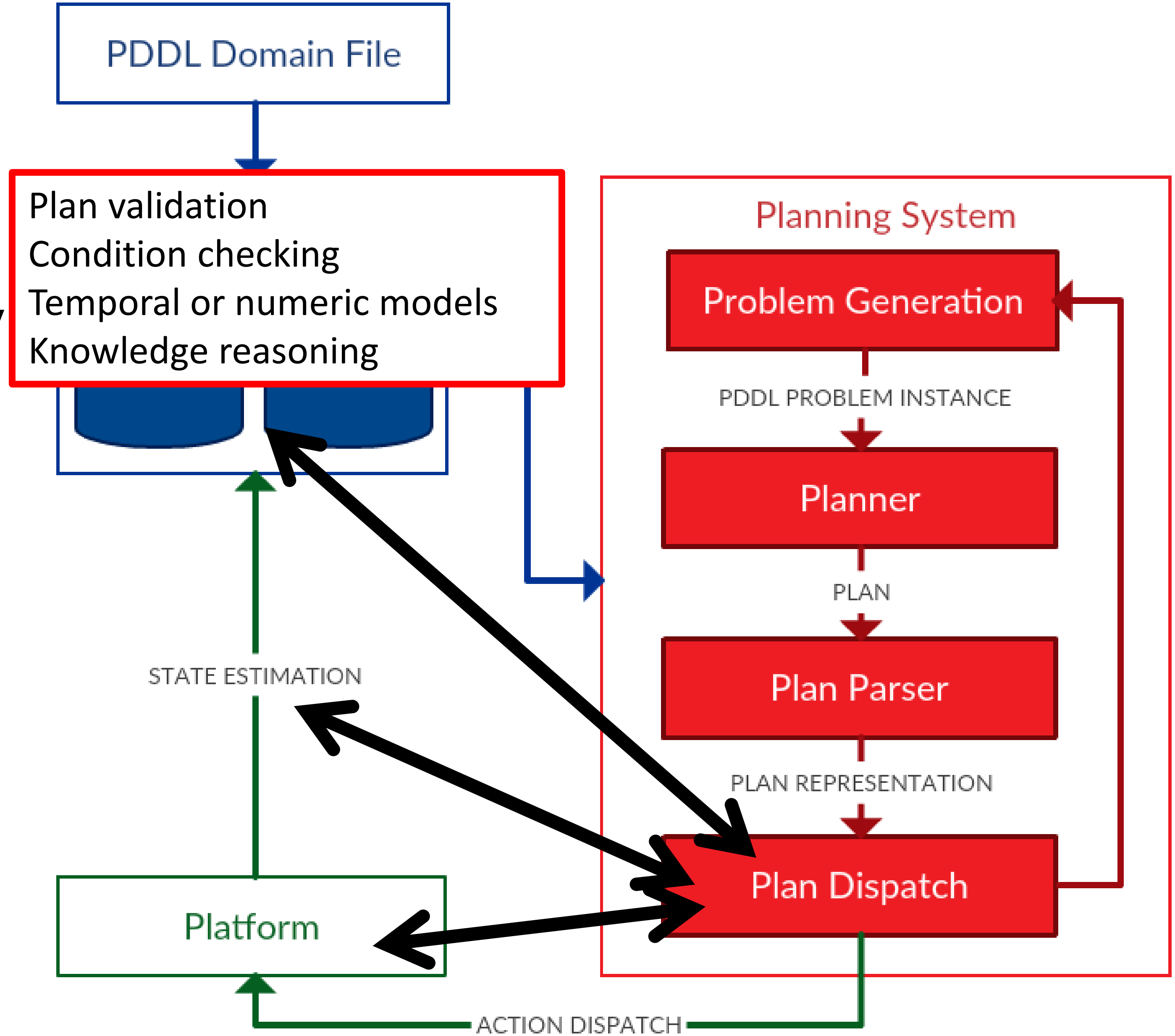
Summary of Very Simple Plan Execution

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.



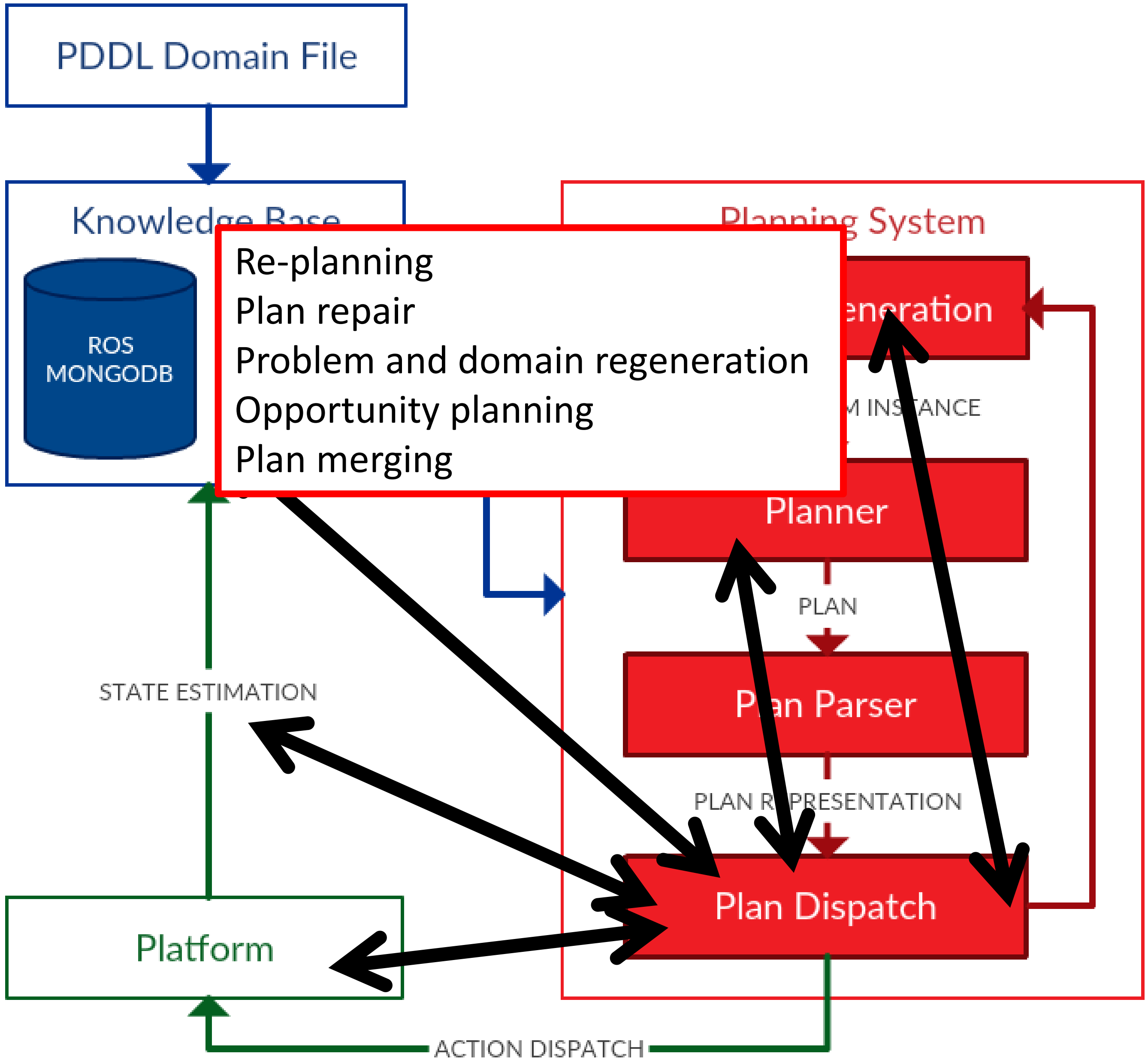
Summary of Very Simple Plan Execution

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

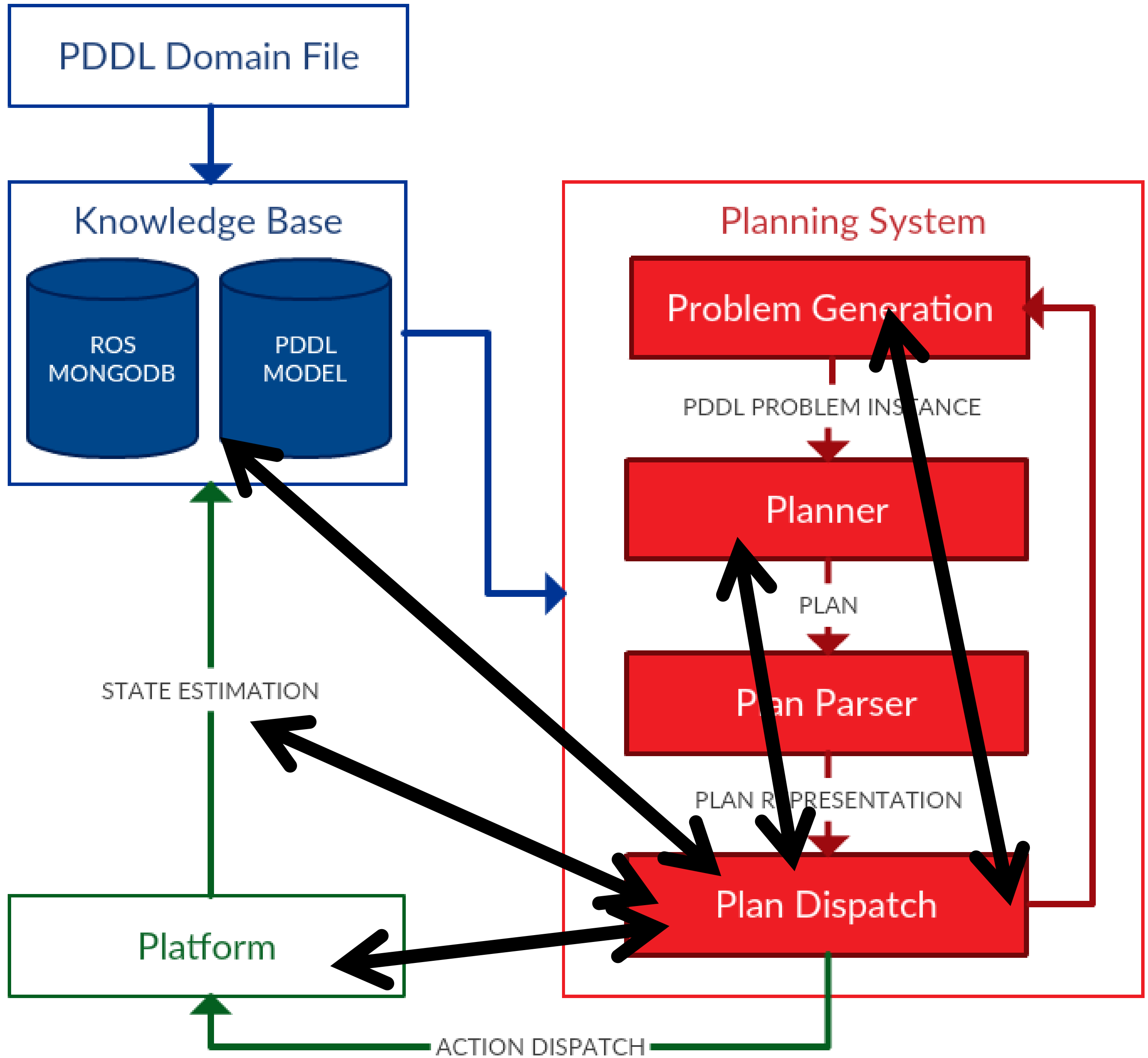
Plan Execution depends upon many components in the system. Changing any one of which will change the robot behaviour, and change the criteria under which the plan will succeed or fail.



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.

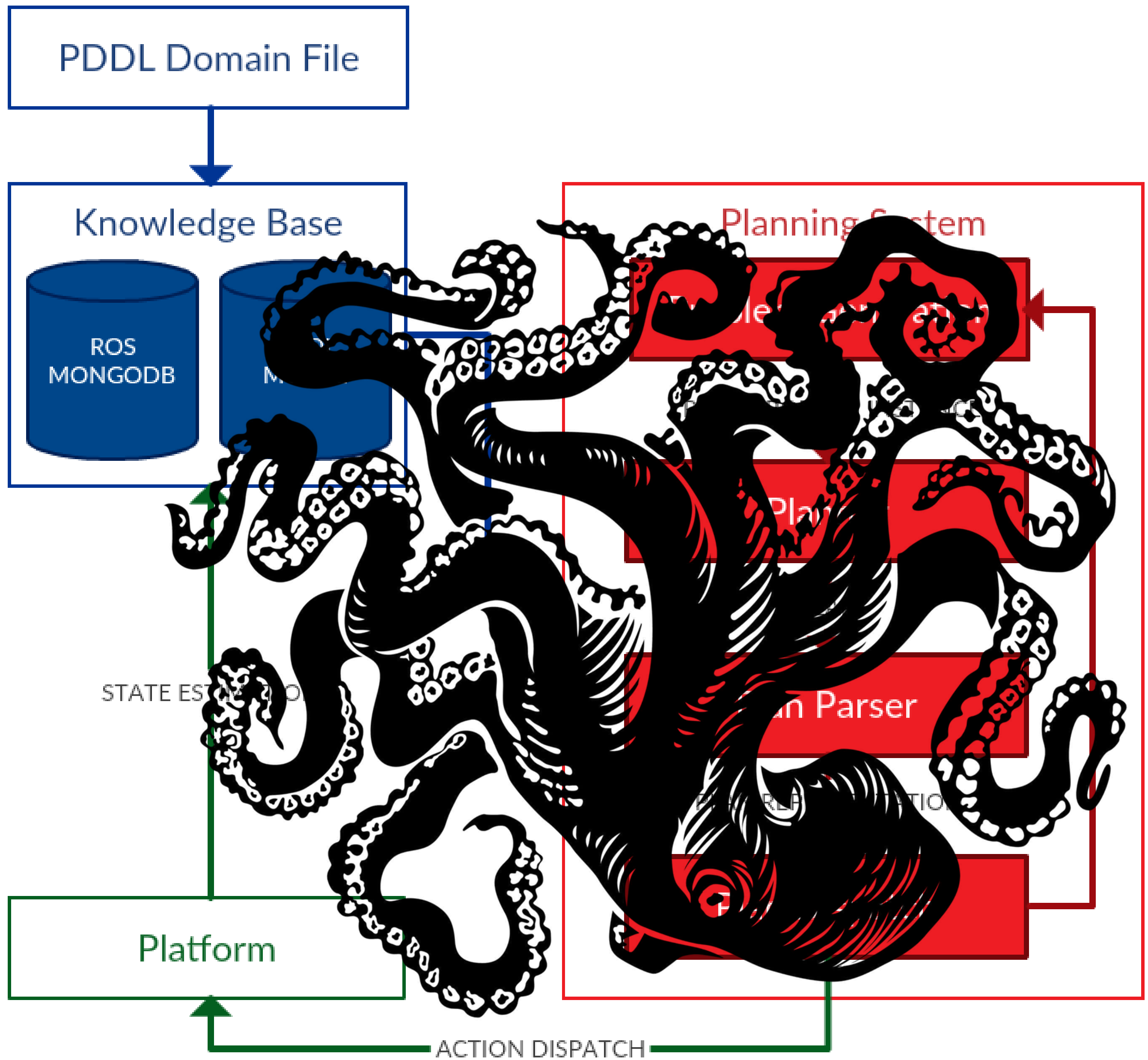
The execution of a plan is an emergent behaviour of the whole system.



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.

