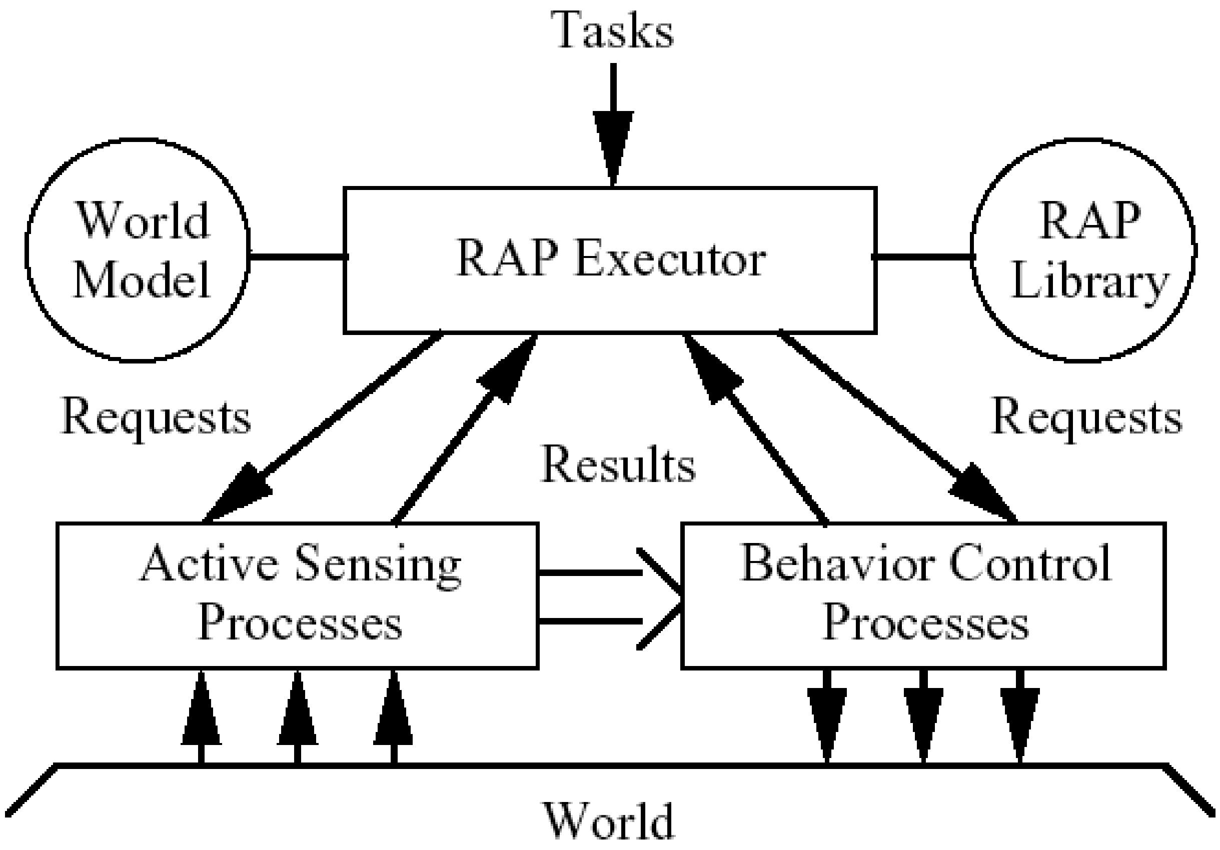
# Executive Frameworks

- RAP (http://people.cs.uchicago.edu/~firby/raps)
  - Firby, J "Task Networks for Controlling Continuous Processes", Proceedings of Artificial Intelligence Planning conference, 1994.
- TCA (http://www-2.cs.cmu.edu/afs/cs/project/TCA/release/tca.orig.html, http://www-2.cs.cmu.edu/afs/cs/project/TCA/release/tca.html)
  - Simmons, R. "Structured Control for Autonomous Robots", IEEE Transactions on Robotics and Automation, Feb 1994.
- PRS (http://www.ai.sri.com/~prs)
  - Reactive reasoning and planning: an experiment with a mobile robot, M. Georgeff and A. Lansky, in Proceedings of AAAI, 1987.
- RMPL

# Reactive Action Packages (RAP)



# Reactive Actions Packages

To cope with unpredictable details:

Choose the appropriate plan completions at execution time.

## What is a RAP?

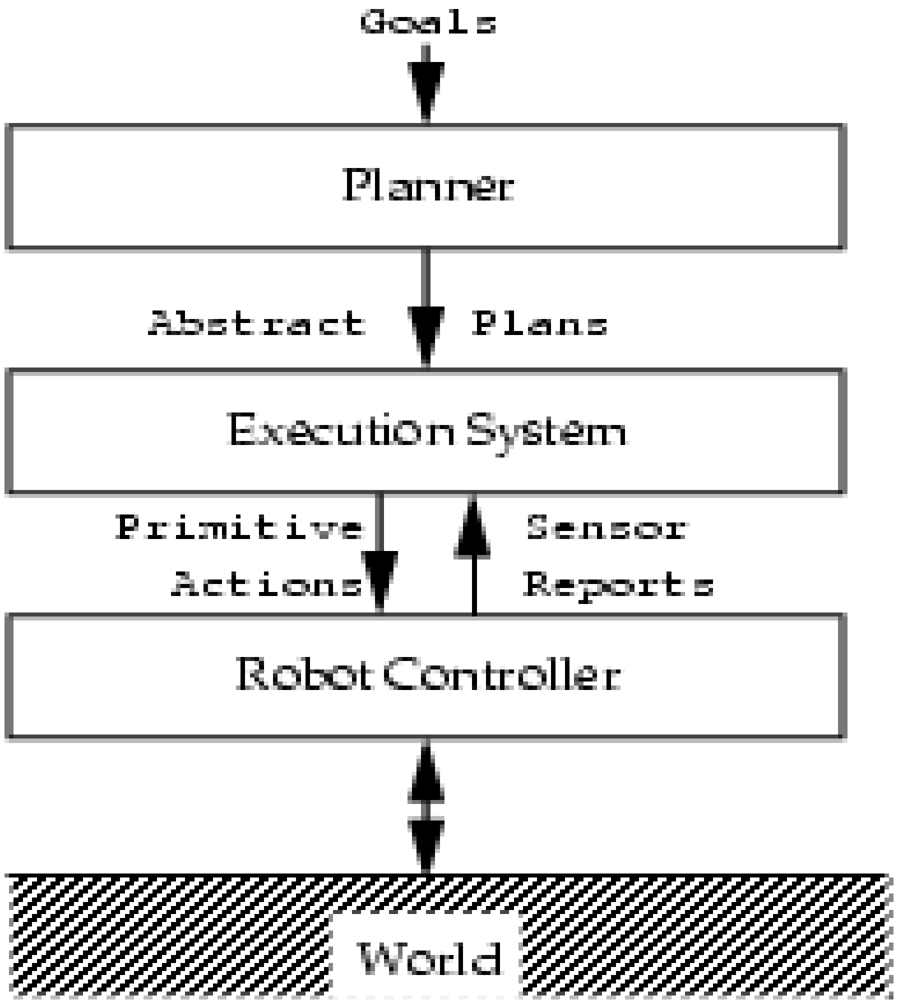
- A RAP is a representation that groups together and describes all known ways to carry out a task in different situations.
- “Situation-driven execution”

# Situation-driven execution

Tasks should have a:

- Satisfaction test
- Window of activity
- Set of execution methods

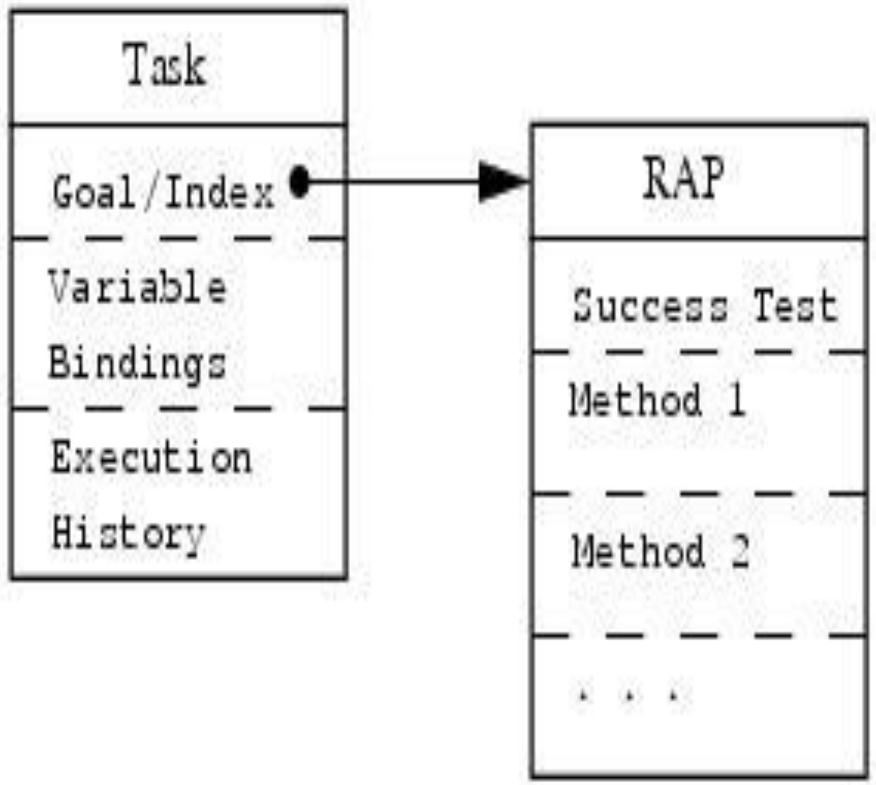
# The RAPs architecture



## Example of RAP

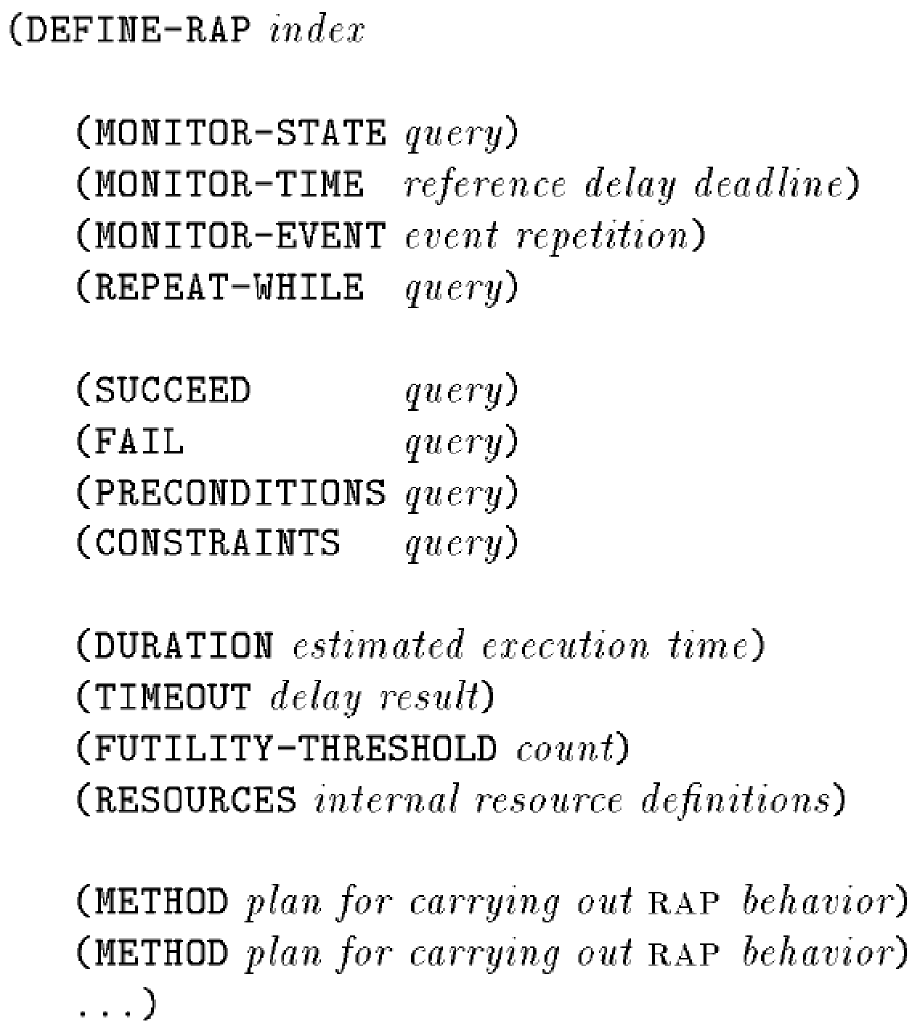
```
(DEFINE-RAP
   (INDEX  (load-into-truck ?object))
   (SUCCESS (location ?object in-truck))
   (METHOD
      (CONTEXT (or (not (size-of ?object ?size))
                   (and (size-of ?object ?size)
                        (<= ?size arm-capacity))))
      (TASK-NET
         (t1 (pickup ?object)
             ((holding arm ?object) for t2))
         (t2 (putdown ?object in-truck))))
   (METHOD
      (CONTEXT (and (size-of ?object ?size)
                    (> ?size arm-capacity)))
      (TASK-NET
         (t1 (pickup lifting-aid)
             ((holding arm lifting-aid) for t2))
         (t2 (pickup ?object)
             ((holding arm ?object) for t3))
         (t3 (putdown ?object in-truck)))))
```

# Reactive Action Packages



- RAPs are basic blocks
- Representation of task in all situations

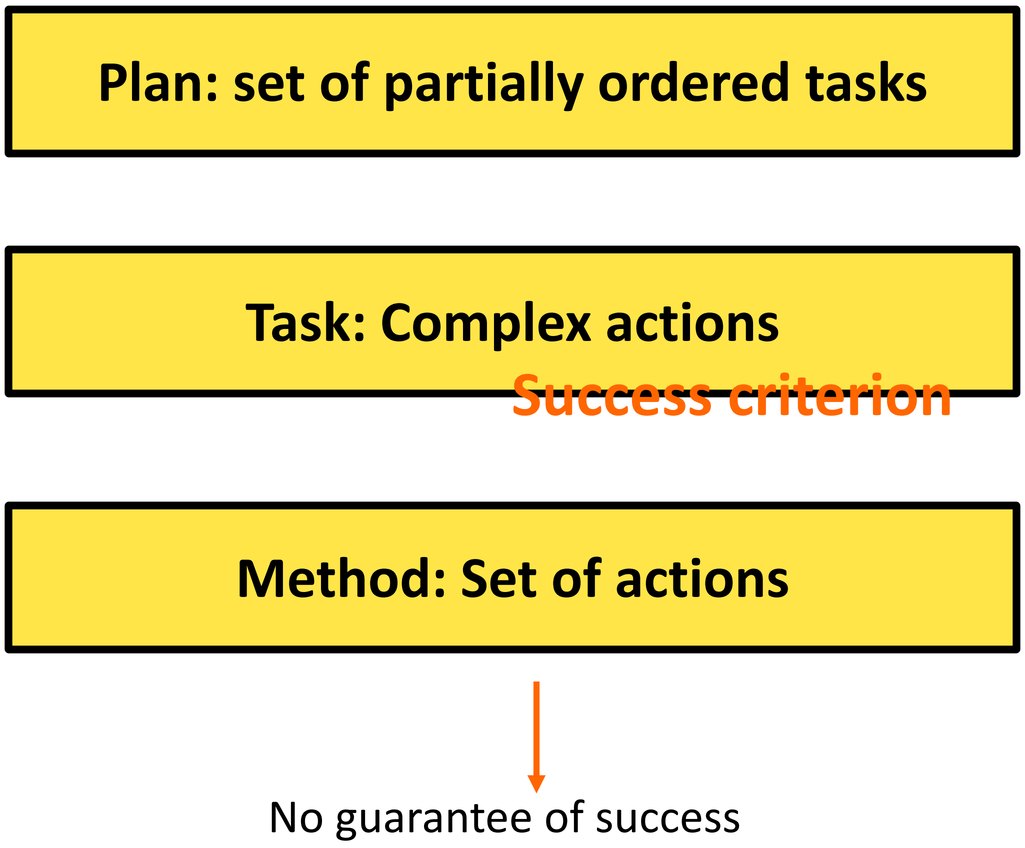
# General Structure of a RAP



```
(DEFINE-RAP index

    (MONITOR-STATE query)
    (MONITOR-TIME  reference delay deadline)
    (MONITOR-EVENT event repetition)
    (REPEAT-WHILE  query)

    (SUCCEED       query)
    (FAIL          query)
    (PRECONDITIONS query)
    (CONSTRAINTS   query)

    (DURATION estimated execution time)
    (TIMEOUT delay result)
    (FUTILITY-THRESHOLD count)
    (RESOURCES internal resource definitions)

    (METHOD plan for carrying out RAP behavior)
    (METHOD plan for carrying out RAP behavior)
    ...)
```

# Execution of RAPs

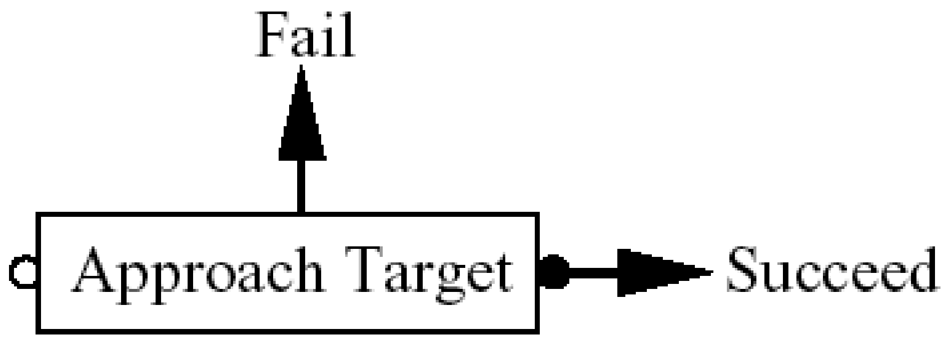
- Make a sketchy plan
- Choose a task
- Assess current situation
- Choose method
- Execute