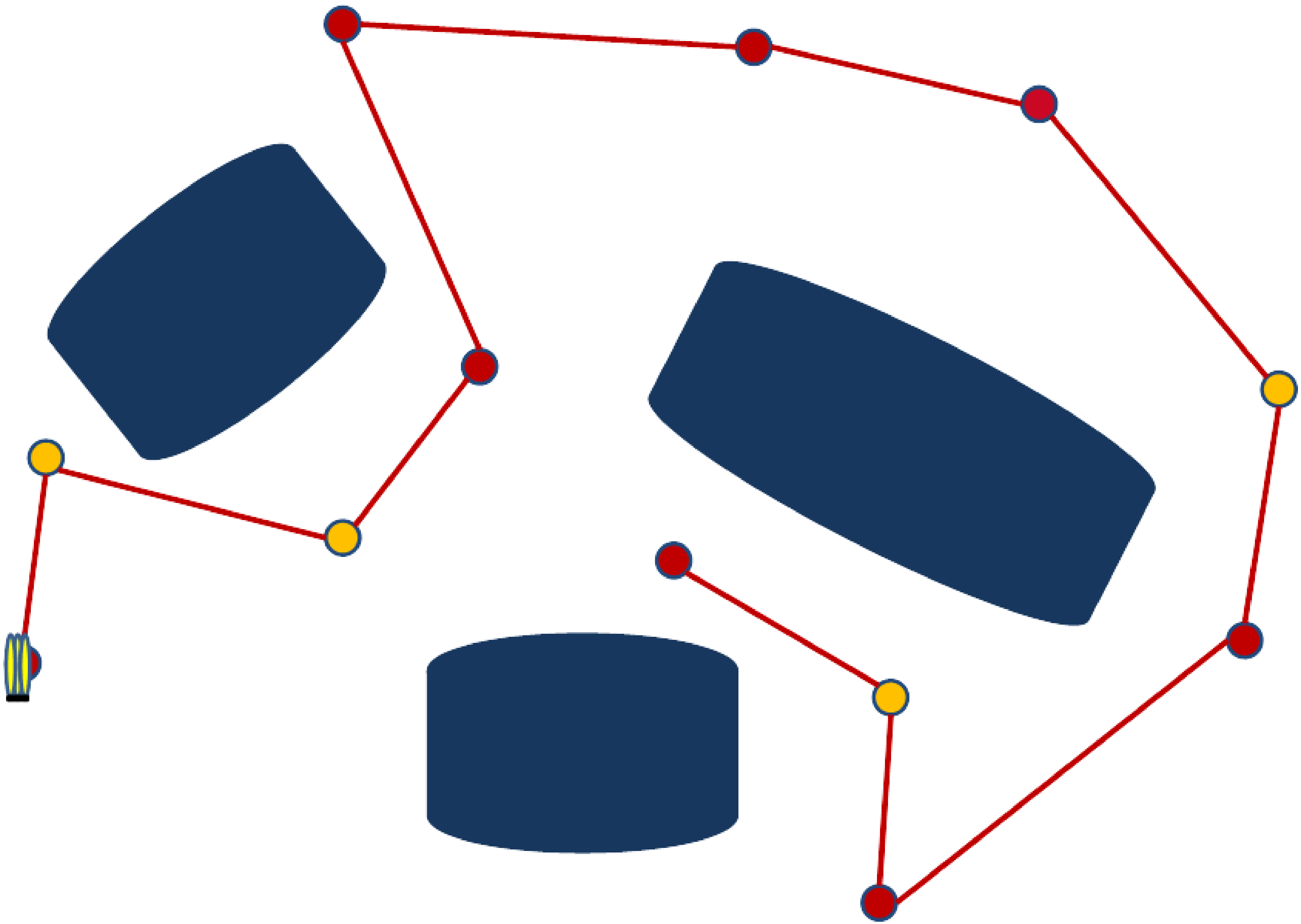
Dispatching more than a Single Plan

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

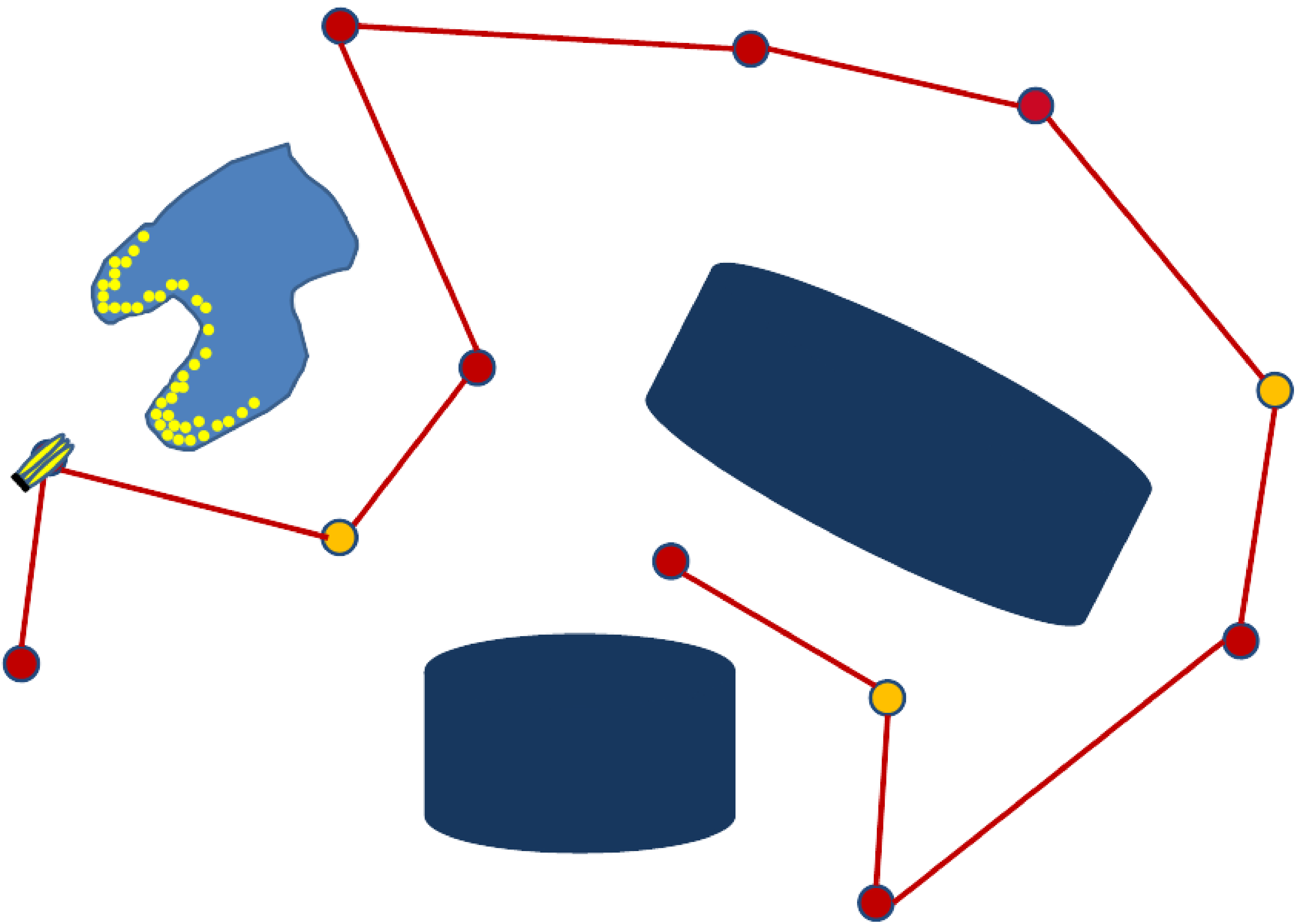
One very common and simple scenario with robots is planning a search scenario. For tracking targets, tidying household objects, or interacting with people.

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

Dispatching more than a Single Plan

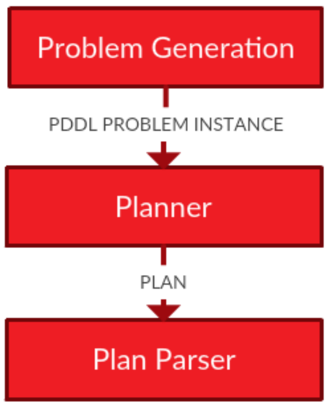


Dispatching more than a Single Plan



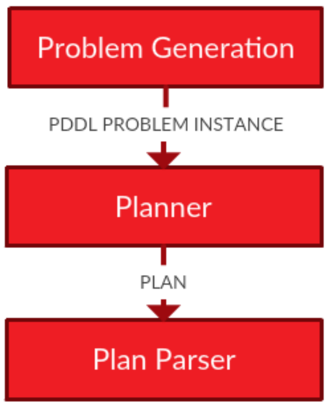
Hierarchical and Recursive Planning

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.



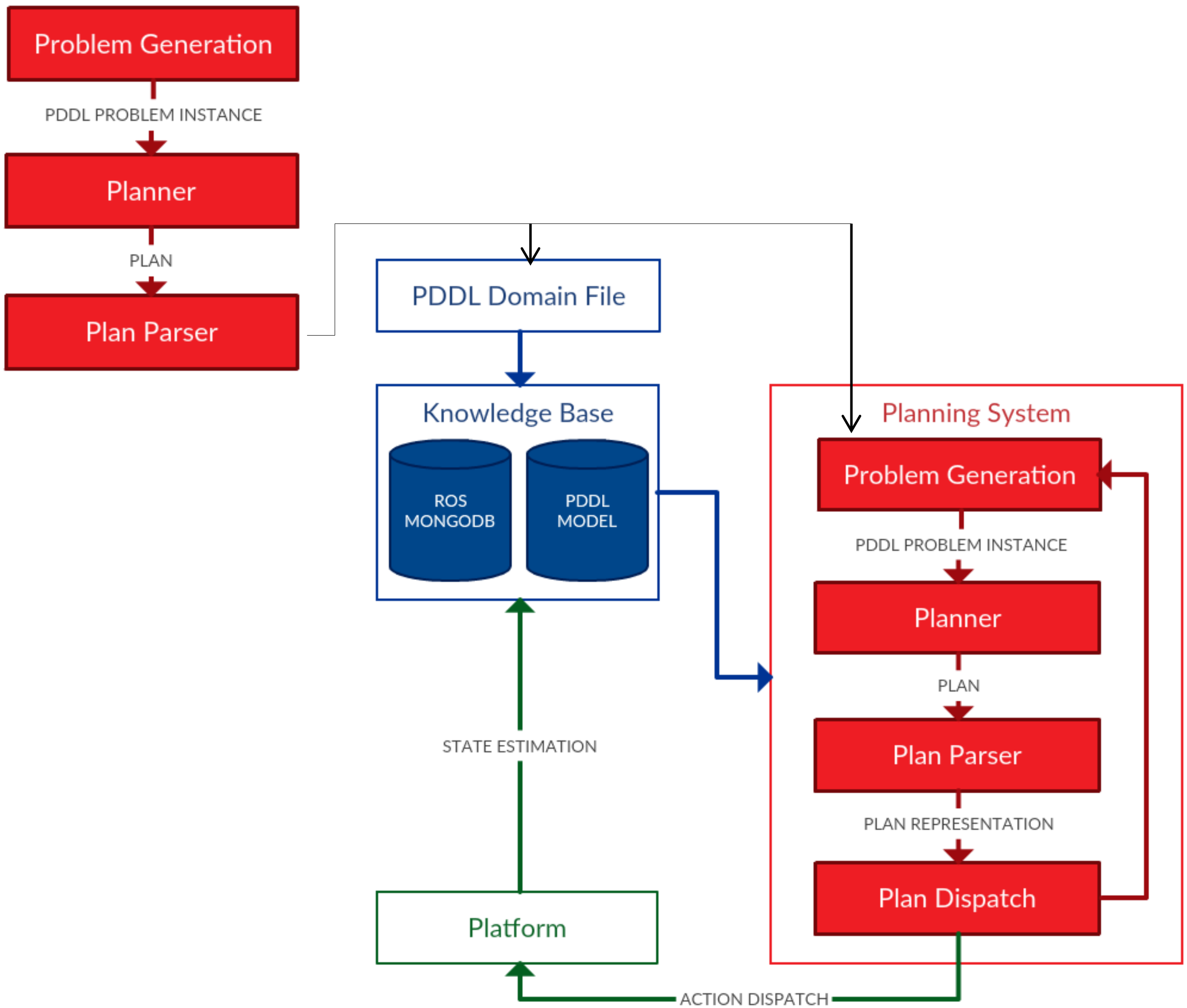

```
0.000: (correct_position auv0 wp_auv0) [3.000]
3.001: (do_hover_fast auv0 wp_auv0 strategic_location_7)
[11.403]
14.405: (correct_position auv0_strategic_location_78)
[3.000]
17.406: (observe_inspection_point auv0 strategic_location_7
inspection_point_2) [10.000]
27.407: (correct_position auv0 strategic_location_7)
[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]
...
```

complete_mission

Energy consumption = 10W
Duration = 86.43s

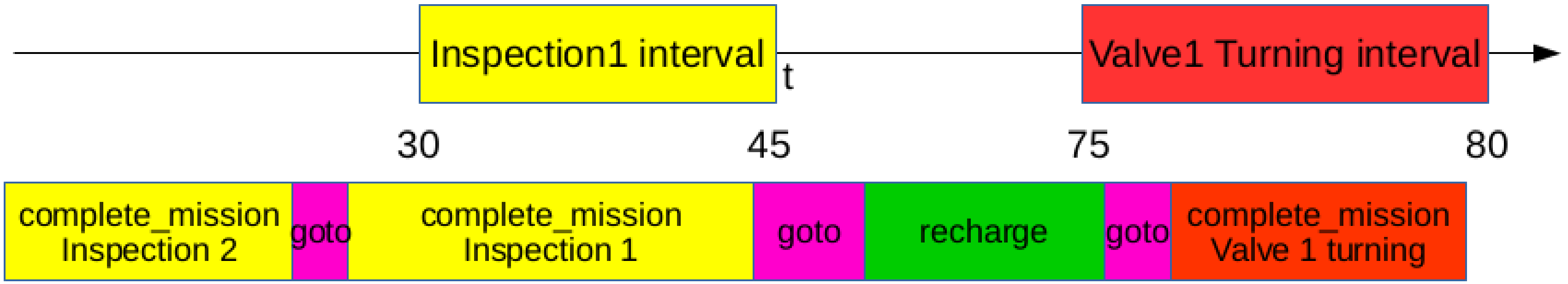
Hierarchical and Recursive Planning

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

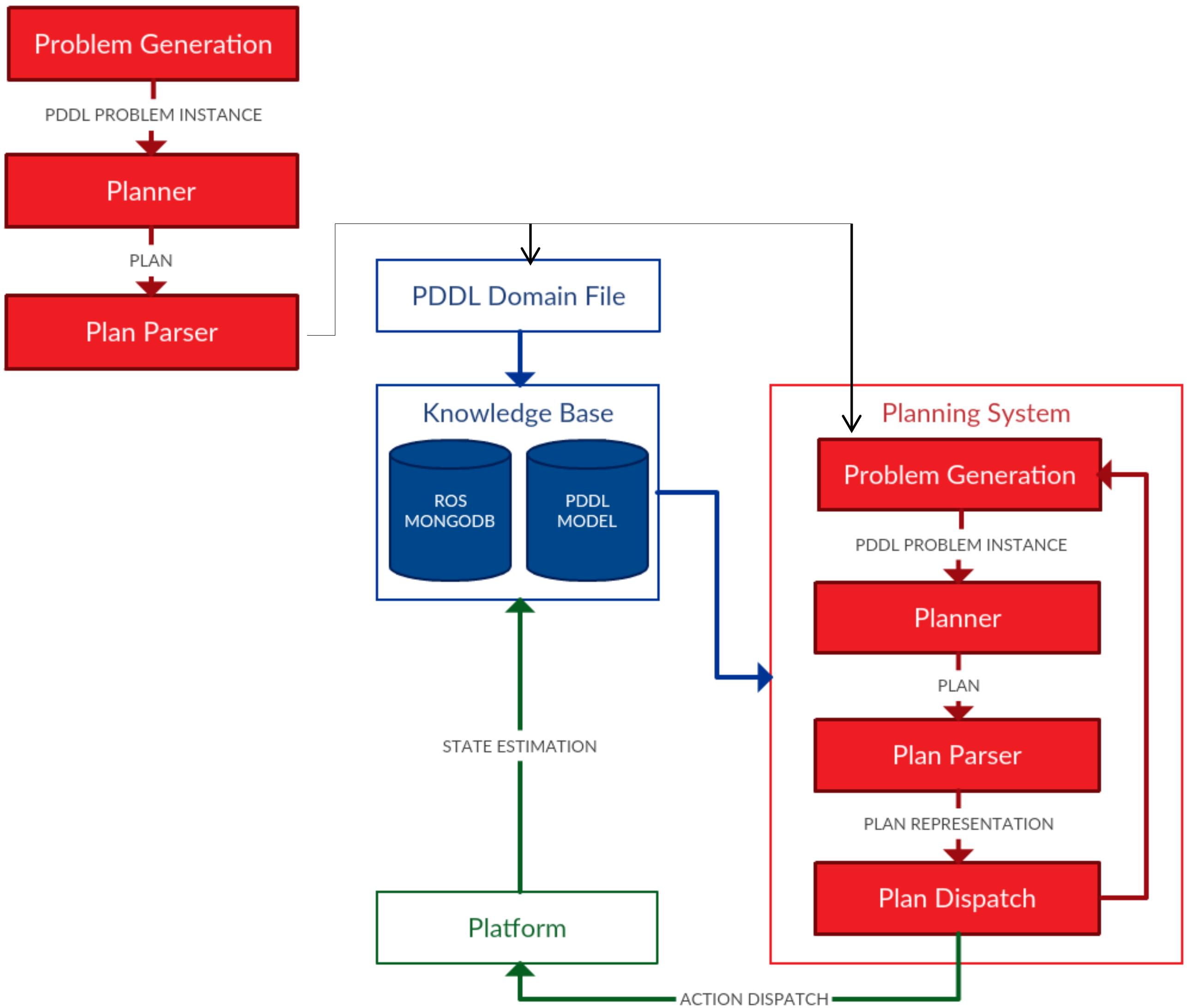
For each task we generate a *tactical plan*. The time and resource constraints are used in the generation of the strategic problem.



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

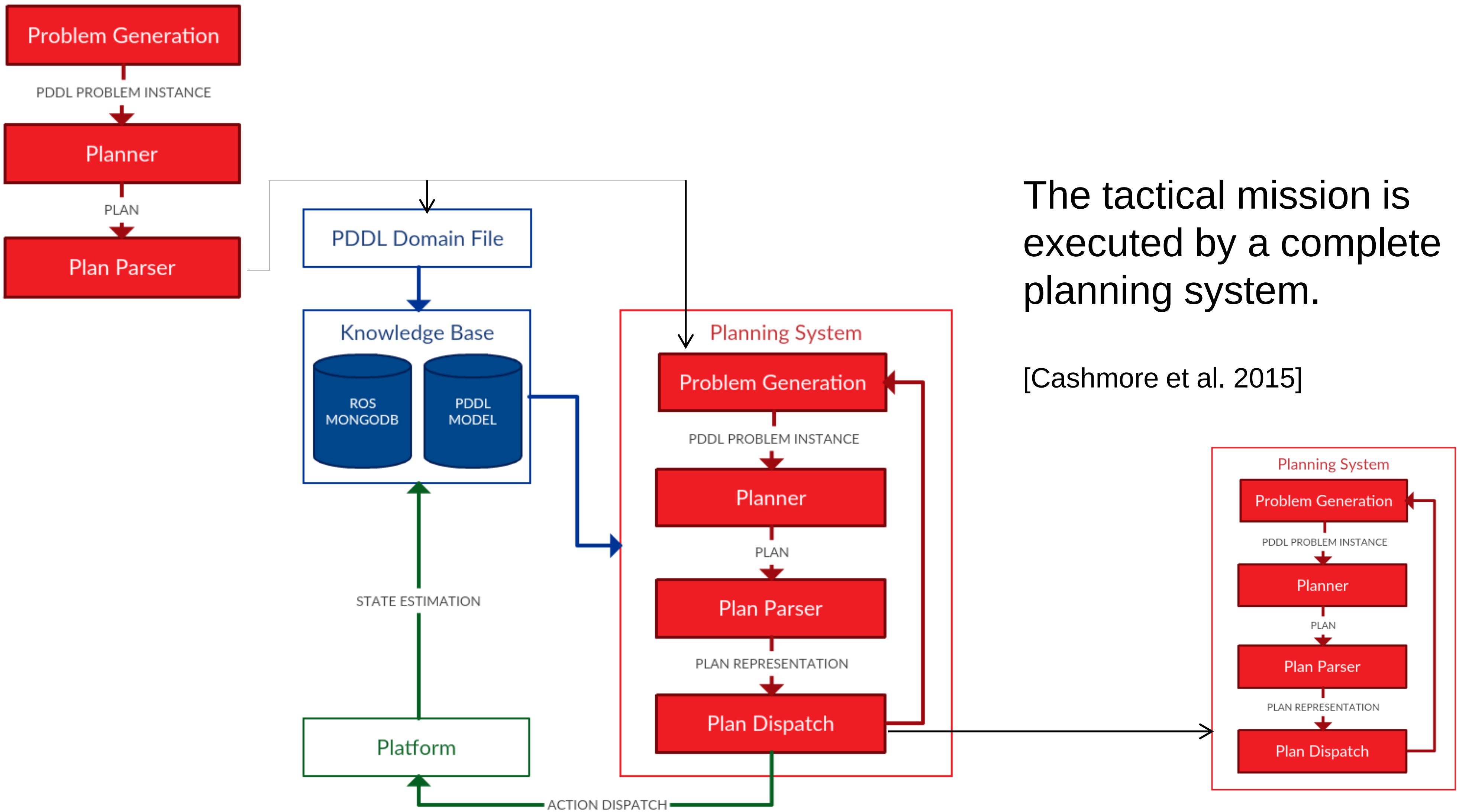
Hierarchical and Recursive Planning

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.



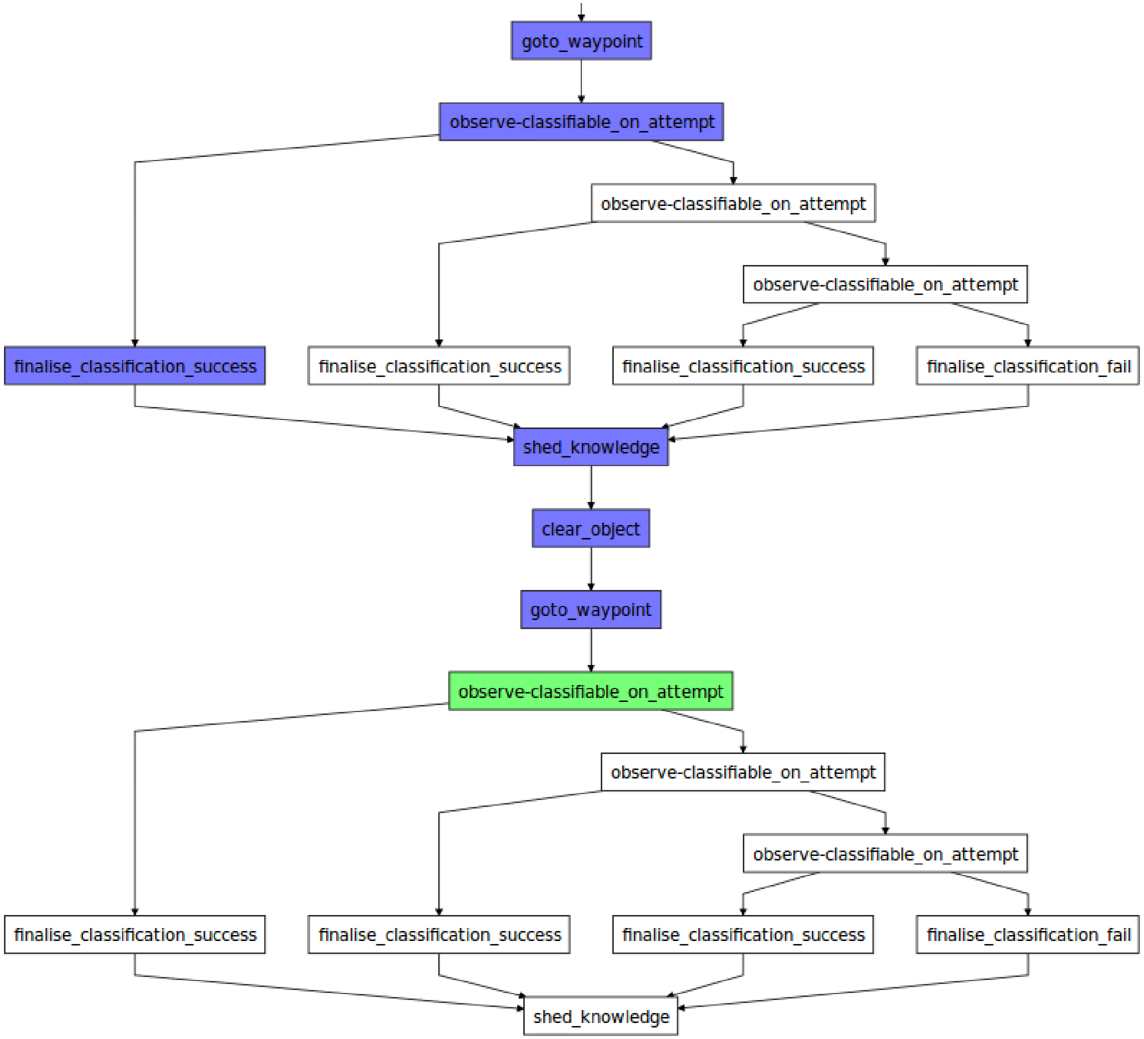
The tactical mission is executed by a complete planning system.

[Cashmore et al. 2015]

Hierarchical and Recursive Planning

Observing an object has two outcomes:

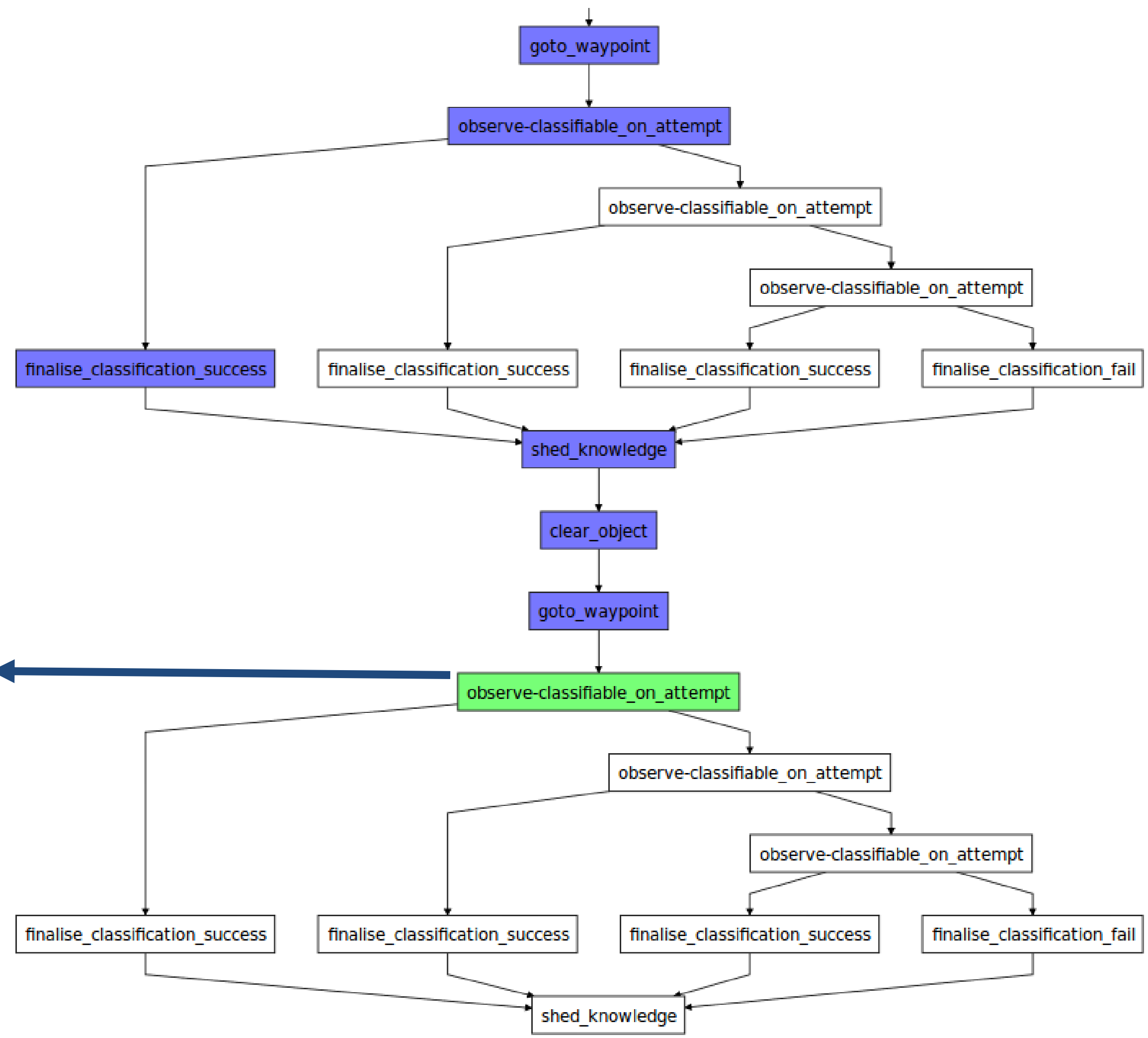
- Success. The object is classified or recognised
- Failure. The object type is still unknown, but new viewpoints are generated to discriminate between high-probability possibilities.



Hierarchical and Recursive Planning

The action corresponds to a short tactical plan to observe viewpoints.

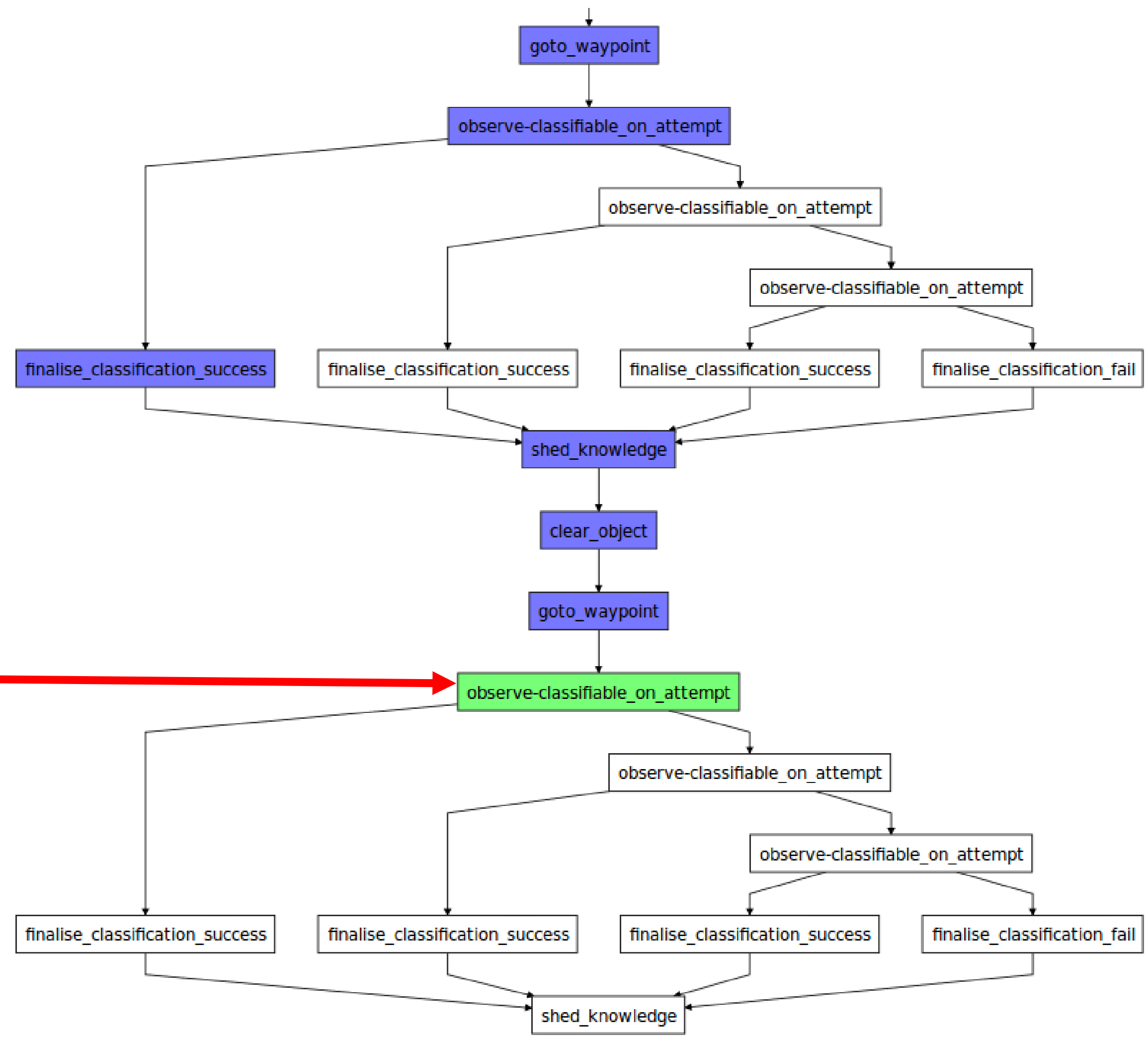
0.000: (goto_waypoint) [10.0]
0.000: (observe) [2.0]
0.000: (goto_waypoint) [10.0]
0.000: (pickup-object) [16.0]



Hierarchical and Recursive Planning

The action corresponds to a short tactical plan to observe viewpoints.

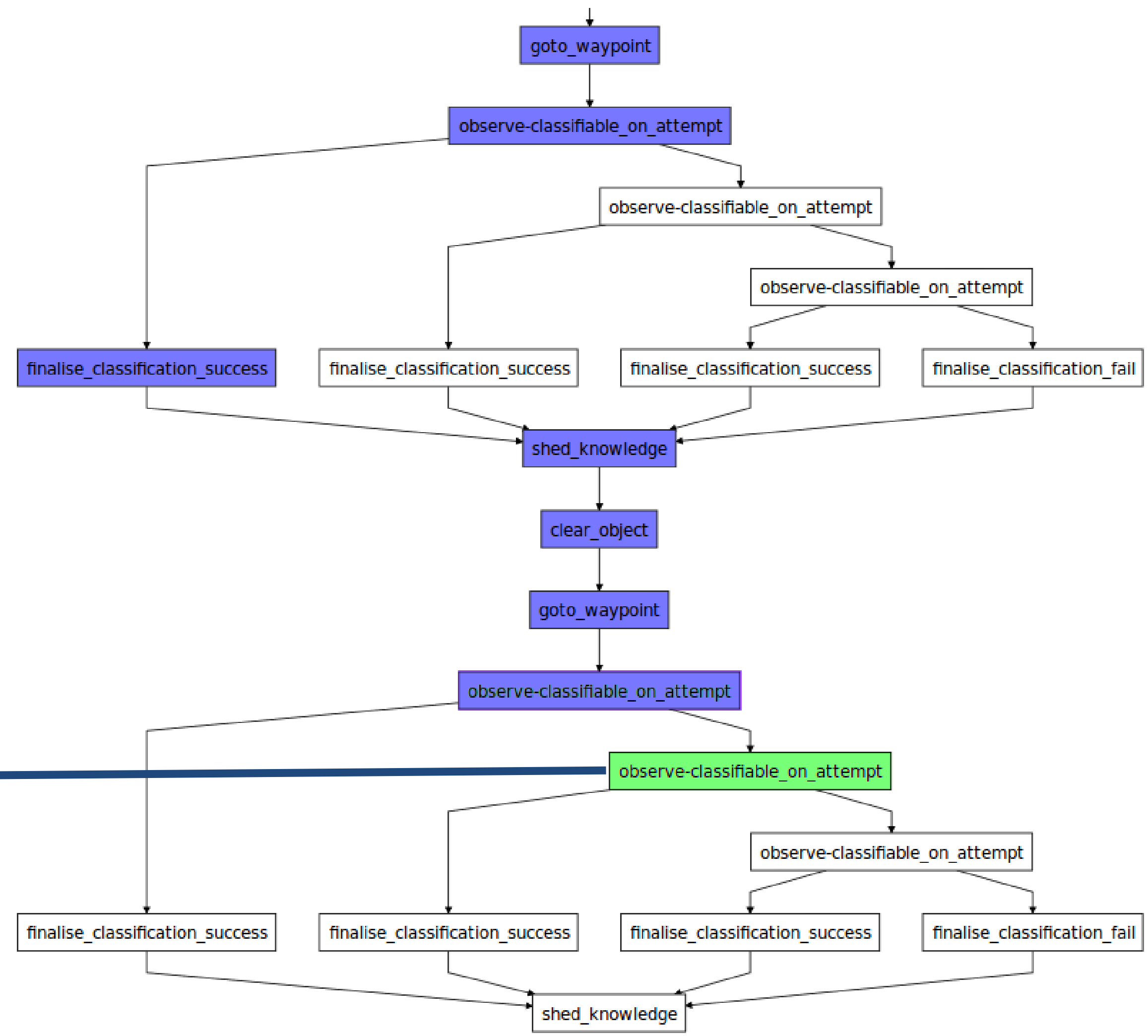
0.000: (goto_waypoint) [10.0]
0.000: (observe) [2.0]
0.000: (goto_waypoint) [10.0]
0.000: (pickup-object) [16.0]

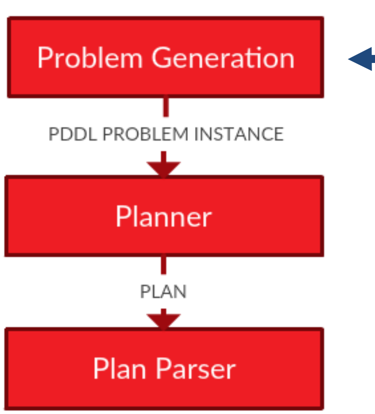


Hierarchical and Recursive Planning

The action corresponds to a short tactical plan to observe viewpoints.

The next tactical plan can only be generated once the new viewpoints are known.

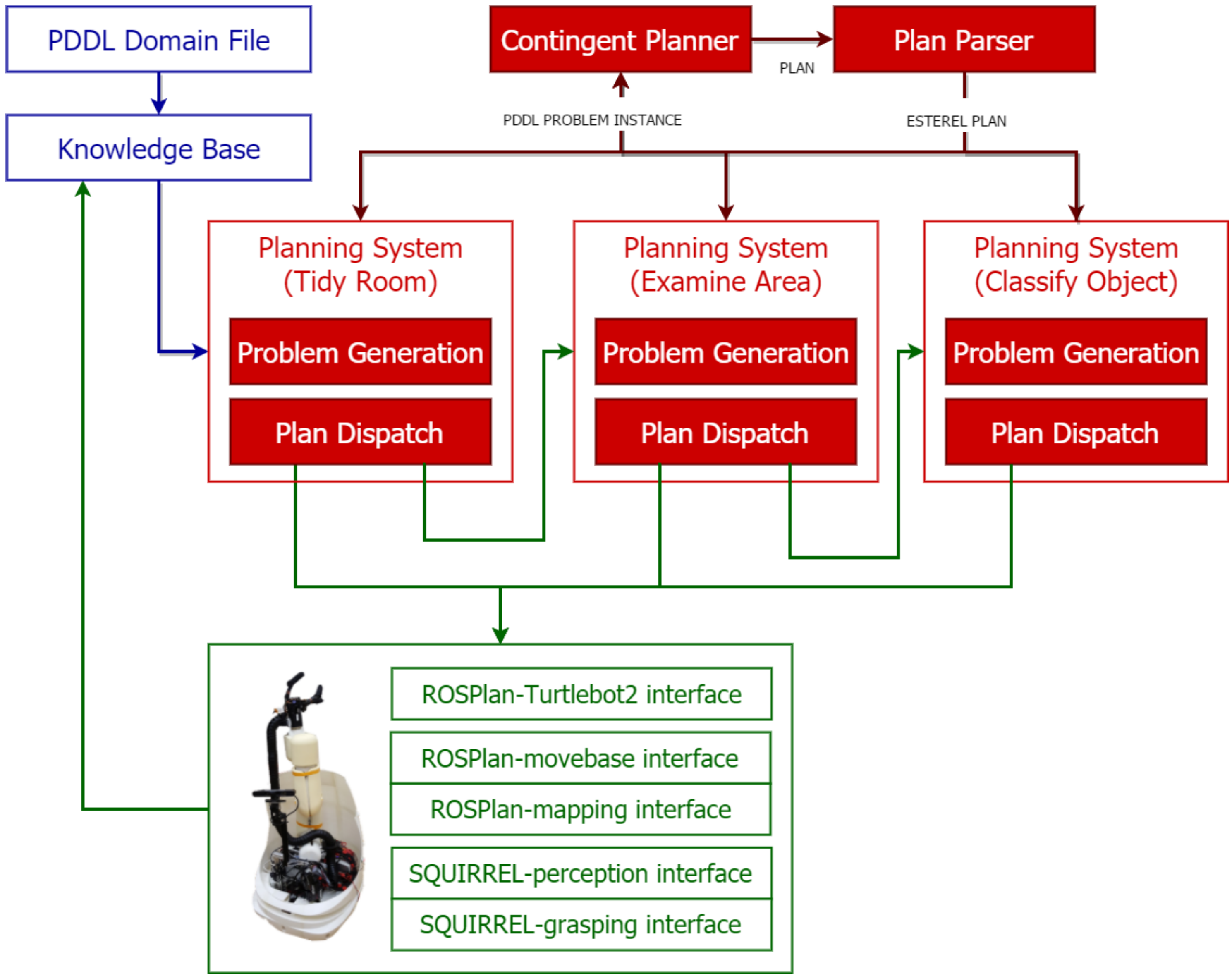




Hierarchical and Recursive Planning

The components of the system are the same as the very simple dispatch.