# Task Control Architecture (TCA)

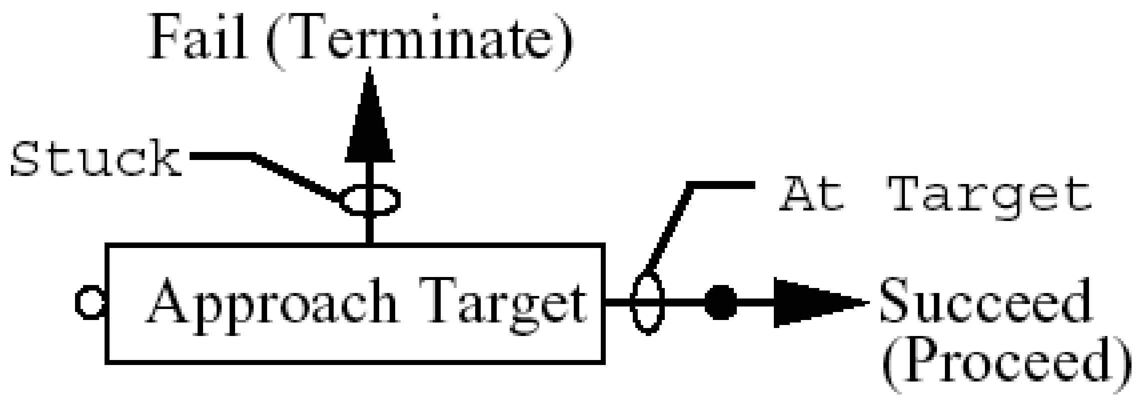
- Vertical task decomposition: several task-specific modules communicate through a central control module
- Deliberation: top-down task-subtask, resolve constraints
- Central control routes messages
  - Inform, query, command, monitor

# A Symbolic Discrete Task



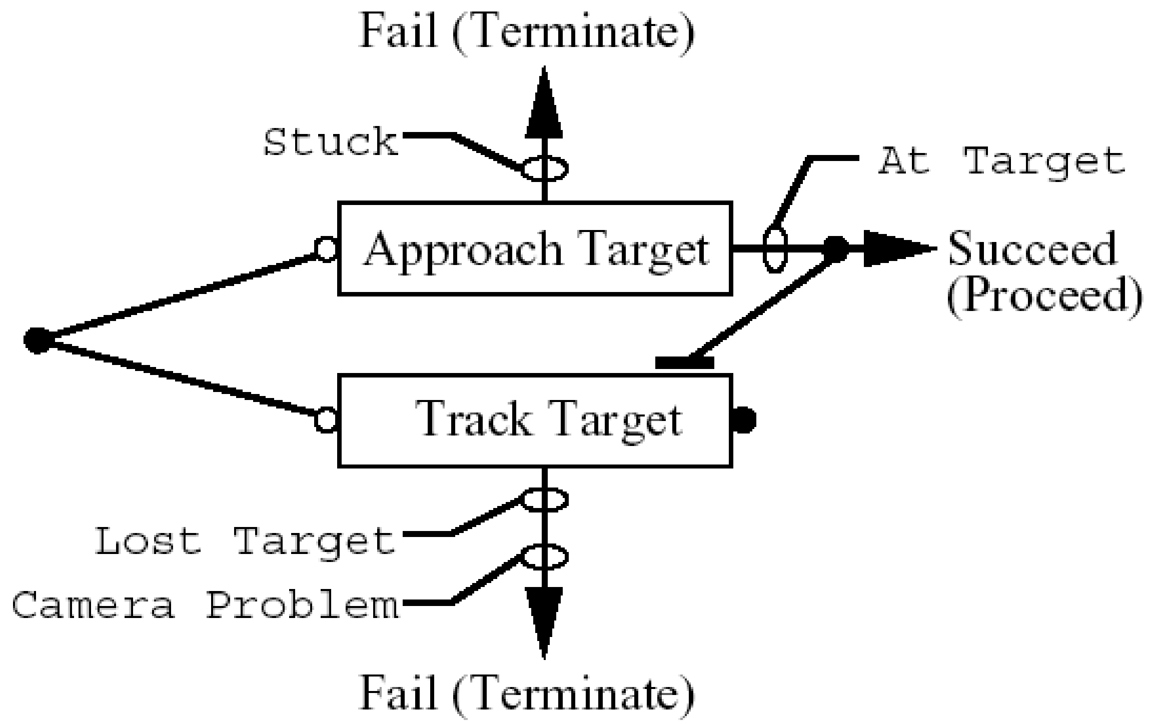
```
(task-net
  (t1 (approach-target ?target)))
```

## Waiting for a signal to proceed



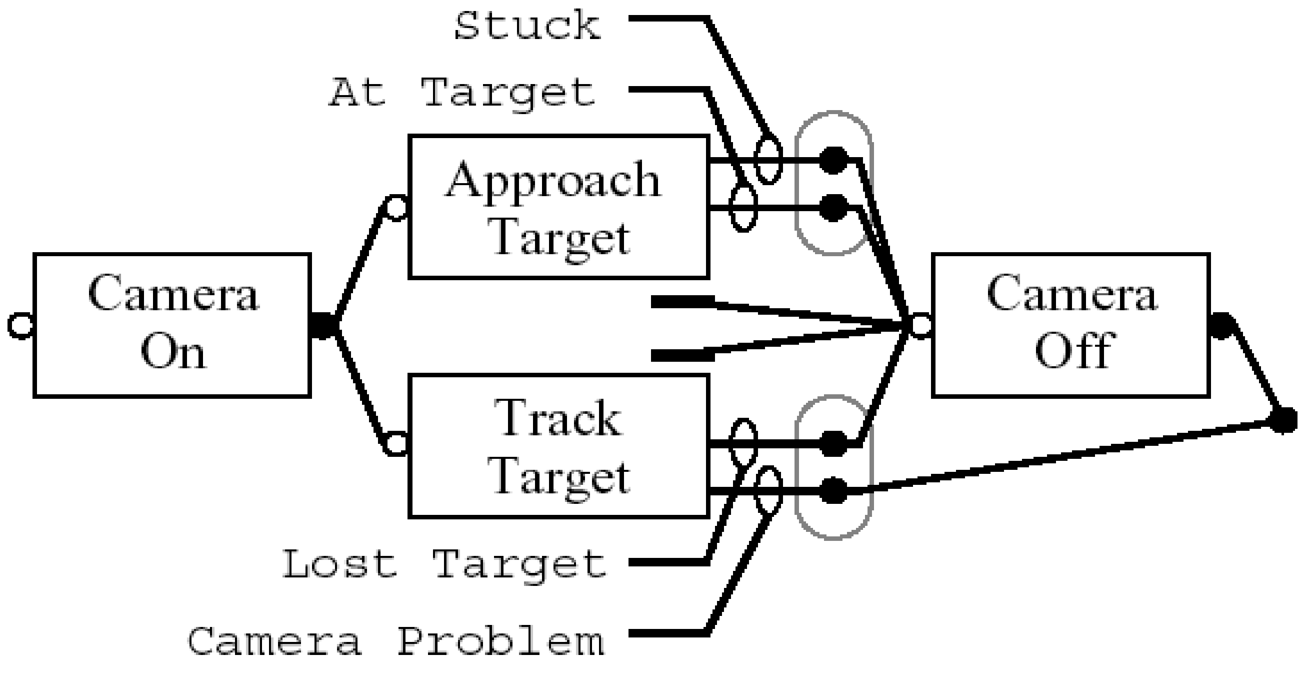
```
(task-net
  (t1 (approach-target ?target)
      (wait-for (at-target) :proceed)
      (wait-for (stuck) :terminate)))
```

# Concurrent tasks



```
(task-net
  (t1 (approach-target ?target)
      (wait-for (at-target) :proceed)
      (wait-for (stuck) :terminate))
  (t2 (track-target ?target)
      (wait-for (lost-target) :terminate)
      (wait-for (camera-problem) :terminate)
      (until-end t1)))
```

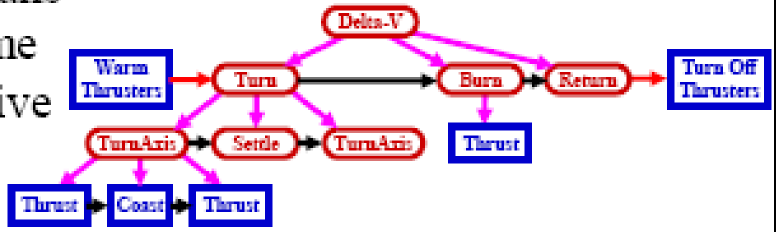
# More Complex Task Networks



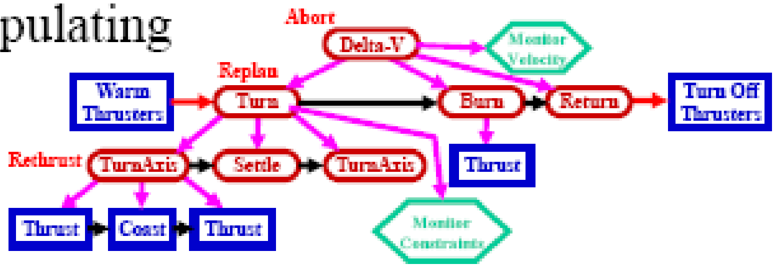
```
(task-net
  (t0 (camera-on) (wait-for :success t1) (for t2))
  (t1 (approach-target ?target)
      (wait-for (at-target) t3)
      (wait-for (stuck) t3)
      (until-start t3))
  (t2 (track-target ?target)
      (wait-for (lost-target) t3)
      (wait-for (camera-problem) :terminate)
      (until-start t3))
  (t3 (camera-off)))
```

# **TDL** (Simmons 1998)

- TDL Maintains and Coordinates *Task Trees*
  - *Execution trace* of hierarchical plans
    - Created dynamically at run time
    - Can be conditional and recursive
  - Temporal constraints (partially) order task execution
  - Planning and sensing treated as schedulable activities; Concurrent planning, sensing, and execution

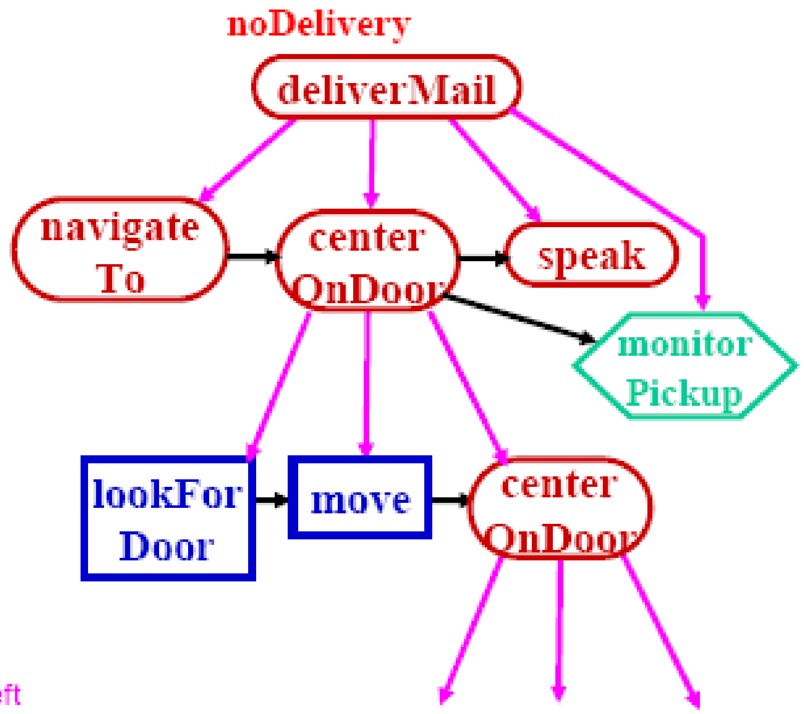


- Task Trees Augmented with *Reactive* Elements
  - Task-specific execution monitors
  - Context-dependent, hierarchical exception handlers
- Replan by Analyzing and Manipulating Task Trees
  - Terminate subtrees
  - Add new nodes and/or temporal constraints

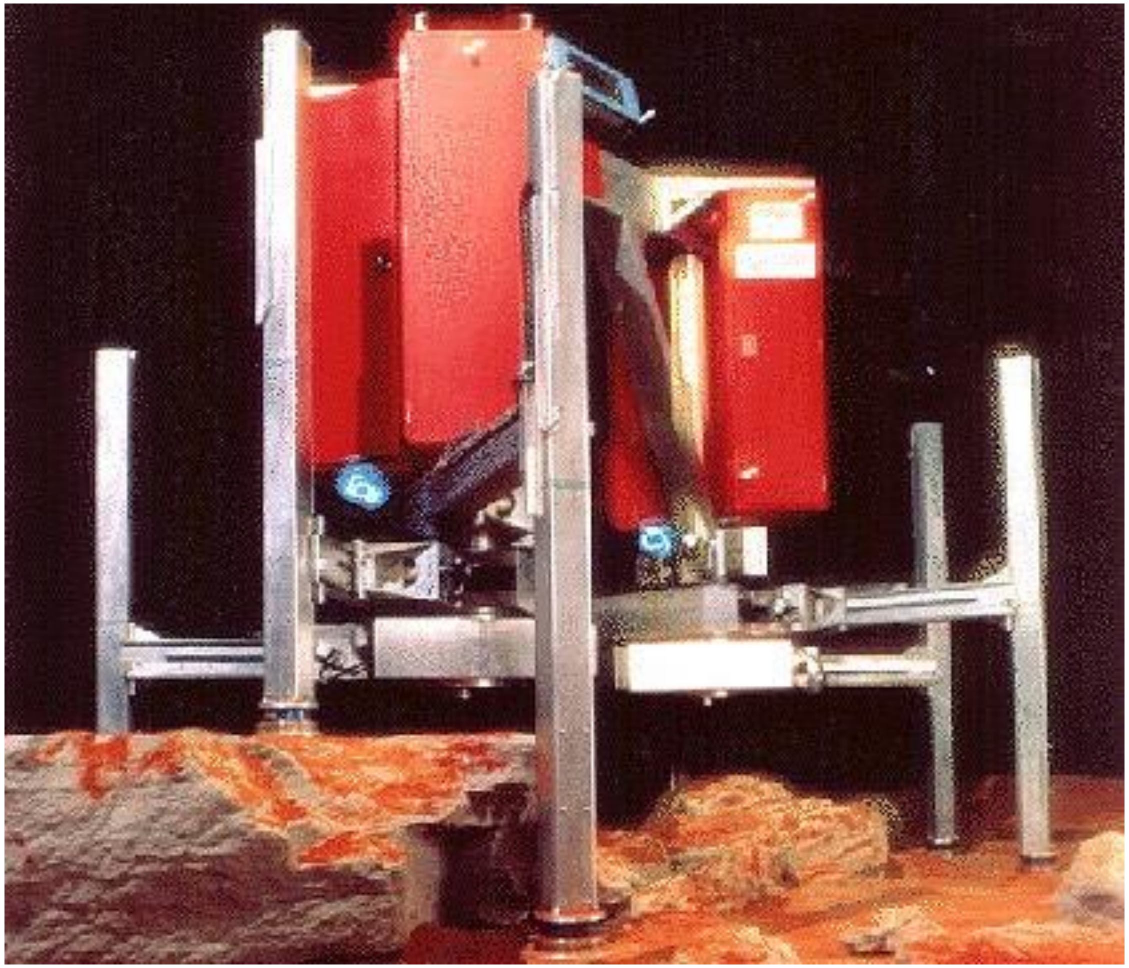


## TDL

```
Goal deliverMail (int room)
        Exception Handler noDelivery
{
        double x, y;
        getRoomCoordinates(room, &x, &y);
        spawn navigateTo(x, y);
         spawn centerOnDoor(x, y)
            with sequential execution previous,
                    terminate in 30.0;
         spawn speak("Xavier here with your mail")
            with sequential execution centerOnDoor,
                    terminate at monitorPickup completed;
         spawn monitorPickup()
            with sequential execution centerOnDoor;
}

Goal centerOnDoor(double x, double y)
{
        int whichSide;
         spawn lookForDoor(&whichSide) with wait;
        if (whichSide != 0) {
            if (whichSide < 0) spawn move(-10); // move left
            else spawn move(10);  // move right
            spawn centerOnDoor(x, y)
               with disable execution until
                    previous execution completed;
          }
}
```



## Ambler Walking Robot



# Ambler Modules

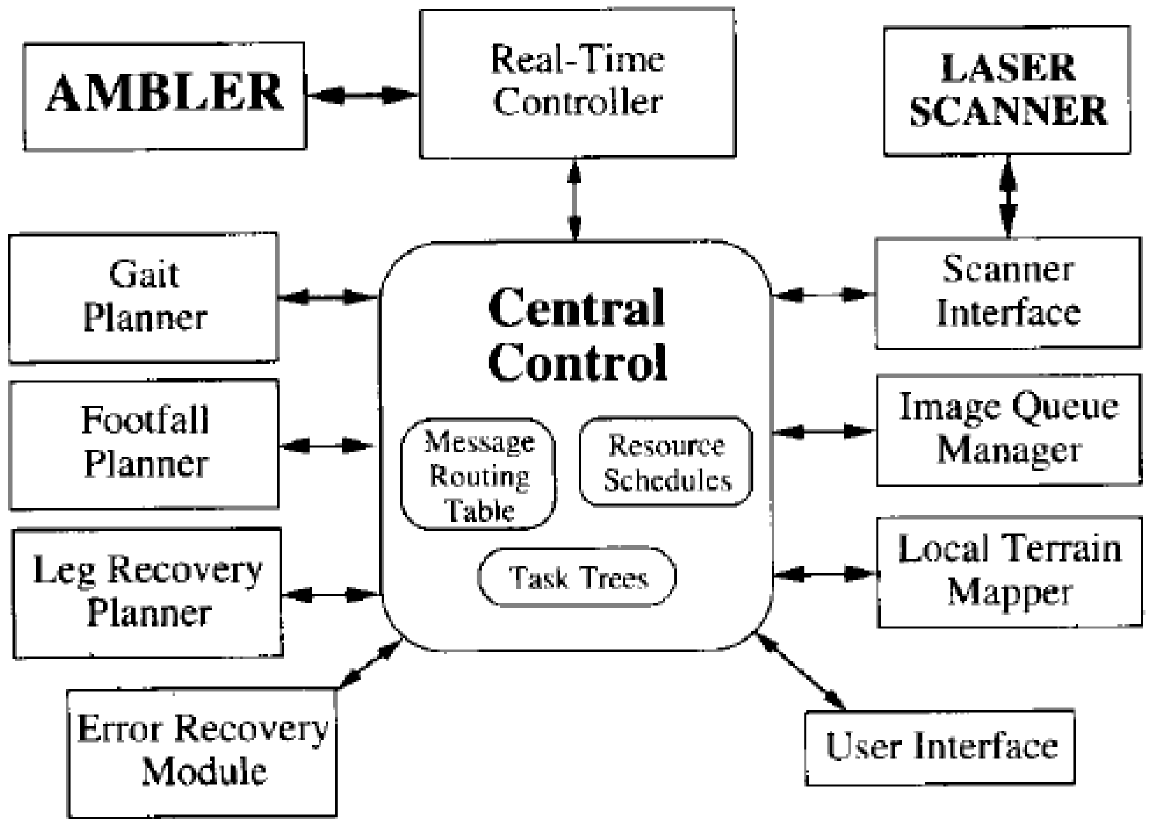


Fig. 3. Modules for the Ambler walking system.