The behaviour of the robot is very different.



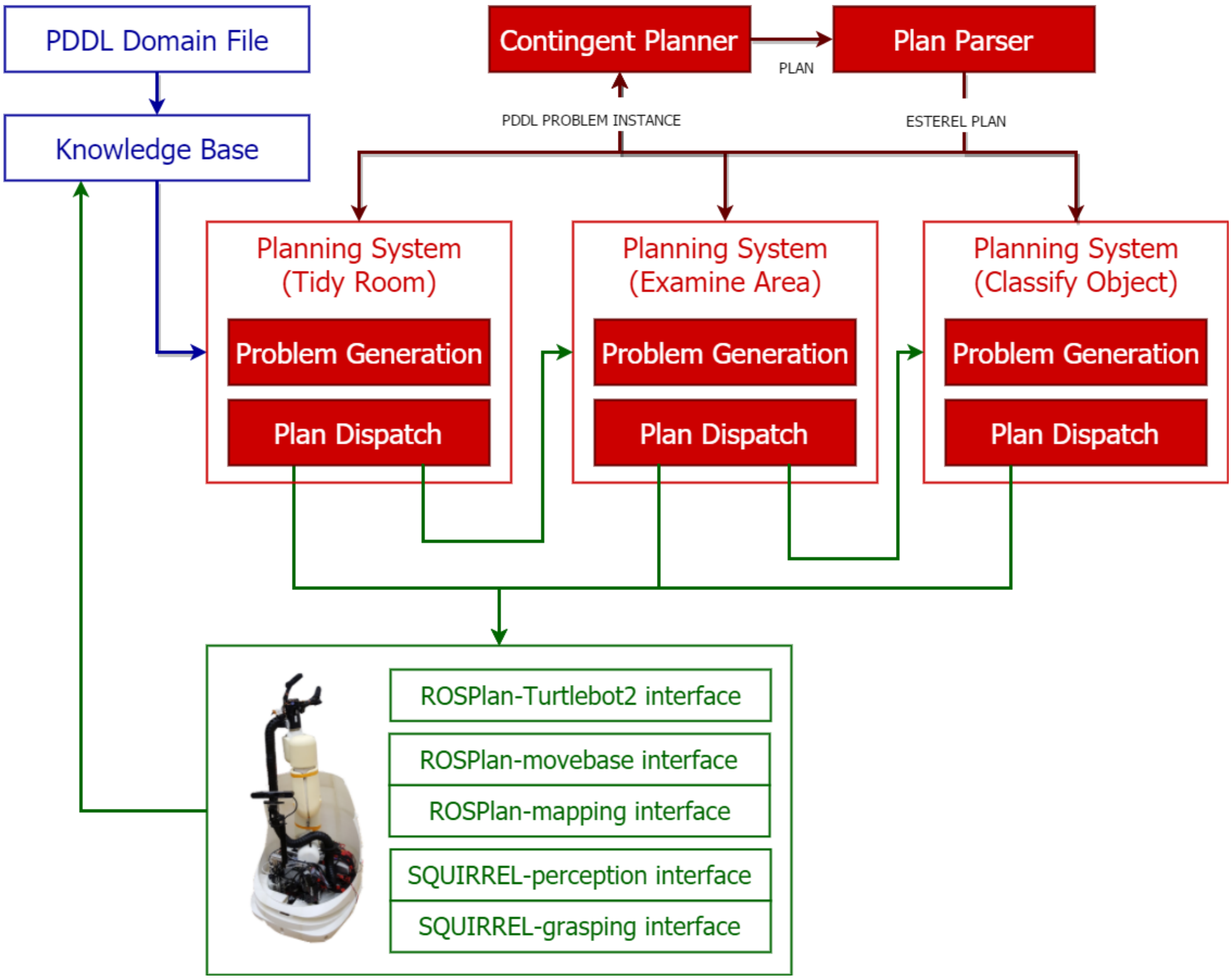
Hierarchical and Recursive Planning

The components of the system are the same as the very simple dispatch.

The behaviour of the robot is very different.

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

Both the components and how they are used.



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 more Plans: Opportunistic Planning

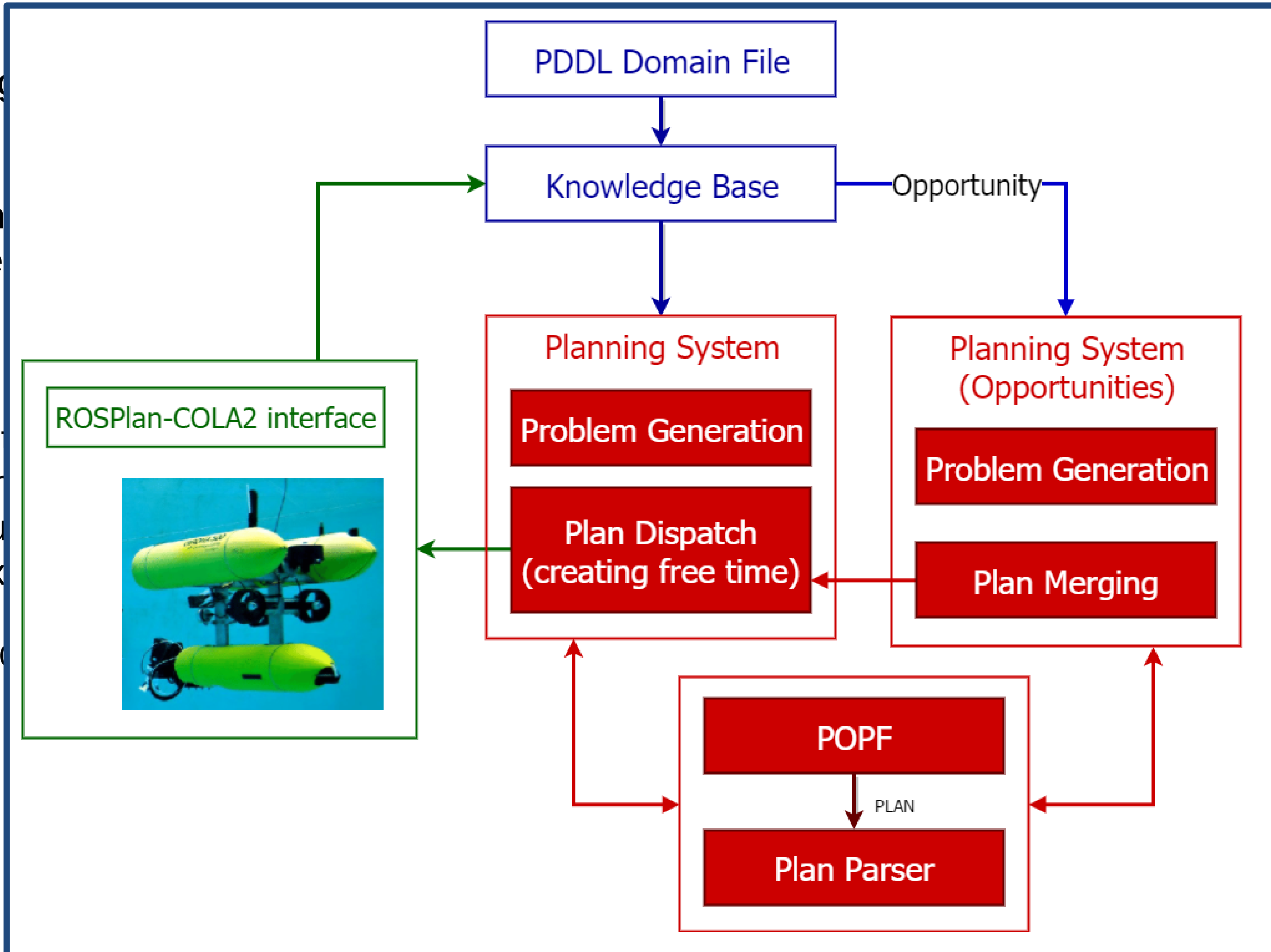
New plans are
goals and the
current plan.

If the new plan
then it is imme

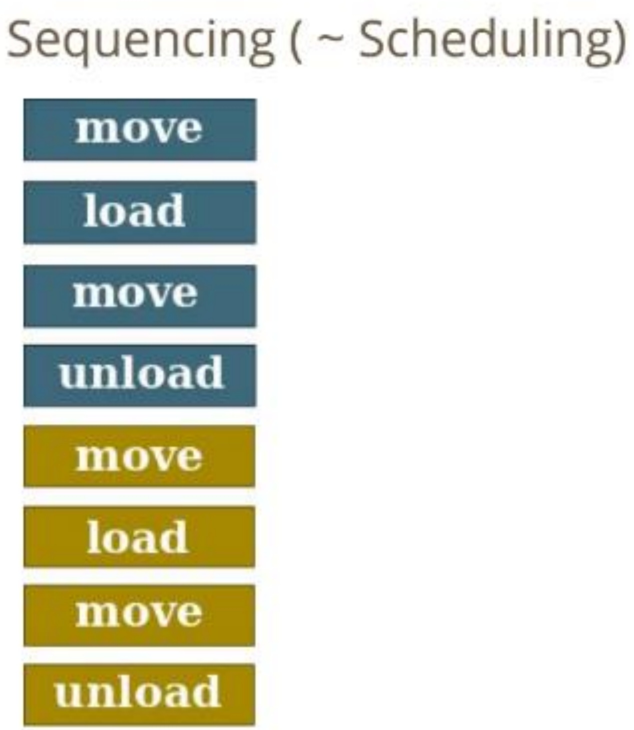
The approach

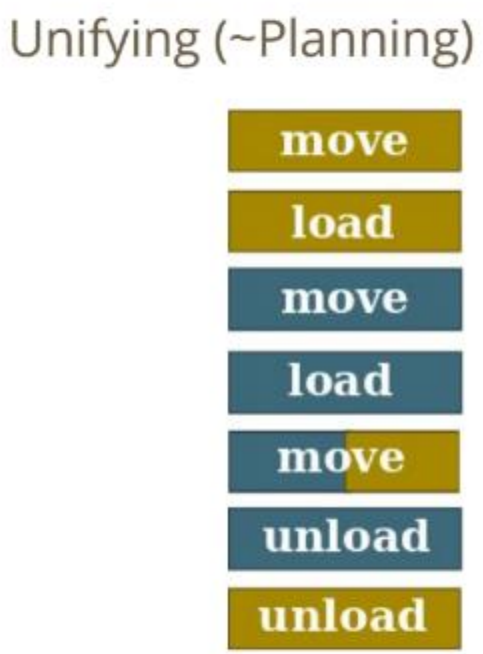
If an opportuni
of a plan fragm
plan can be pu
plan can be ex

[Cashmore et al. 2



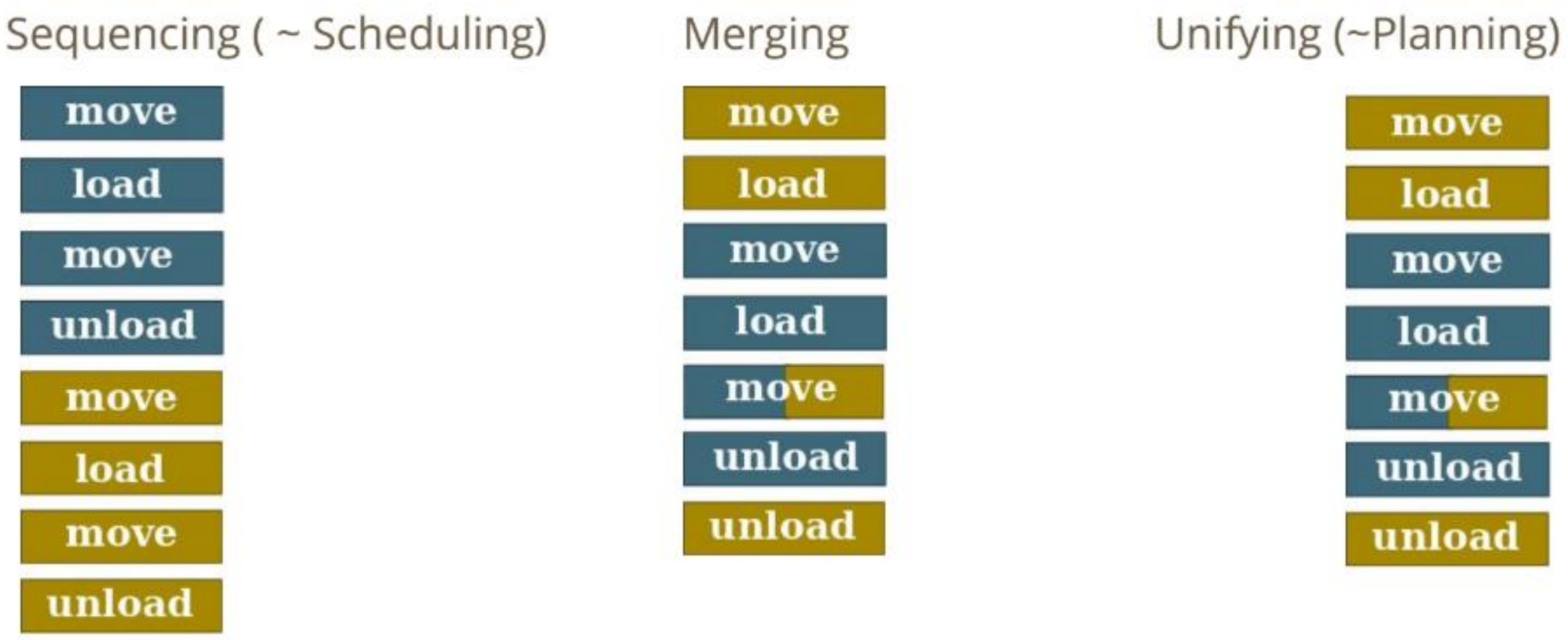
Dispatching Plans at the same time





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.

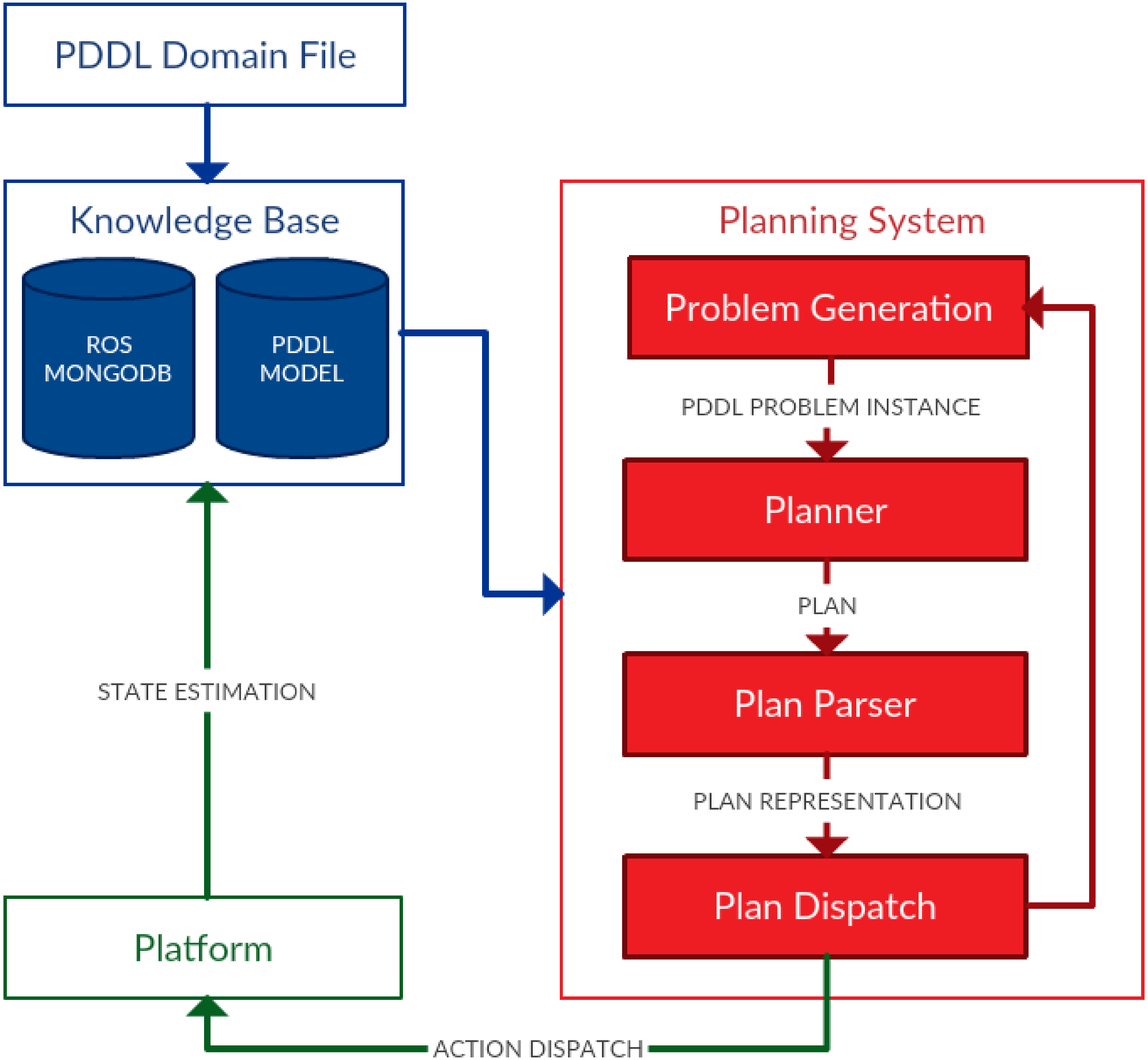
Plans can be merged in a more intelligent way. A single action can support the advancement towards multiple goals.