# Ambler Task Tree

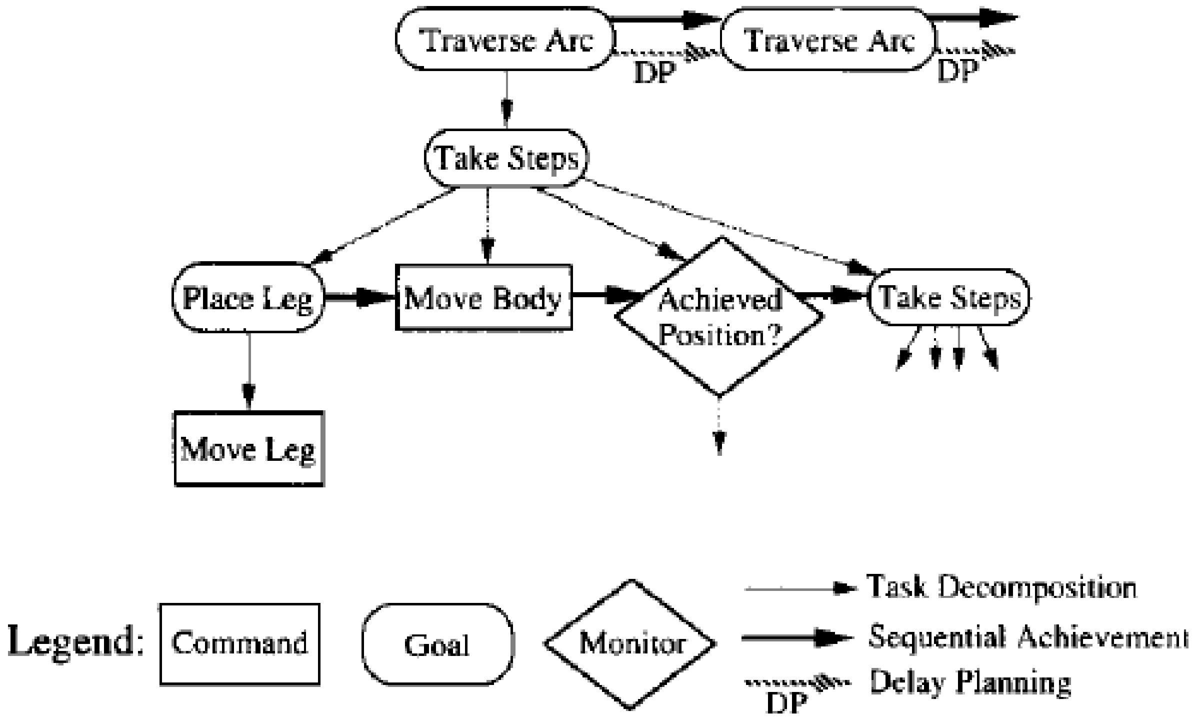


Fig. 4. Task tree for autonomous walking.

## TCA: Monitoring

- Central control traverses tree and handles messages:
  - asks gait planner to traverse arc,
  - gait planner asks terrain mapper for elevation map in order to take steps
  - Gait planner asks leg recovery planner to place leg, move leg, move body,
  - Gait planner activates monitor whether achieved position

# TCA: Control

- Ordering and temporal constraints
- Delay planning constraint: goal cannot be issued until previous task achieved
  - Can do place leg planning while monitoring achieve position
- Exception handling: error recovery modules examine and modify task trees
  - Eg: if position not achieved, add take steps subtask

# Ambler Planning and Execution

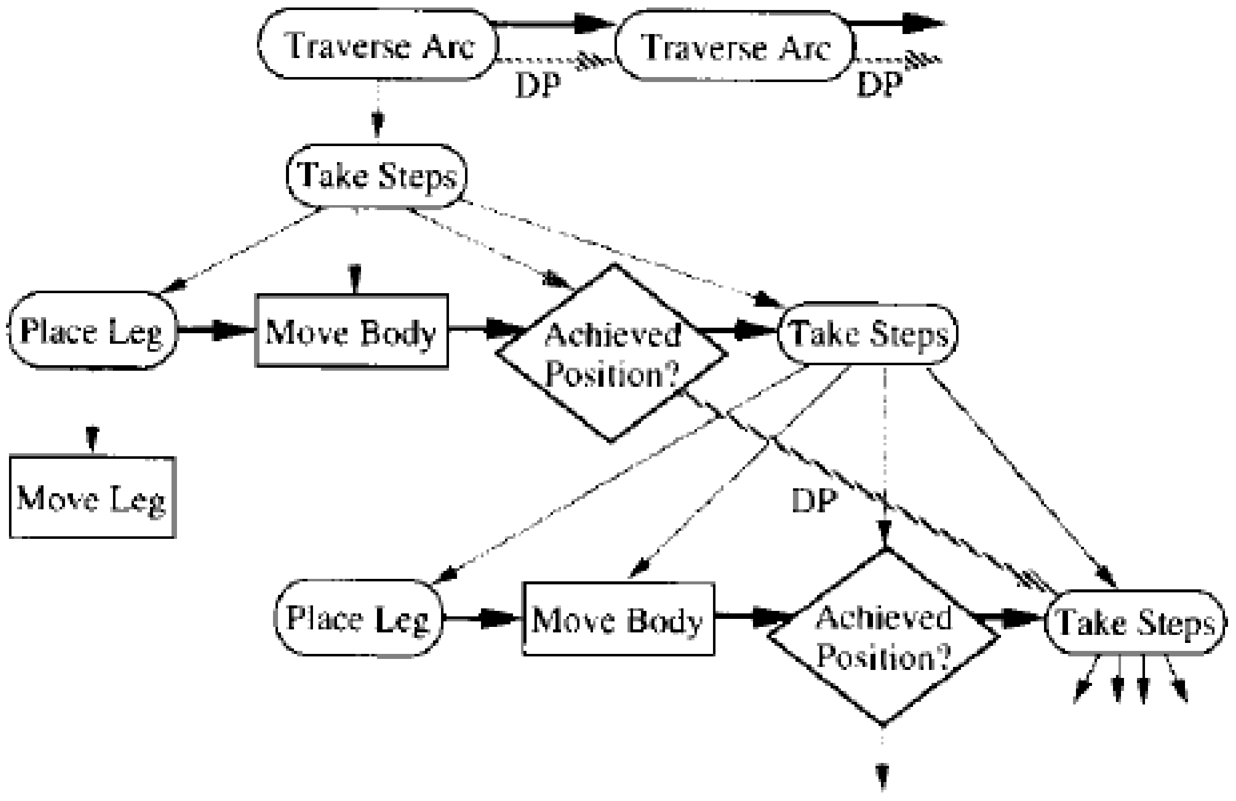
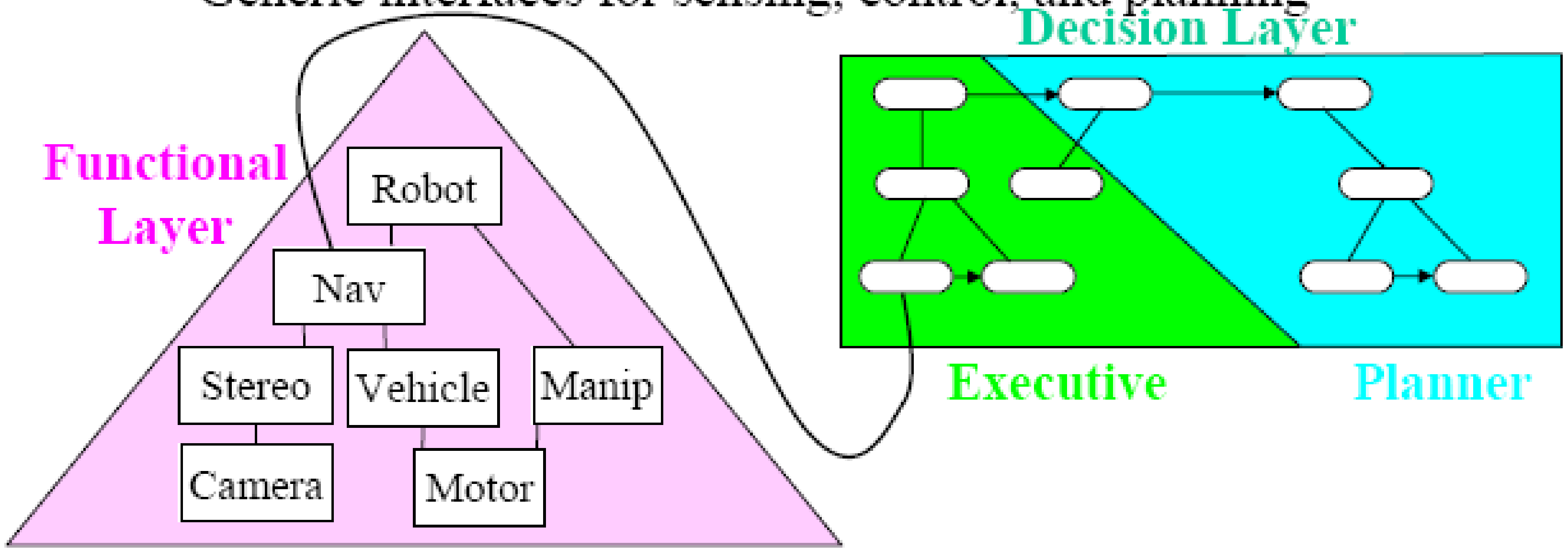


Fig. 7. Concurrent planning and execution.