[Mudrova et al. 2016]

ROSPlan and PNP

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

The plan is validated against a model that is continually changing and only partially sensed.



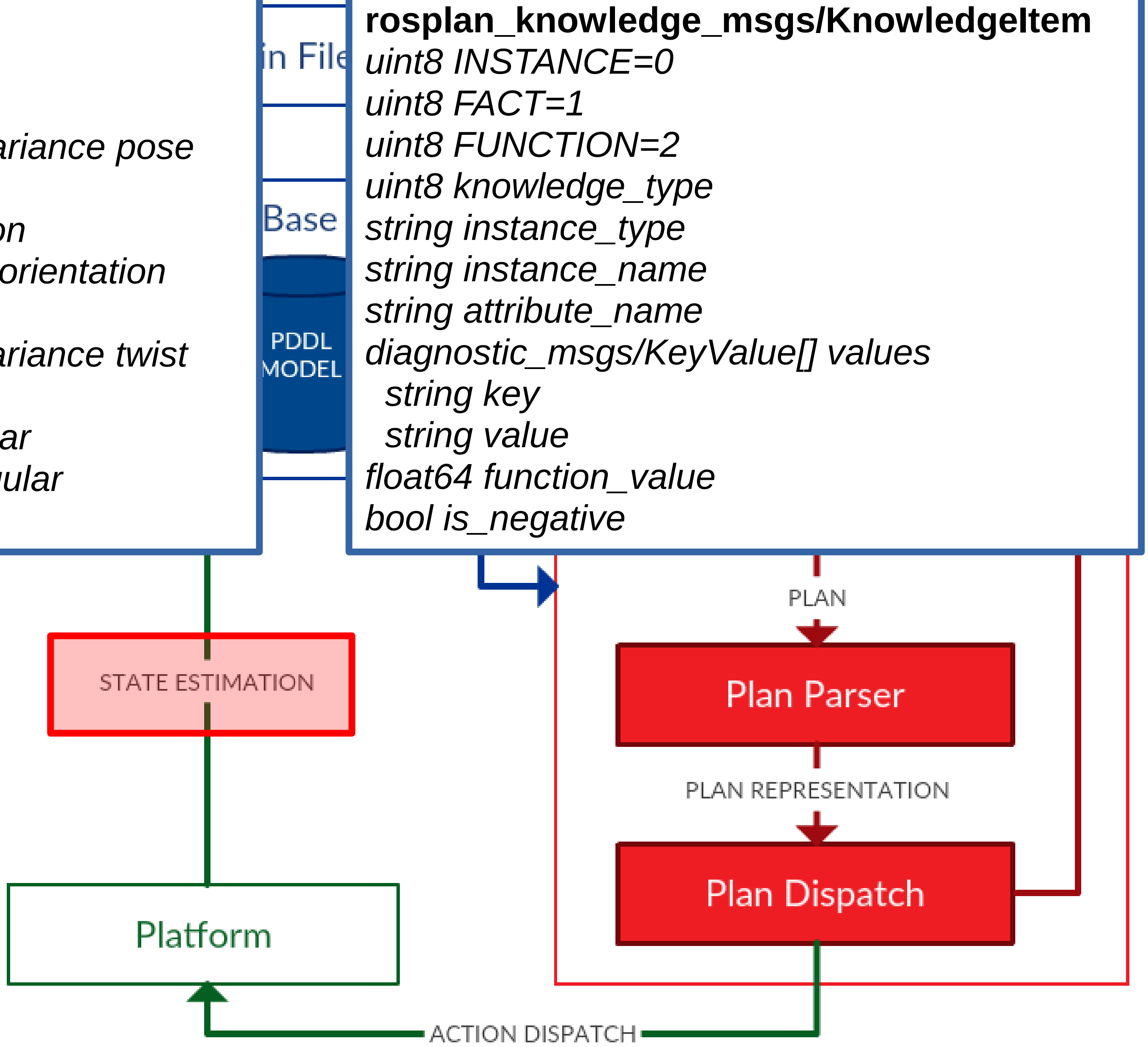
ROSPlan and PNP

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

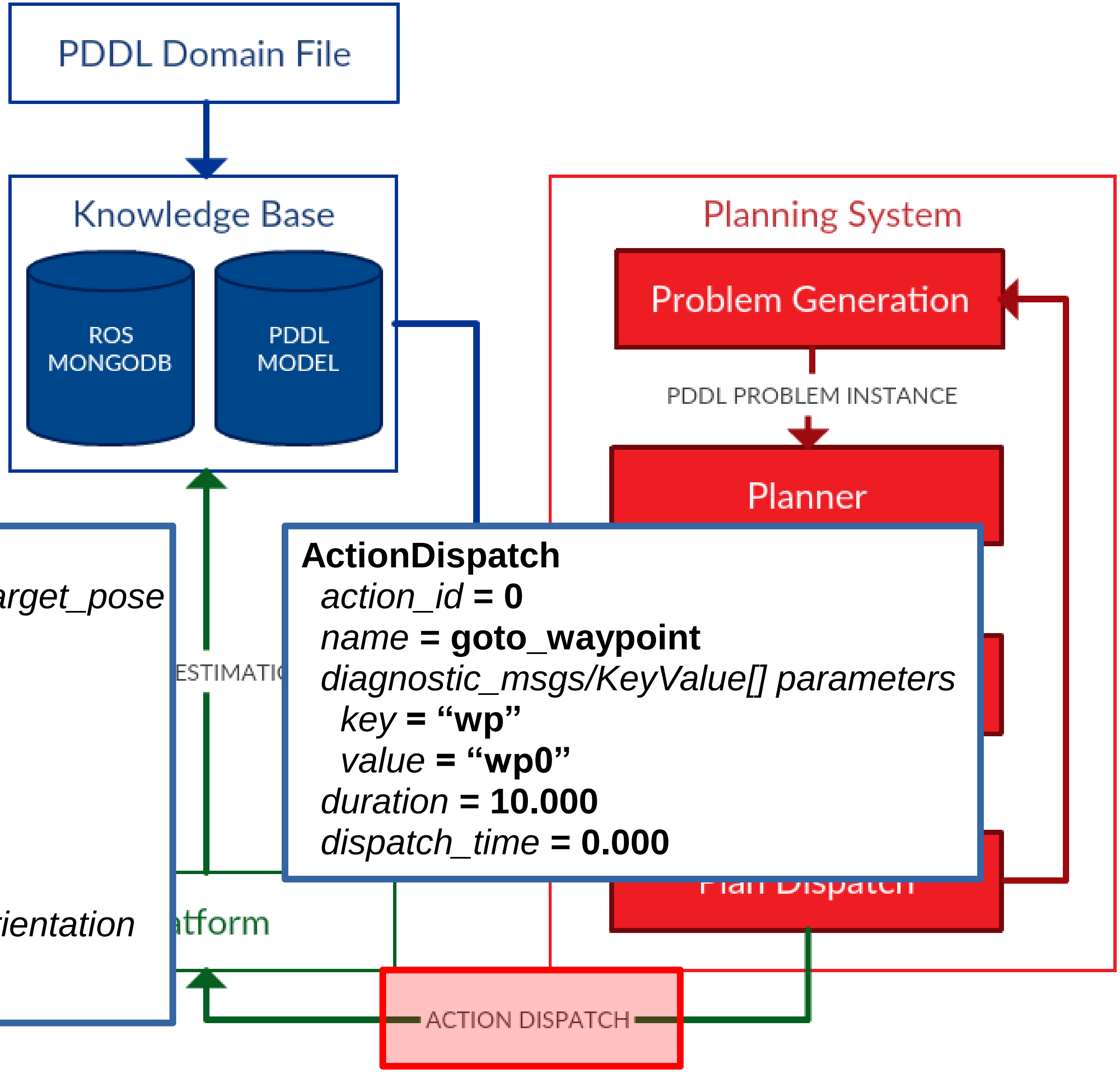
```
uint8 INSTANCE=0
uint8 FACT=1
uint8 FUNCTION=2
uint8 knowledge_type
string instance_type
string instance_name
string attribute_name
diagnostic_msgs/KeyValue[] values
  string key
  string value
float64 function_value
bool is_negative
```



ROSPlan and PNP

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

The plan is validated against a model that is continually changing and only partially



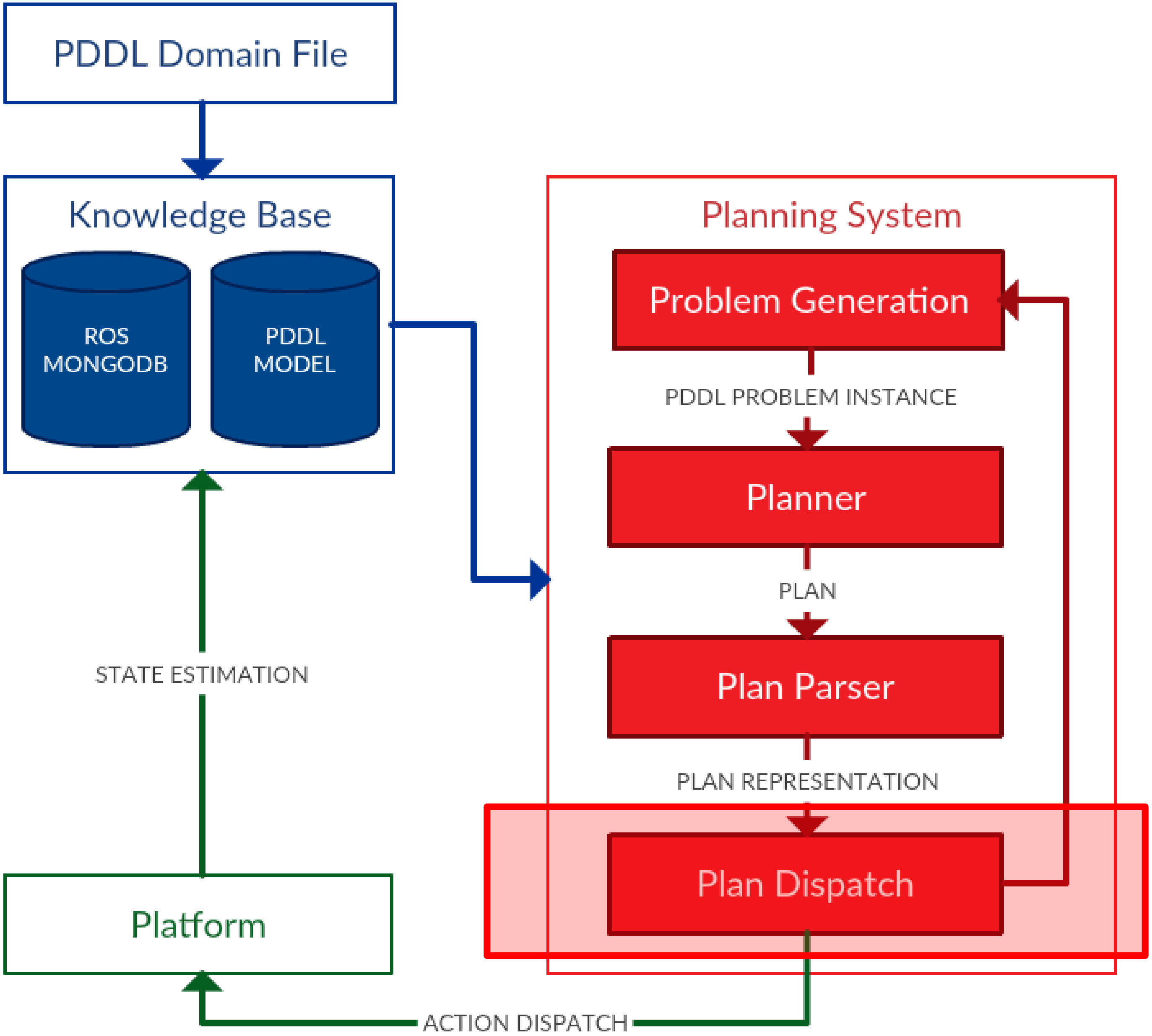
move_base/MoveBaseGoal
geometry_msgs/PoseStamped target_pose
  std_msgs/Header header
  ...
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      ...

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

ROSPlan and PNP

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

The plan is validated against a model that is continually changing and only partially sensed.



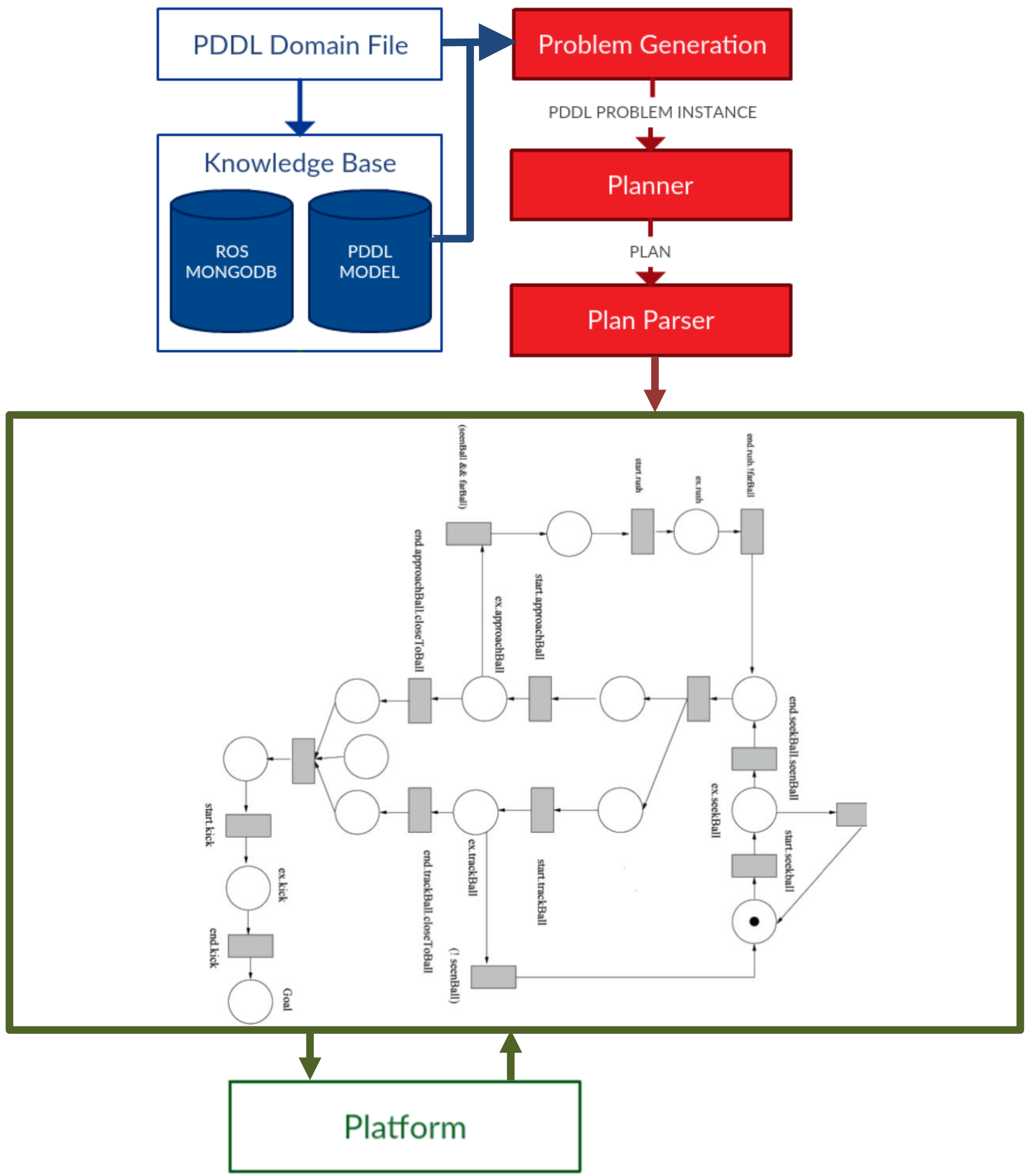
ROSPlan and PNP

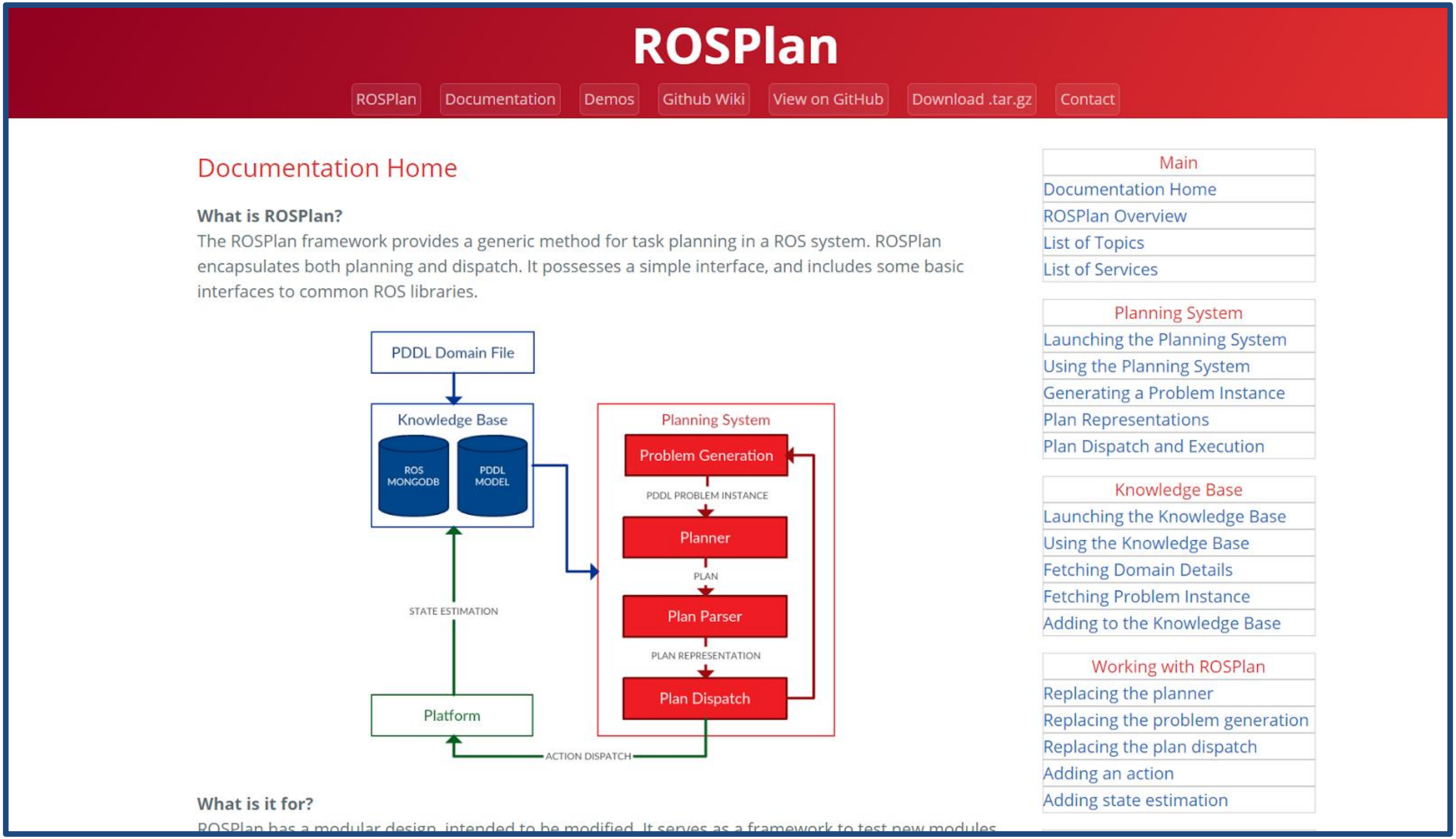
The domain model is *always* incomplete as well as inaccurate.

The plan is validated against a model that is continually changing and only partially sensed.

The RosPNP Library encapsulates both action dispatch and state updates.

In a Petri Net plan the only state estimation performed is explicit in the plan.




ROSPlan

ROSPlan | Documentation | Demos | Github Wiki | View on GitHub | Download .tar.gz | Contact

Documentation Home

What is ROSPlan?

The ROSPlan framework provides a generic method for task planning in a ROS system. ROSPlan encapsulates both planning and dispatch. It possesses a simple interface, and includes some basic interfaces to common ROS libraries.

What is it for?

ROSPlan has a modular design, intended to be modified. It serves as a framework to test new modules

ROSPlan documentation and source:
kcl-planning.github.io/ROSPlan

Petri Net Plans
Execution Framework

SAPIENZA
UNIVERSITÀ DI ROMA

Luca Iocchi
Dipartimento di Ingegneria Informatica
Automatica e Gestionale

Petri Net Plans

- High-level plan representation formalism based on Petri nets
- Explicit and formal representation of actions and conditions
- Execution Algorithm implemented and tested in many robotic applications
- Open-source release with support for different robots and development environments (ROS, Naoqi, …)

Petri Net Plans library

PNP library contains

- PNP execution engine
- PNP generation tools
- Bridges: ROS, Naoqi (Nao, Pepper)

pnp.dis.uniroma1.it

[Ziparo et al., JAAMAS 2011]

Plan representation in PNP

- Petri nets are exponentially more compact than other structures (e.g., transition graphs) and can thus efficiently represent several kinds of plans:
  - Linear plans
  - Contingent/conditional plans
  - Plans with loop
  - Policies
  - …
- PNP can be used as a general plan execution framework

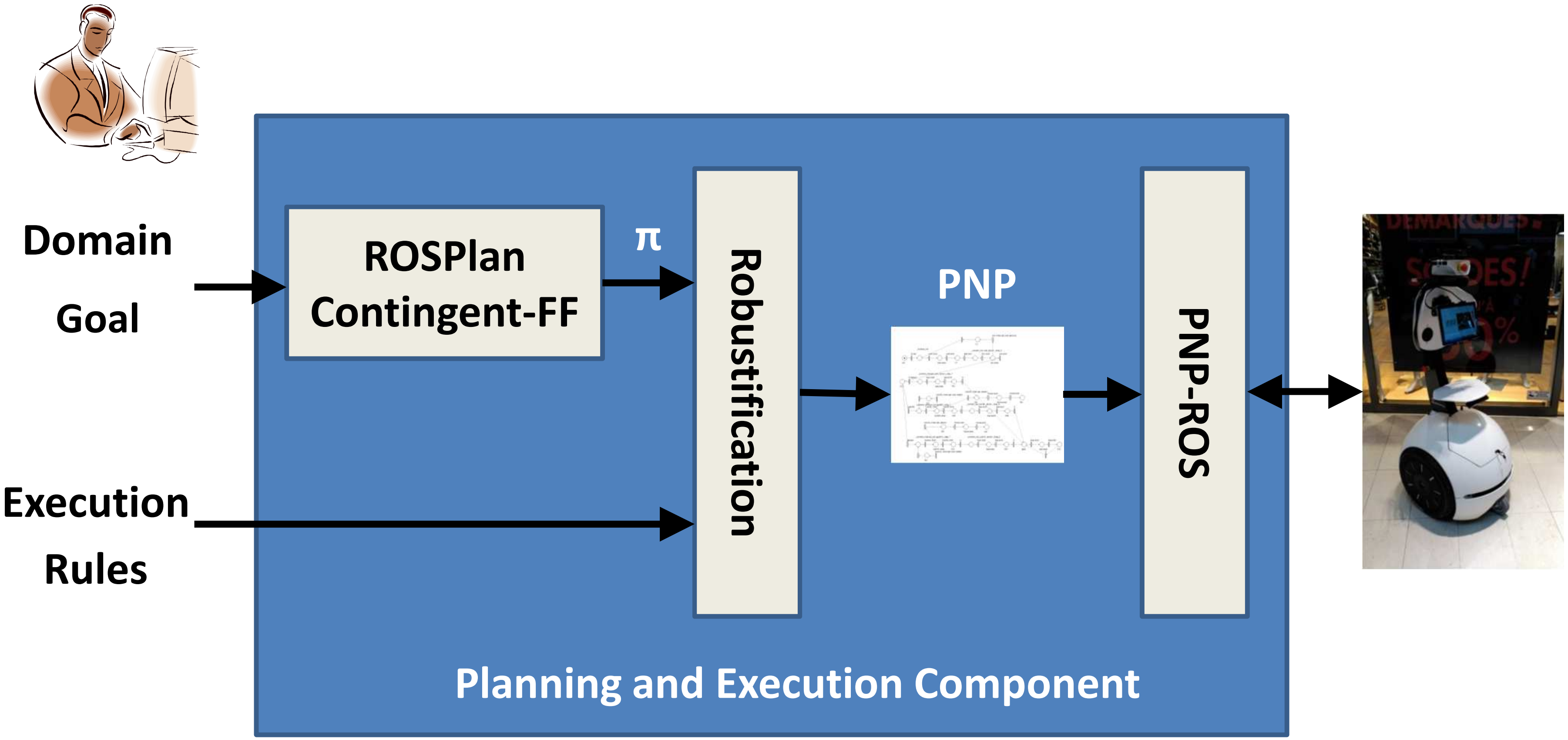
Plan traslation in PNP

- **PNPgen** is a library that translates a plan (the output of some planning system) in a PNP.
- **PNPgen** includes additional facilities to extend the generated PNP with constructs that are not available on the planning system (e.g., interrupt and recovery procedures).
- Plan formats supported:
  ROSPlan (linear/conditional), HATP, MDP policies

PNP ROS

- **PNP-ROS** is a bridge for executing PNPs in a ROS-based system.
- **PNP-ROS** uses the ROS actionlib protocol to control the execution of the actions and ROS topics and parameters to access the robot's knowledge.

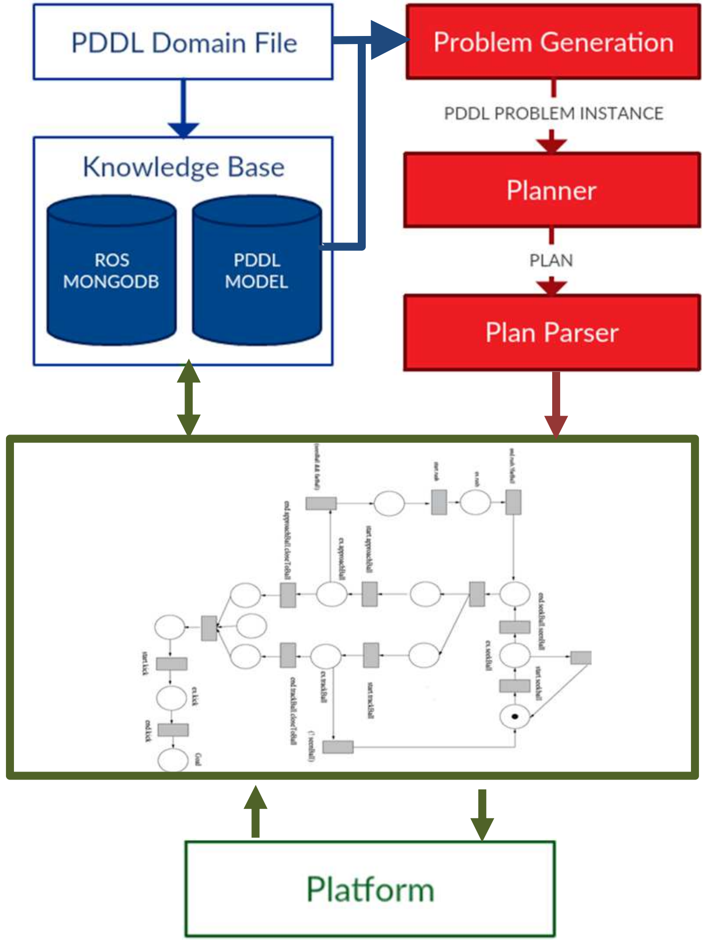
PNP execution framework



ROSPlan + PNPgen + PNP-ROS

- A proper integration of
  - Plan generation
  - Plan execution
  - ROS action execution and condition monitoring

provides an effective framework for **robot planning and execution.**



Outline

- Petri Nets
- Petri Net Plans
- Execution rules
- PNP-ROS
- Demo

Petri Net definition

Definition

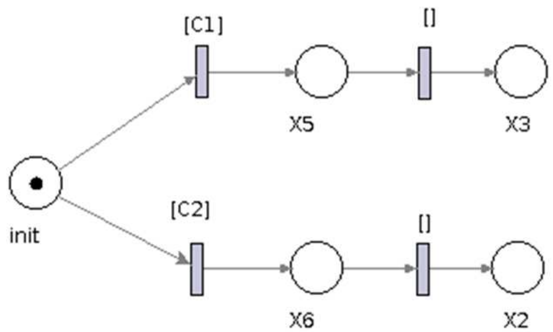
$PN = \langle P, T, F, W, M_0 \rangle$

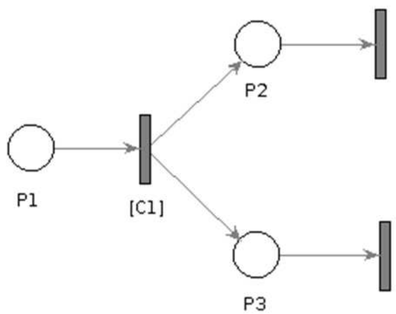
- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*.
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions*.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of edges.
- $W : F \rightarrow \{1, 2, 3, \ldots\}$ is a weight function and $w(n_s, n_d)$ denotes the weight of the edge from n_s to n_d.
- $M_0 : P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking.
- $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$

Petri Net firing rule

Definition

1. A transition t is *enabled*, if each input place p_i (i.e. $(p_i, t) \in F$) is marked with at least $w(p_i, t)$ tokens.
2. An enabled transition may or may not fire, depending on whether related event occurs or not.
3. If an enabled transition t fires, $w(p_i, t)$ tokens are removed for each input place p_i and $w(t, p_o)$ are added to each output place p_o such that $(t, p_o) \in F$.





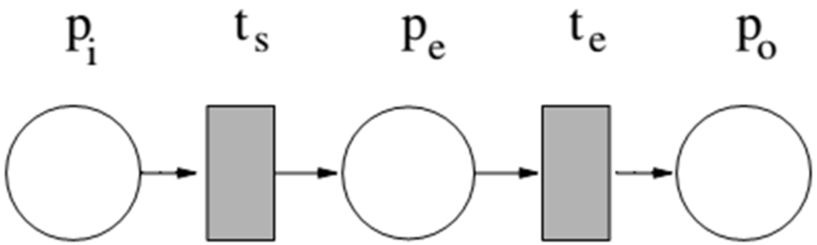
Petri Net Plans

- Petri Net Plans (PNP) are defined in terms of

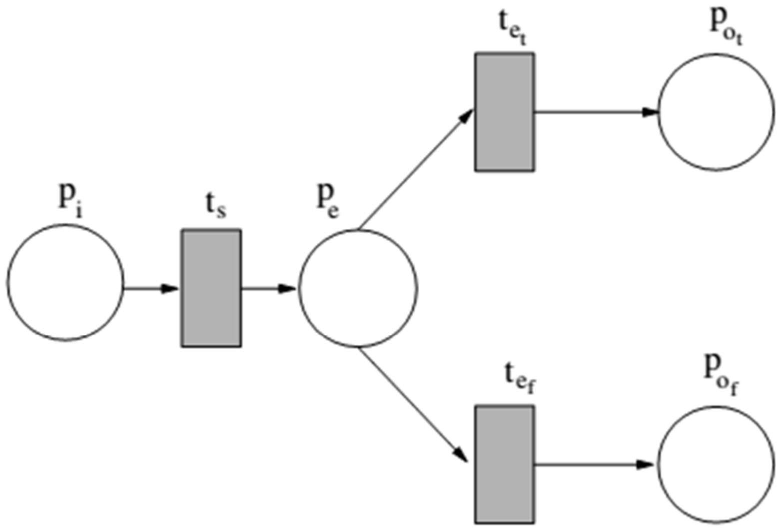
- Actions
  - ordinary actions
  - sensing actions