# Claraty

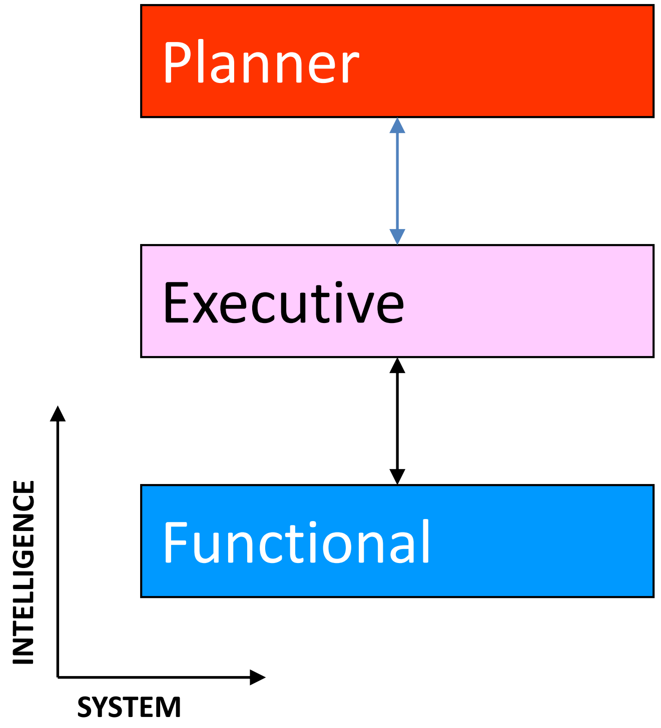
- Two-Tiered Architecture
  - Functional layer: Object oriented, reusable
  - Decision layer: Tightly integrates planner (ASPEN) and executive (TDL)
- Developed at NASA for Next-Generation Mars Rovers
  - Generic interfaces for sensing, control, and planning



# A New View of Architecture Hierarchy

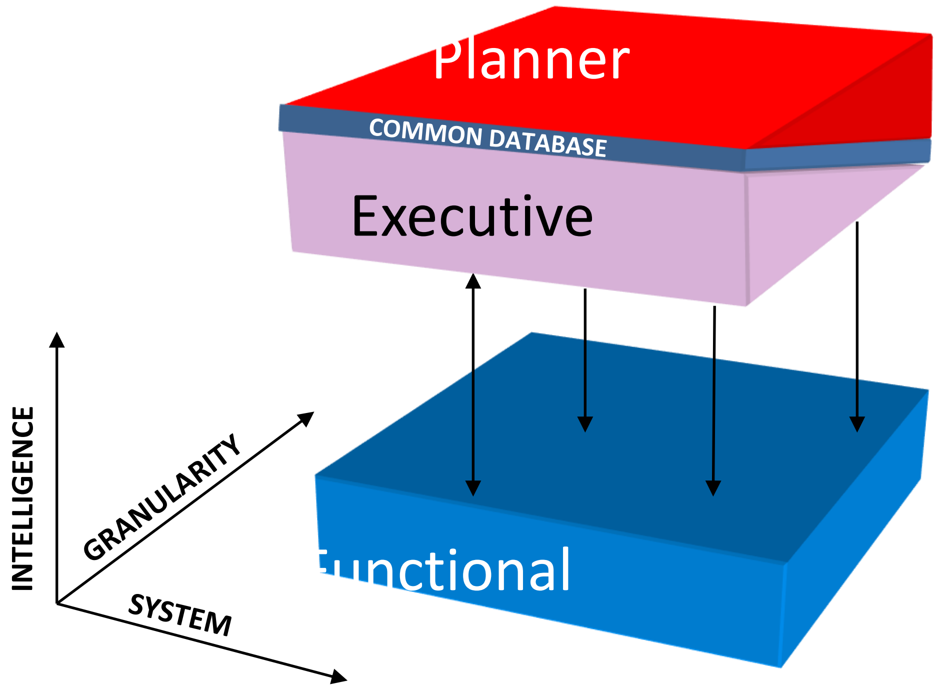
## Typical 3 Level Architecture

- *Functional Level is flat – typically a thin layer over the hardware*
- *Planner has no access to Functional Layer.*
- *Abstraction and granularity is mixed with intelligence.*

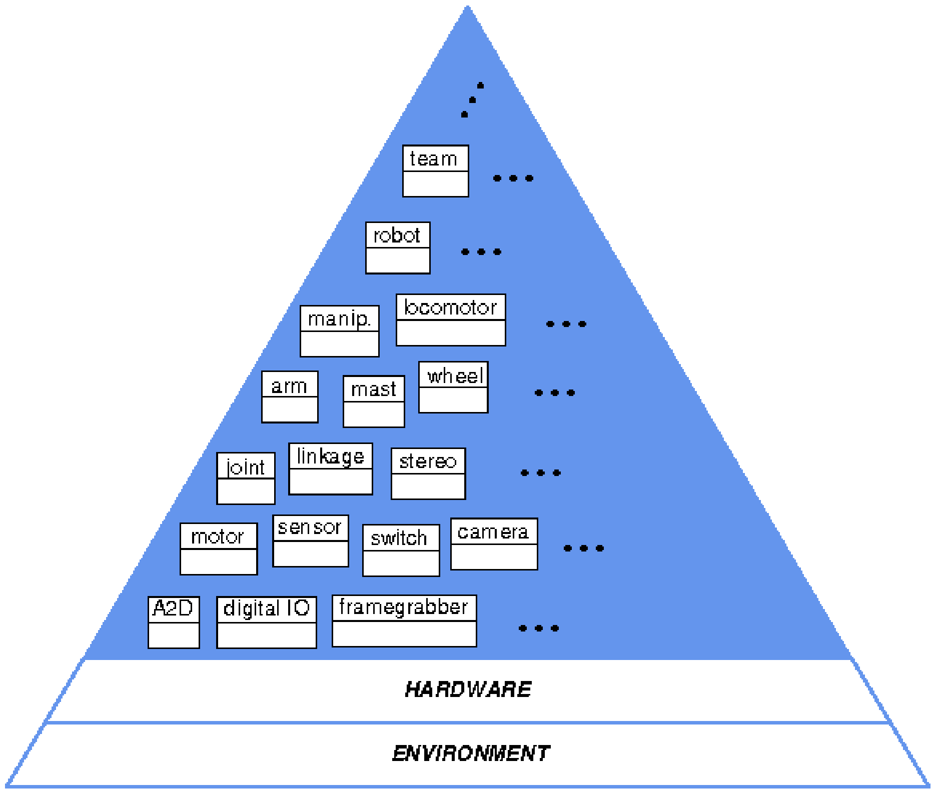


## Proposed 2 Layer Architecture

- *Functional Layer contains object-oriented abstraction of hardware at all levels of system granularity.*
- *Planner and Exec are similar, dominating at different levels of granularity, sharing a common database.*
- *Planner does not have direct access to the Functional Layer for execution, but executive may be minimized.*



## The Functional Layer



- ***Object-oriented Hierarchy:*** captures granularity and abstraction of the system.
- ***Resident State:*** the state of the system components is contained in the appropriate objects and obtained from it by query. This includes state variables values, object state machine status, resource usage, health monitoring, etc.
- ***Local Planners:*** part of the appropriate object. For instance, trajectory planners for arm motion or driving.
- ***Resource Usage Predictors:*** part of the appropriate objects. The prediction can have specified levels of fidelity. For high levels, access of subordinate objects may be needed.
- ***Encoded Functionality:*** objects contain basic functionality for themselves, accessible from within the Functional Layer, as well as the Decision Layer.
- ***Simulation:*** system simulation can be obtained at various levels of fidelity by disconnecting subordinate objects from superior ones.
- ***Test and Debug:*** objects contain test and debug interfaces and have external exercisers.