- Operators
  - sequence, conditional and loops
  - interrupt
  - fork/join

PNP Actions

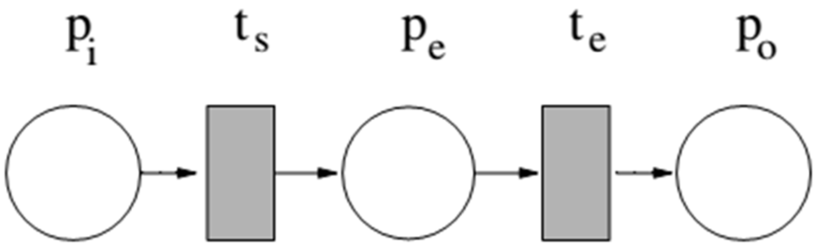


Ordinary Action

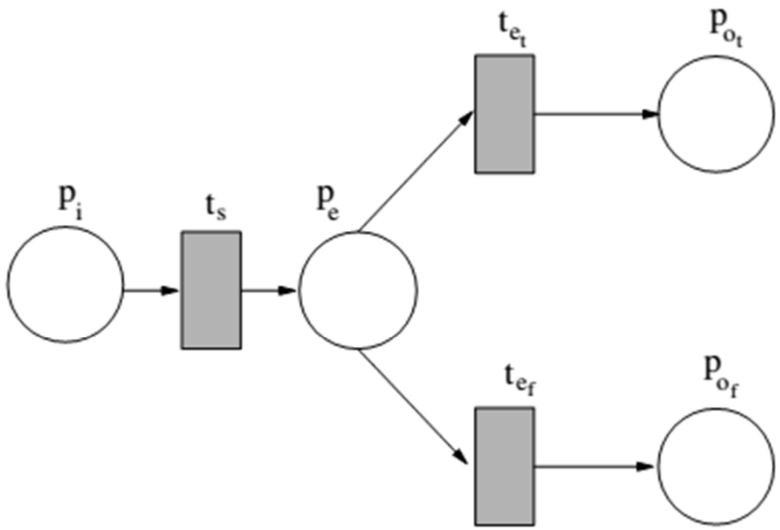


Sensing Action

PNP Actions

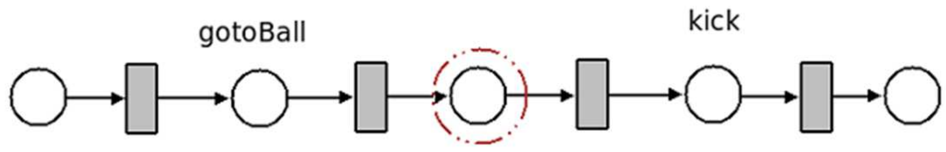


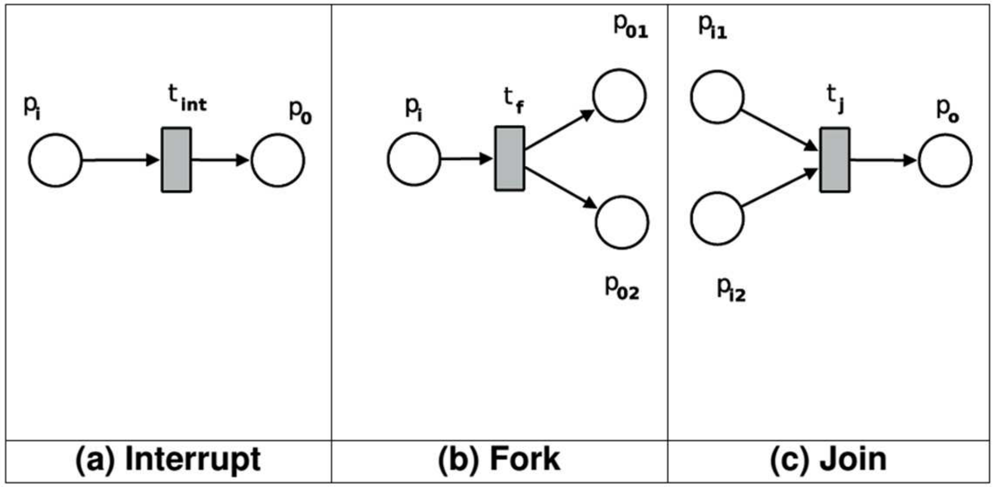
Ordinary Action



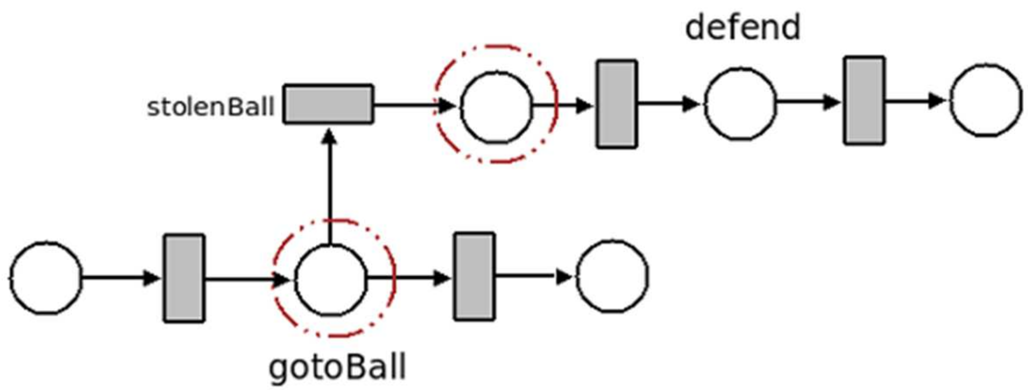
Sensing Action

PNP Operators

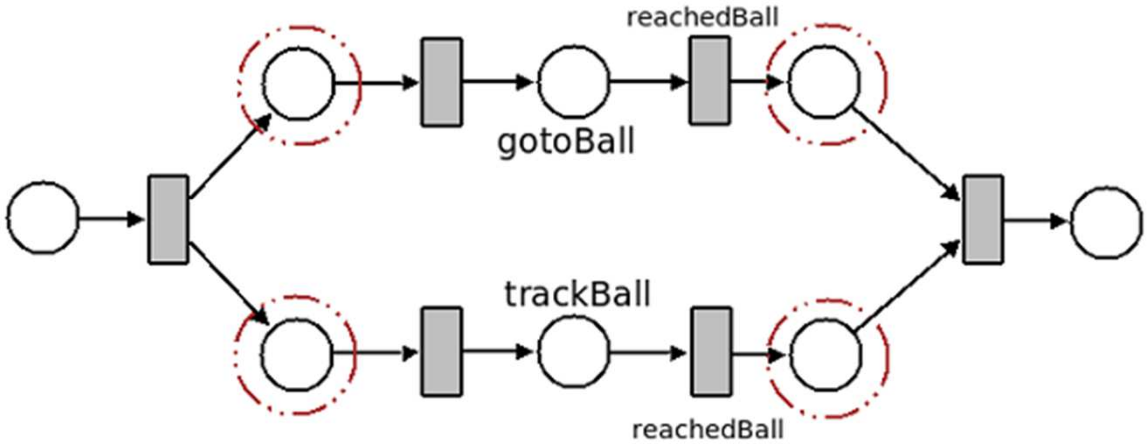




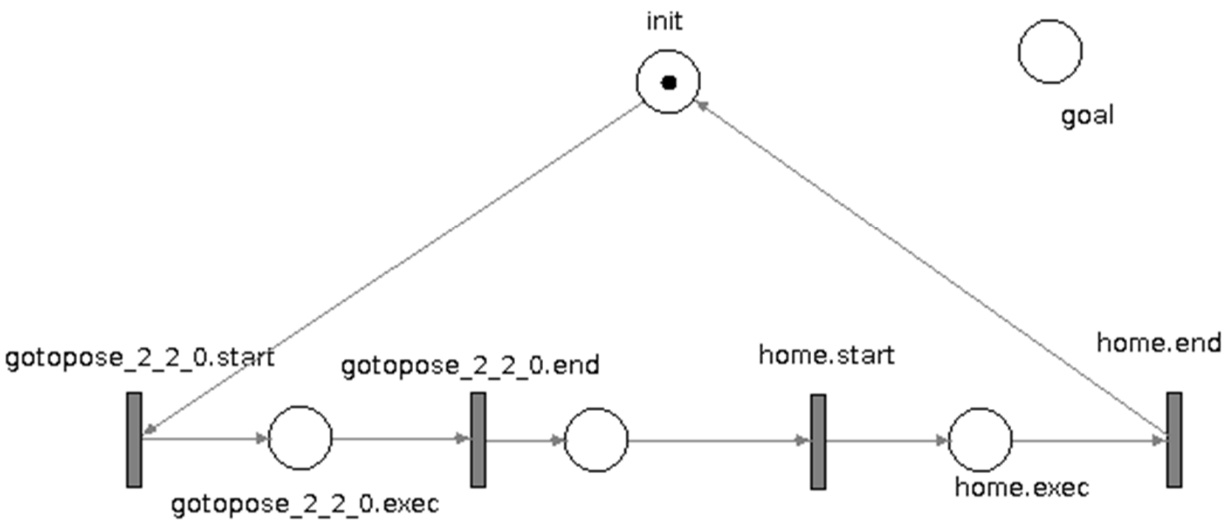
PNP interrupt



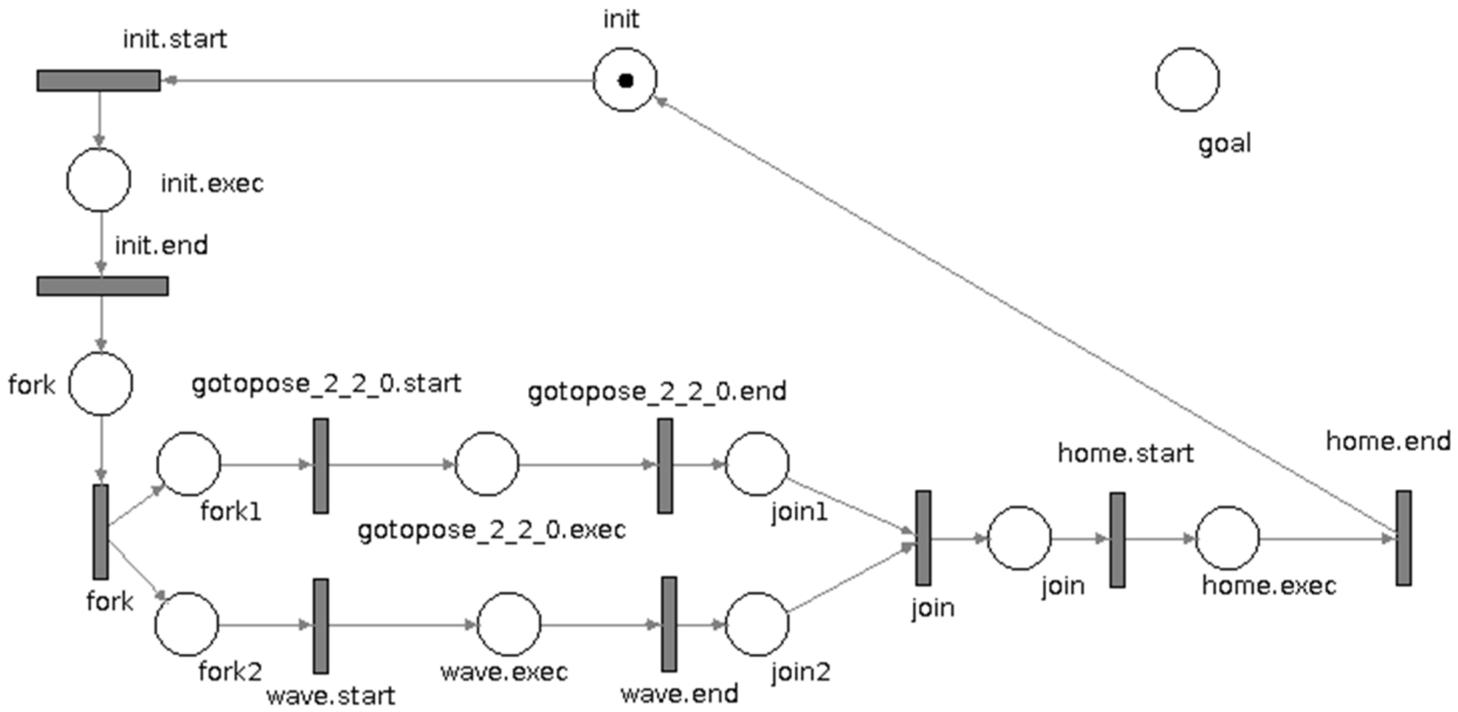
PNP concurrency



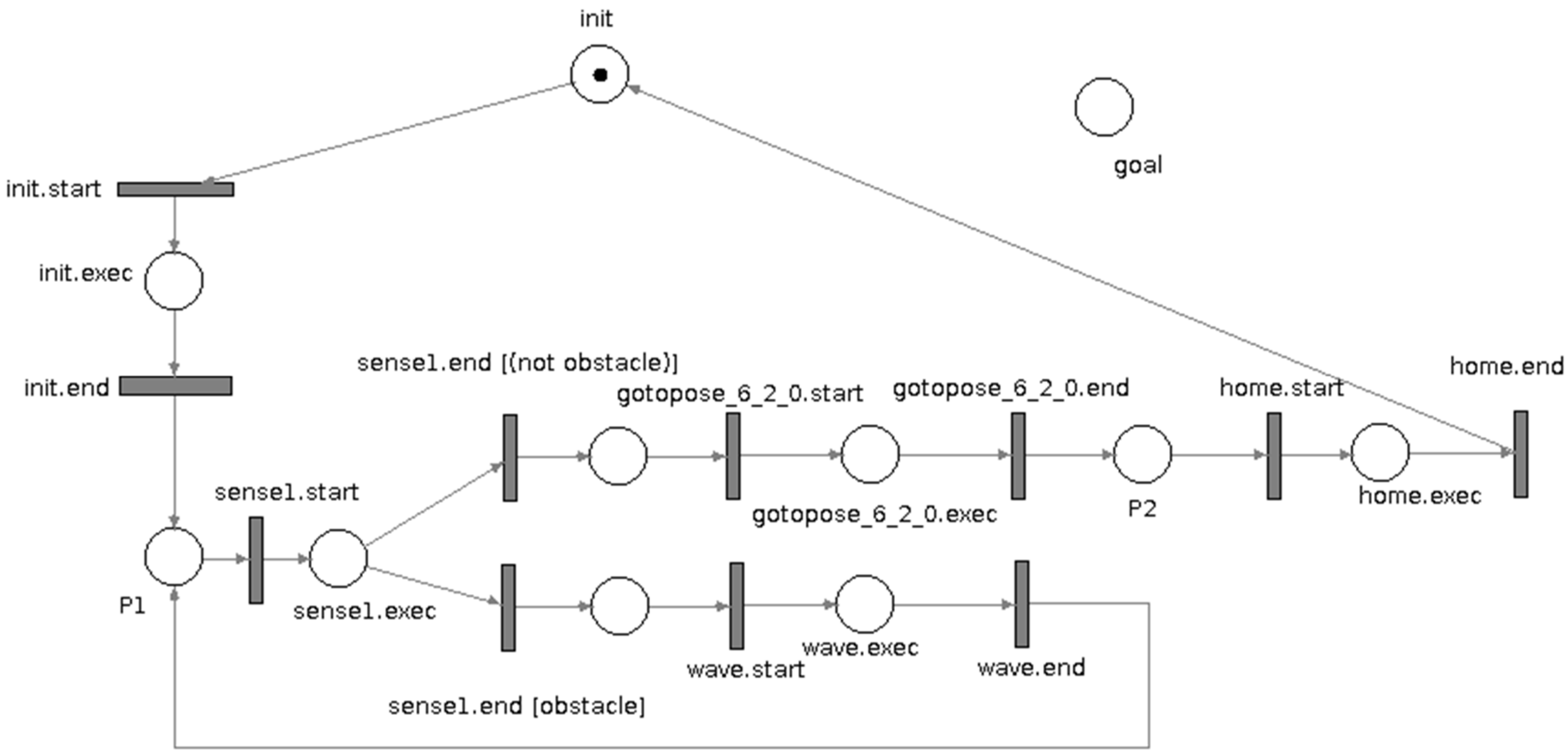
Plan 1: sequence and loop



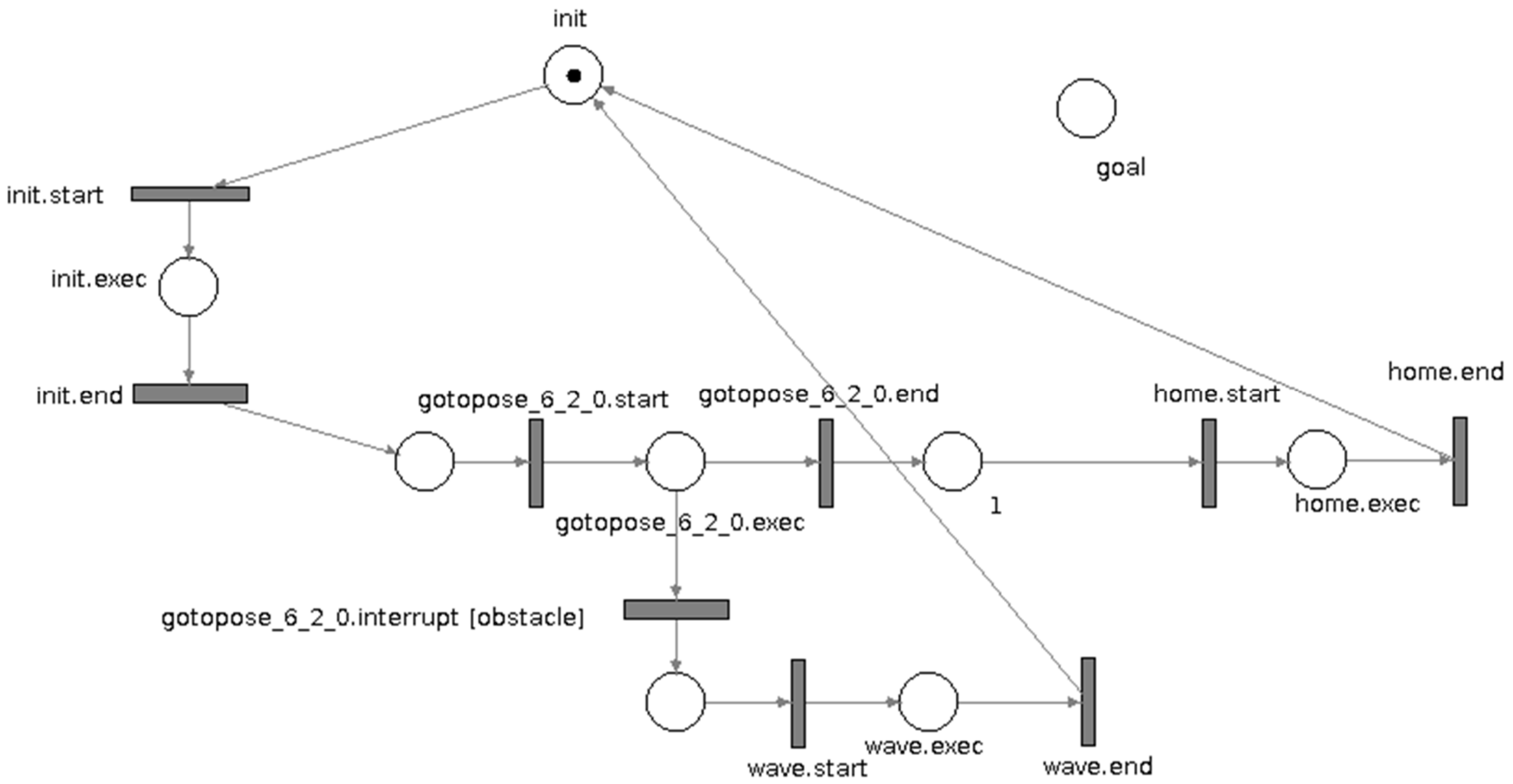
Plan 2: fork and join



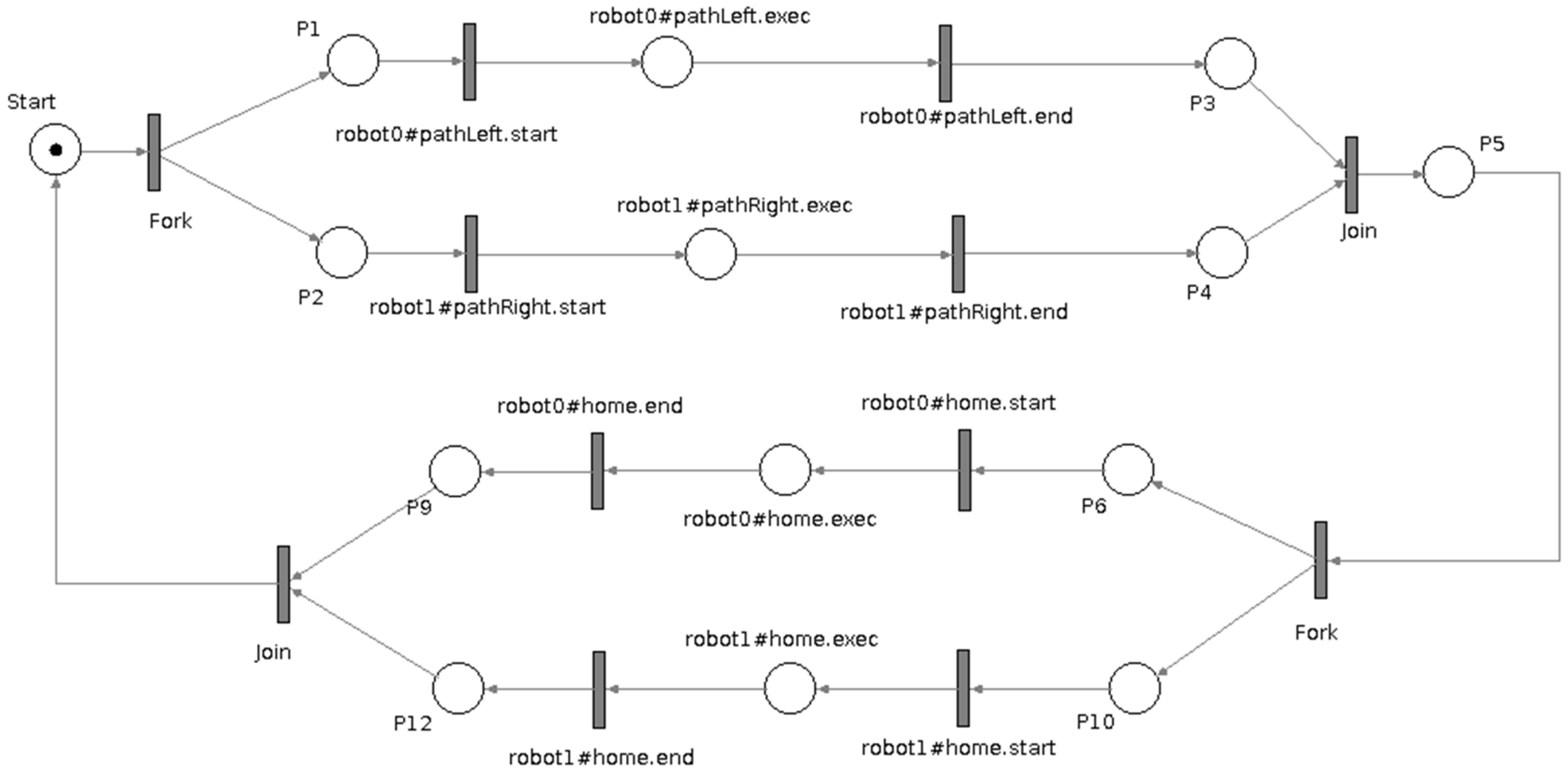
Plan 3: sensing and loop



Plan 4: interrupt



Plan 5: multi robot



PNP Execution Algorithm

procedure *execute*$(PNP\ \langle P, T, F, W, M_0, G\rangle)$

```
1: CurrentMarking = M_0
2: while CurrentMarking ∉ G do
3:     for all t ∈ T do
4:         if enabled(t) ∧ KB ⊨ t.φ then
5:             handleTransition(t)
6:             CurrentMarking = fire(t)
7:         end if
8:     end for
9: end while
```

procedure *handleTransition*(t)

```
if t.t = start then
    t.a.start()
else if t.t = end then
    t.a.end()
else if t.t = interrupt then
    t.a.interrupt()
end if
```

Correctness of PNP execution

- PNP execution is correct with respect to an operational semantics based on Petri nets and the robot's local knowledge.

Theorem

[ZI06] If a PNP can be correctly executed, then the Execution Algorithm computes a sequence of transitions $\{M_0, ..., M_n\}$, such that M_0 is the initial marking, M_n is a goal marking, and $M_i \Rightarrow M_{i+1}$, for each $i = 0, \ldots, n-1$.

PNP sub-plans

- Plans can be organized in a hierarchy, allowing for modularity and reuse

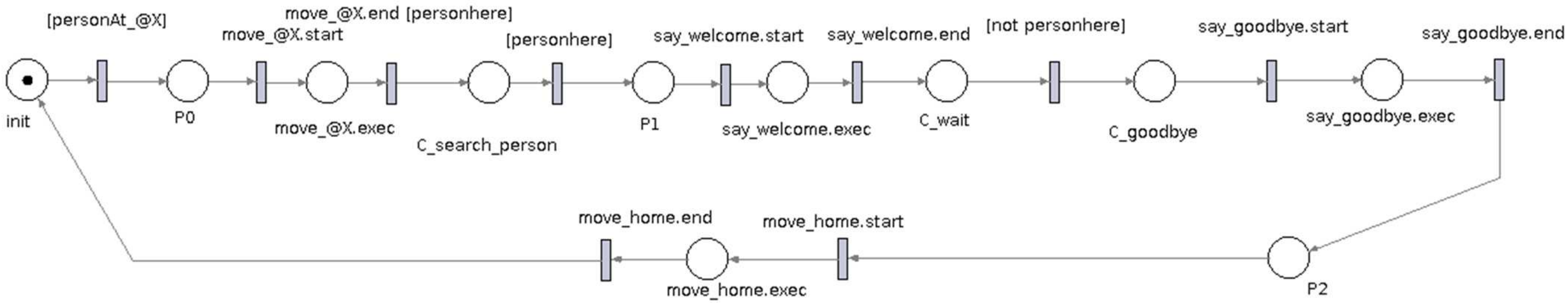
- Sub-plans are like actions:
  - when started, the initial marking is set
  - when goal marking is reached, the sub-plan ends

Plans with variables

[condition_@X] sets the value of variable X

action_@X uses the value of variable X

Example: given a condition personAt_@X, the occurrence of personAt_B115 sets the variable @X to “B115”, next action goto_@X will be interpreted as goto_B115



Execution rules

Adding to the conditional plan

- interrupt (special conditions that determine interruption of an action)
- recovery paths (how to recovery from an interrupt)
- social norms
- parallel execution

Main feature

- Execution variables are generally different from the ones in the planning domain (thus not affecting complexity of planning)

Execution rules

Examples

if personhere and closetotarget **during** goto **do** skip_action

if personhere and not closetotarget **during** goto **do** say_hello; waitfor_not_personhere; restart_action

if lowbattery **during** * **do** recharge; fail_plan

after receivedhelp **do** say_thanks

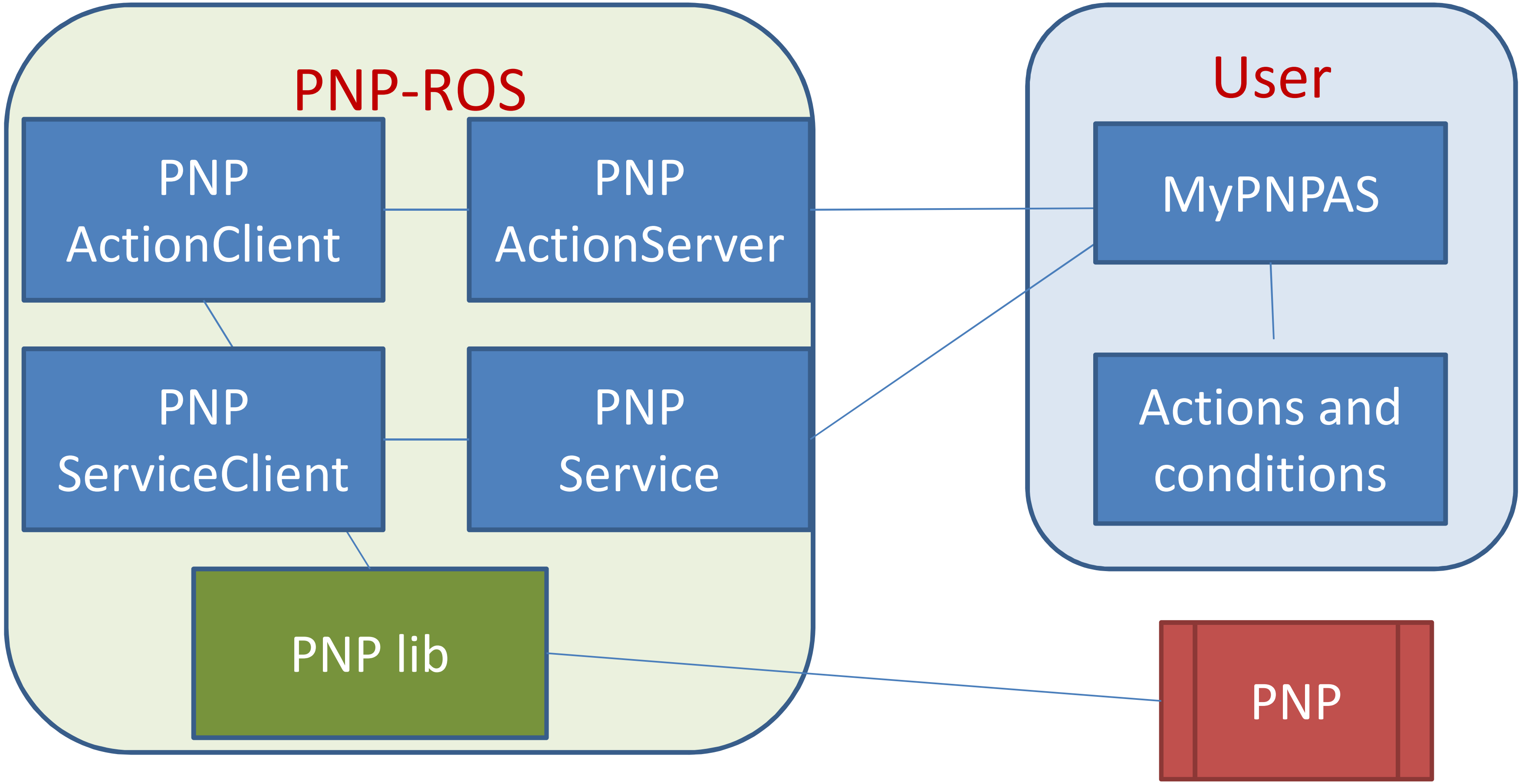
after endinteraction **do** say_goodbye

when say **do** display

PNP-ROS

- Bridge between PNP and ROS
- Allows execution of PNP under ROS using the actionlib module
- Defines a generic PNPAction and an ActionClient for PNPActions
- Defines a client service PNPConditionEval to evaluate conditions

PNP-ROS


PNP-ROS
PNP
ActionClient
PNP
ActionServer
PNP
ServiceClient
PNP
Service
PNP lib
User
MyPNPAS
Actions and
conditions
PNP

PNP-ROS

User development:

1. implement actions and conditions
2. write a PNPActionServer

PNPActionServer

```
class PNPActionServer
{
public:
   PNPActionServer();
   ~PNPActionServer();
   void start();
   // To be provided by actual implementation
   virtual void actionExecutionThread(string action_name,
                                      string action_params, bool *run);
   virtual int evalCondition(string condition); // 1: true, 0: false; -
1:unknown
}
```

PNPActionServer

```
class PNPActionServer
{
public:
    …
    // For registering action functions (MR=multi-robot version )
    void register_action(string actionname, action_fn_t actionfn);
    void register_MRaction(string actionname, MRaction_fn_t actionfn);
    …
}
```

MyPNPActionServer

```
#Include "MyActions.h"

class MyPNPActionServer : public PNPActionServer
{
   MyPNPActionServer() : PNPActionServer() {
        register_action("init",&init);
        ....

   }
}
```

MyPNPActionServer

```
PNP_cond_pub = // asynchronous conditions
        handle.advertise<std_msgs::String>("PNPConditionEvent", 10);

Function SensorProcessing
{
    …

    std_msgs::String out;
    out.data = condition;  // symbol of the condition
    PNP_cond_pub.publish(out);
}
```

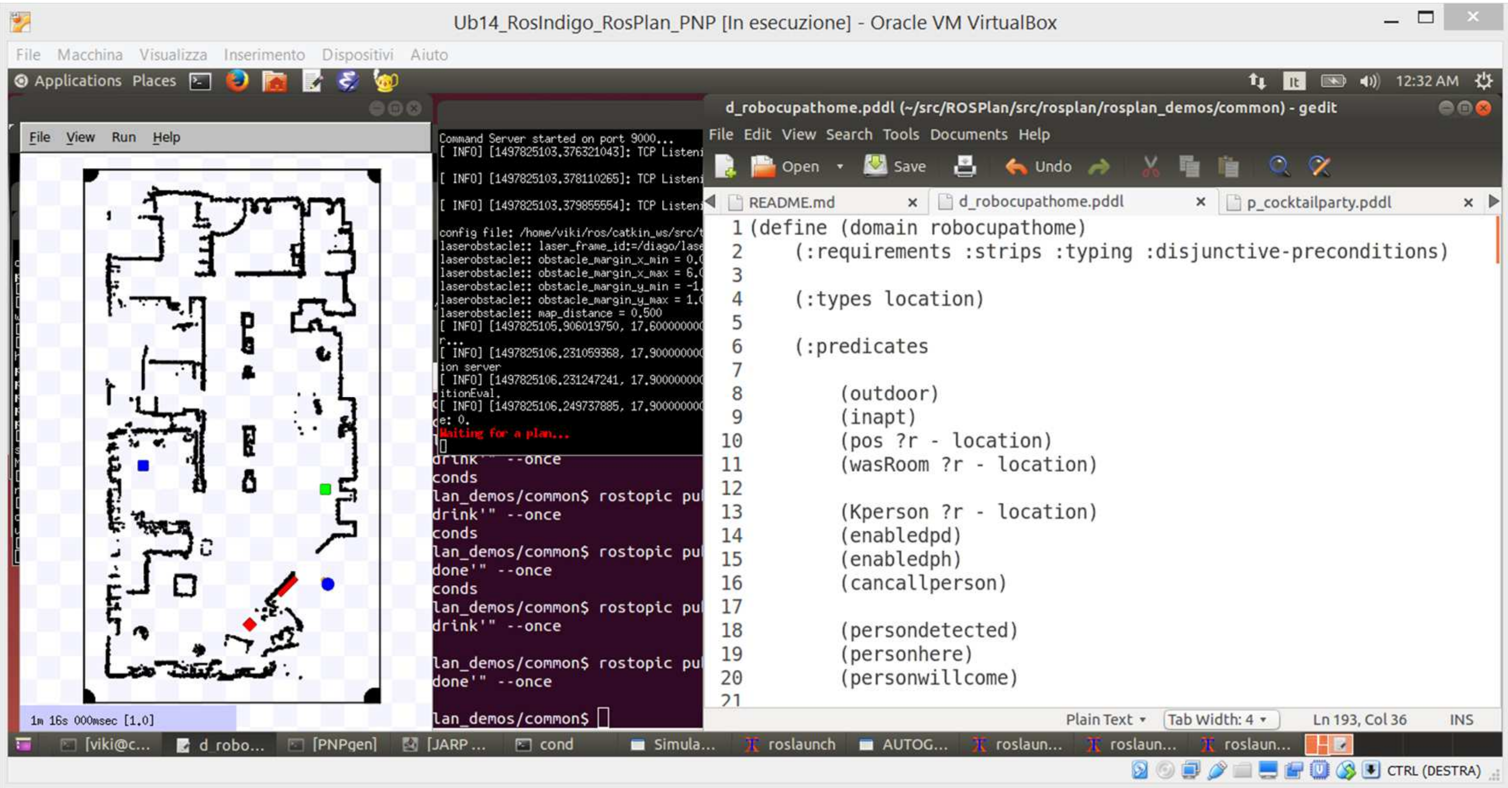
MyPNPActionServer

```
Function SensorProcessing
{
  …
  string param = “PNPconditionsBuffer/<CONDITION>”;
  node_handle.setParam(param, <VALUE {1|0}>);

}
```

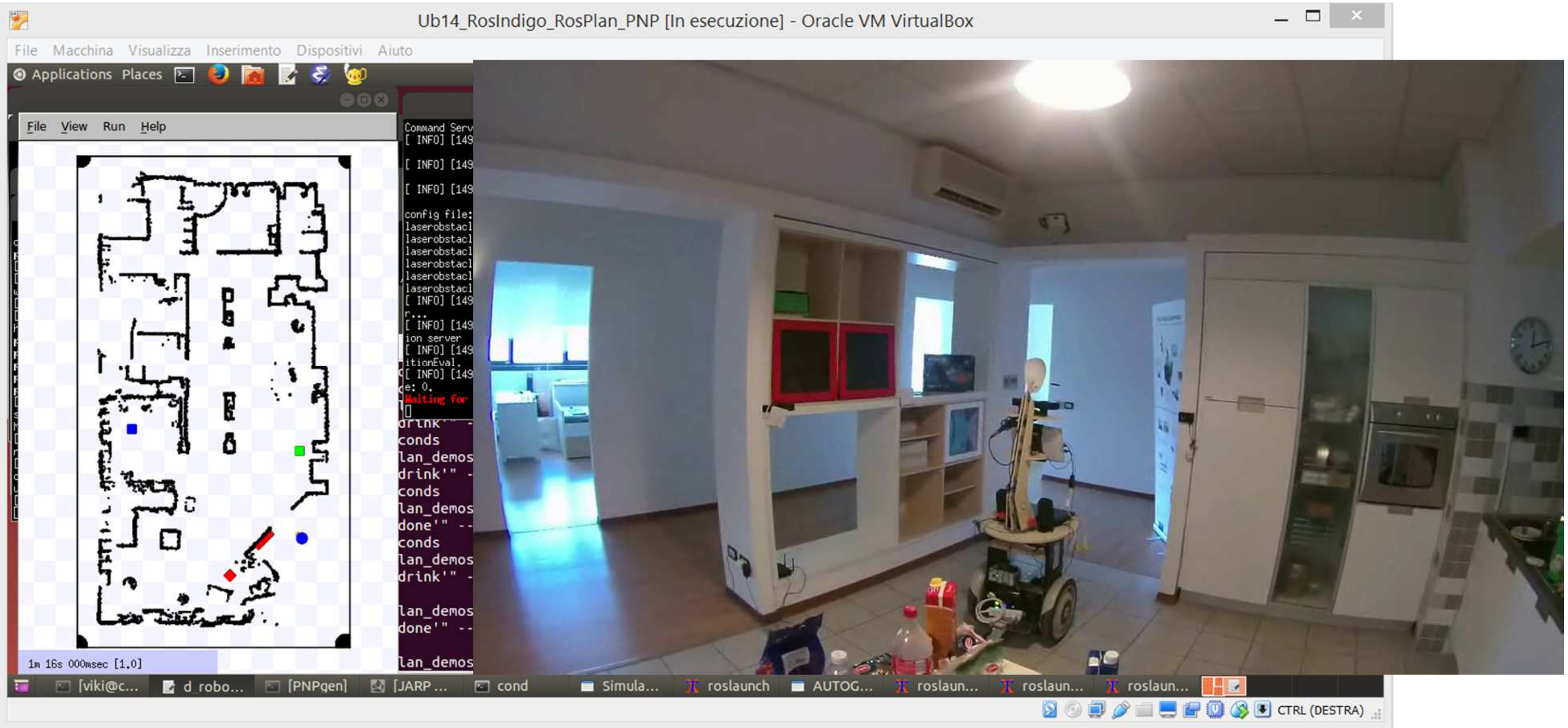
Demo

Virtual machine available in the Tutorial web site



Demo

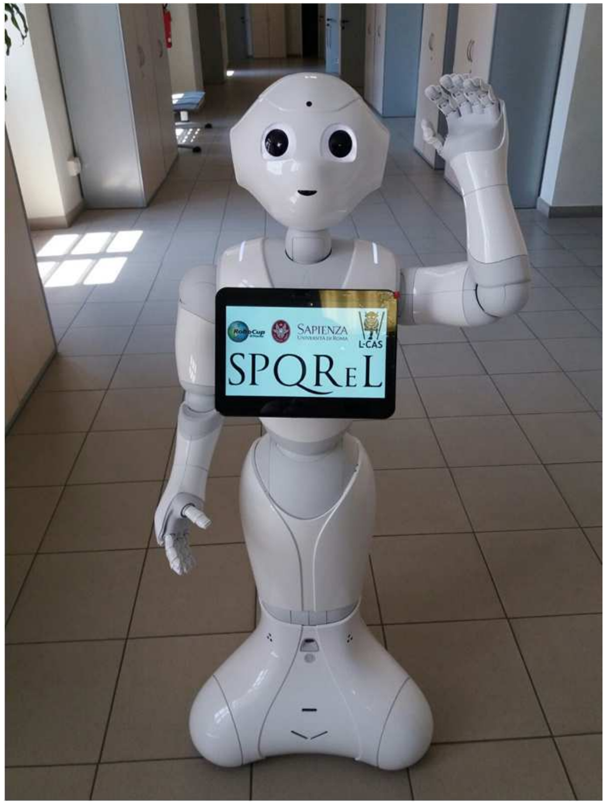
Virtual machine available in the Tutorial web site



Demo

Inspired by RoboCup@Home tasks

- RoboCup@Home domain
- Planning problems for @Home tasks
  - Navigation (rulebook 2016)
  - Cocktail Party (rulebook 2017)



NOTE: We are using this framework in our SPQReL team that will compete in RoboCup@Home 2017 SSPL

References

- **Petri Net Plans - A framework for collaboration and coordination in multi-robot systems**. V. A. Ziparo, L. Iocchi, Pedro Lima, D. Nardi, P. Palamara. Autonomous Agents and Multi-Agent Systems, vol. 23, no. 3, 2011.
- **Dealing with On-line Human-Robot Negotiations in Hierarchical Agent-based Task Planner**. E. Sebastiani, R. Lallement, R. Alami, L. Iocchi. In Proc. of International Conference on Automated Planning and Scheduling (ICAPS), 2017.
- **Short-Term Human Robot Interaction through Conditional Planning and Execution.** V. Sanelli, M. Cashmore, D. Magazzeni, L. Iocchi. In Proc. of International Conference on Automated Planning and Scheduling (ICAPS), 2017.
- **A practical framework for robust decision-theoretic planning and execution for service robots.** L. Iocchi, L. Jeanpierre, M. T. Lazaro, A.-I. Mouaddib. In Proc. of International Conference on Automated Planning and Scheduling (ICAPS), 2016.
- **Explicit Representation of Social Norms for Social Robots.** F. M. Carlucci, L. Nardi, L. Iocchi, D. Nardi. In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2015.