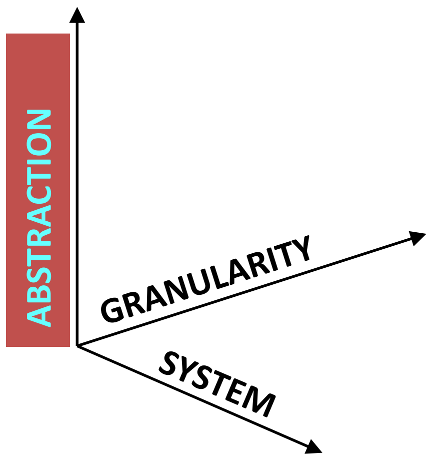
# Functional Layer Object-Oriented Hierarchy

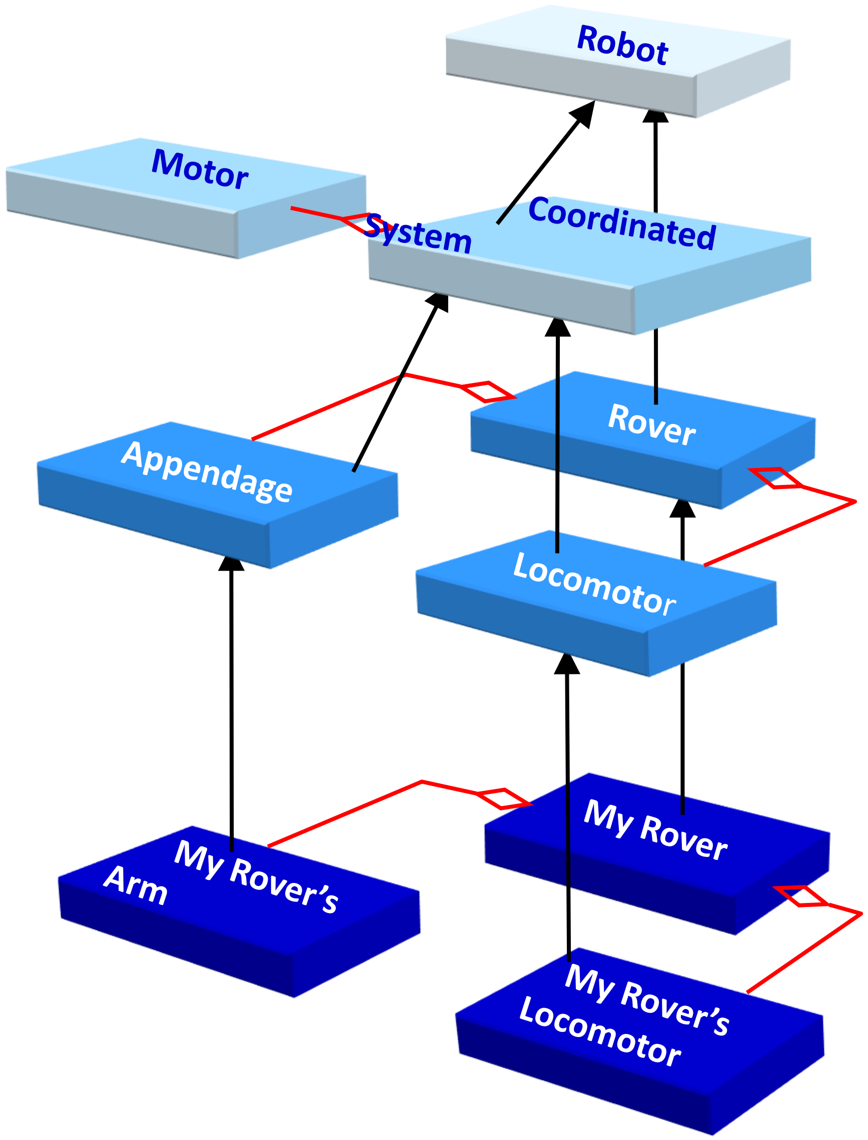
**Implementation Instance:** *Functional Layer* software controlling the system is an instance of software derived from more abstract classes.

**Aggregate Functionality:** Centralizes and enhances capabilities.

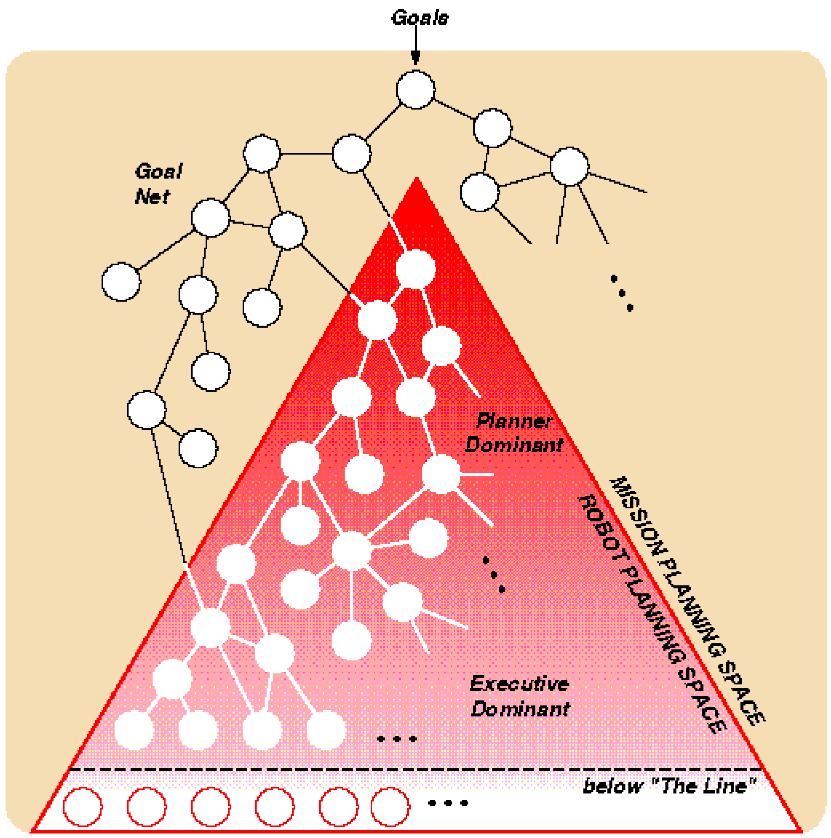
**Abstraction dimension:** In addition to system, granularity, and intelligence.

**Inheritance:** Abstraction dimension is best described by the inheritance properties of an object oriented description





# The Decision Layer



- ***Goal Net:*** temporal constraint network at planning level, task tree at executive level.
- ***Goals:*** specified as constraints on state over time. Consistent with MDS, and ASPEN/CASPER. Runtime loadable. ("What not to do.")
- ***Tasks:*** specified as explicitly parallel or sequential activities that are tightly linked. Consistent with TDL. Compiled in. ("What to do.")
- ***Commands:*** termination fringes of goal net where the Functional Layer is accessed.
- ***The Line:*** the lower border of the elaborated goal net. It is floating, according to the current elaboration. When projected on the Functional Layer, it denotes the border below which the system is a "black box".
- ***State:*** the state of the Functional Layer is obtained by query. The state of the Decision Layer, which is essentially its plan, the active elaboration, and history of execution, is maintained by this layer. It may be save, or reloaded, in whole or part.

## *Decision/Functional Layer Connectivity*

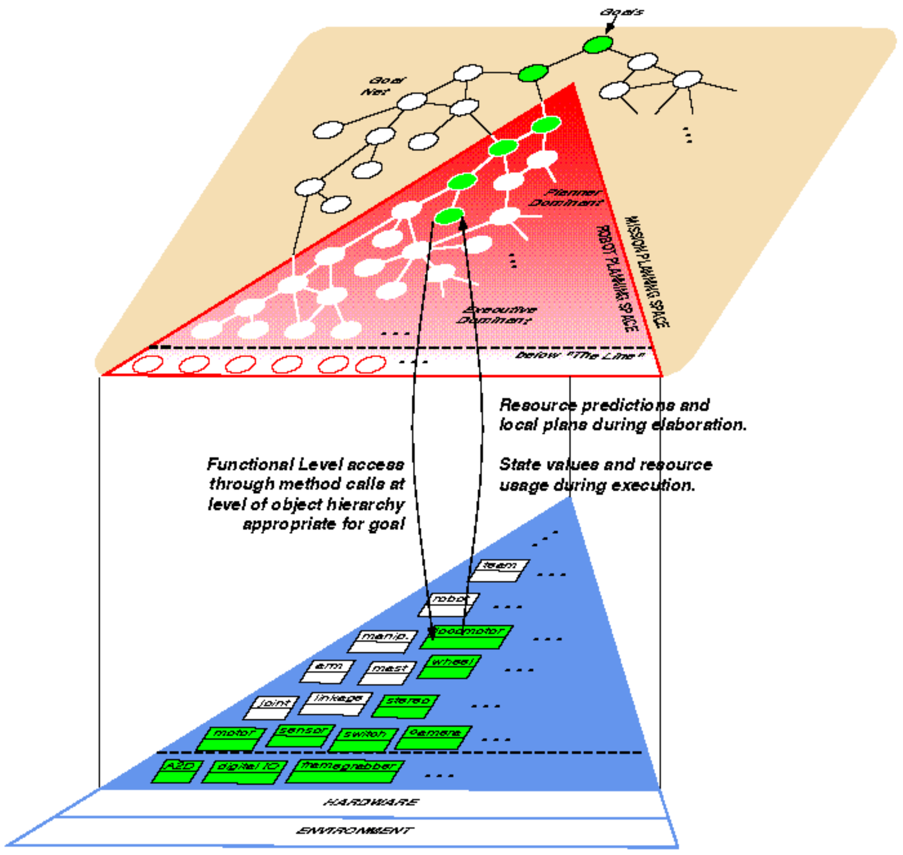
**Lower Granularity**

- Enable global decisions that are lacking in Functional Layer.
- Utilize better decision making capability for smaller scale issues.
- Bypass built-in default capabilities of Functional Layer.
- Take advantage of faster planner or exec.

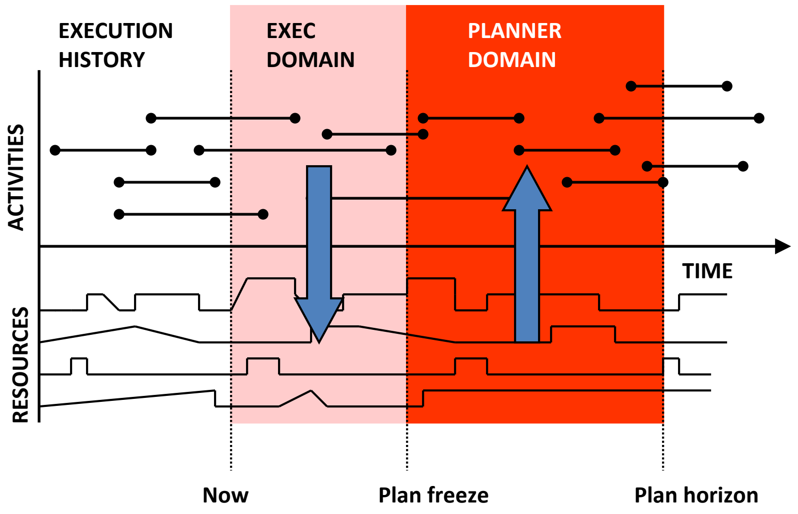
**Higher Granularity**

- Take advantage of built in capabilities of Functional Layer
- Avoid second guessing Functional Layer algorithms and error handling.
- Approach planning and commanding at appropriate level of granularity for the problem – don’t micro-manage the system.
- Allow the use of slower Decision Layer tools.

# Timeline Interaction



- ***Elaboration/Planning:*** expansion of goals/activities, along with resource prediction requests from Functional Layer
- ***Scheduling:*** rearrangement of activities based on resource constraints.
- ***Execution:*** Executive expands activities into task trees or directly into commands. Conditional activities plus state feedback cause changes in resource usage values.
- ***Plan Repair:*** Planner iteratively repairs the plan based on the new projections of resources.



# An Alternative to TCA’s Vertical Capabilities: Horizontal Layered Control

Reason about behavior of objects

Plan changes to the world

Identify objects

Monitor changes

Build maps

Explore

Wander

Avoid objects

# BDI Systems

[Bratman,1987]. Intention, Plans, and Practical Reason.

- BDI model inspired by the Michael Bratman's theory of human practical reasoning:
  - resource-bounded agent
  - intention and desire are proactive, intentions as commitments

Core concepts

Beliefs = information the agent has about the world
Desires = state of affairs that the agent would wish to bring about
Intentions = desires (or actions) that the agent has committed to achieve

**Belief:** the agent knowledge about about the world (belief set)
**Desires:** motivational state, objectives, tasks to be acheived (goals)
**Intentions:** desires with commitment, i.e. plans ready for the execution (plans)

# BDI Systems

BDI particularly compelling because:

- **philosophical component** - based on a theory of rational actions in humans
- **software architecture** - it has been implemented and successfully used in a number of complex fielded applications
  - IRMA - Intelligent Resource-bounded Machine Architecture
  - PRS - Procedural Reasoning System
- **logical component** - the model has been rigorously formalized in a family of BDI logics
  - Rao & Georgeff, Wooldrige
  - $(\text{Int}\ A_i\ \varphi) \rightarrow \neg(\text{Bel}\ A_i\ \varphi)$

## Practical Reasoning Agents: Deliberation: Intentions and Desires

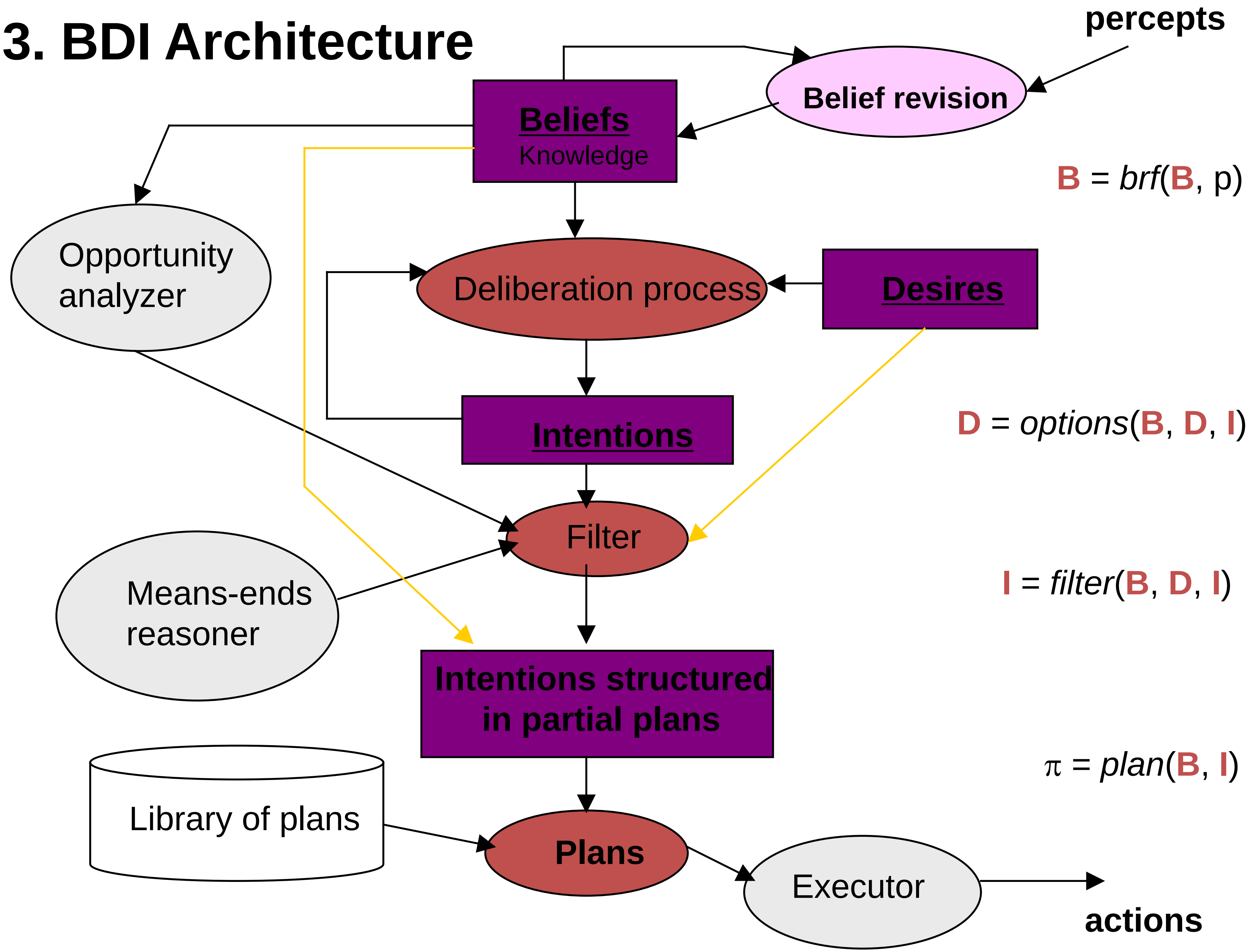
– intentions are stronger than desires

  – “My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires [. . . ] before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions.” *[Bratman, 1990]*

# Practical Reasoning Agents: Intentions

1. agents are expected to **determine ways of achieving** intentions
   - *If I have an intention to Φ, you would expect me to devote resources to deciding how to bring about Φ*
2. agents **cannot** adopt intentions which **conflict**
   - *If I have an intention to Φ , you would not expect me to adopt an intention Ψ that was incompatible with Φ*
3. agents are inclined to **try again** if their attempts to achieve their intention fail
   - *If an agent's first attempt to achieve Φ fails, then all other things being equal, it will try an alternative plan to achieve Φ*
4. agents **believe** their intentions are **possible**
   - *That is, they believe there is at least some way that the intentions could be brought about.*
5. agents do **not believe** they will **not bring about** their intentions
   - *It would not be rational of me to adopt an intention to Φ if I believed that I would fail with Φ*
6. under certain circumstances, agents **believe** they **will bring about** their intentions
   - *If I intend Φ, then I believe that under "normal circumstances" I will succeed withΦ*
7. agents need **not intend** all the expected **side effects** of their intentions
   - *I may believe that going to the dentist involves pain, and I may also intend to go to the dentist — but this does not imply that I intend to suffer pain!*

## 3. BDI Architecture



**B** = *brf*(**B**, p)

**D** = *options*(**B**, **D**, **I**)

**I** = *filter*(**B**, **D**, **I**)

$\pi$ = *plan*(**B**, **I**)

# Practical Reasoning Agents

- agent control loop

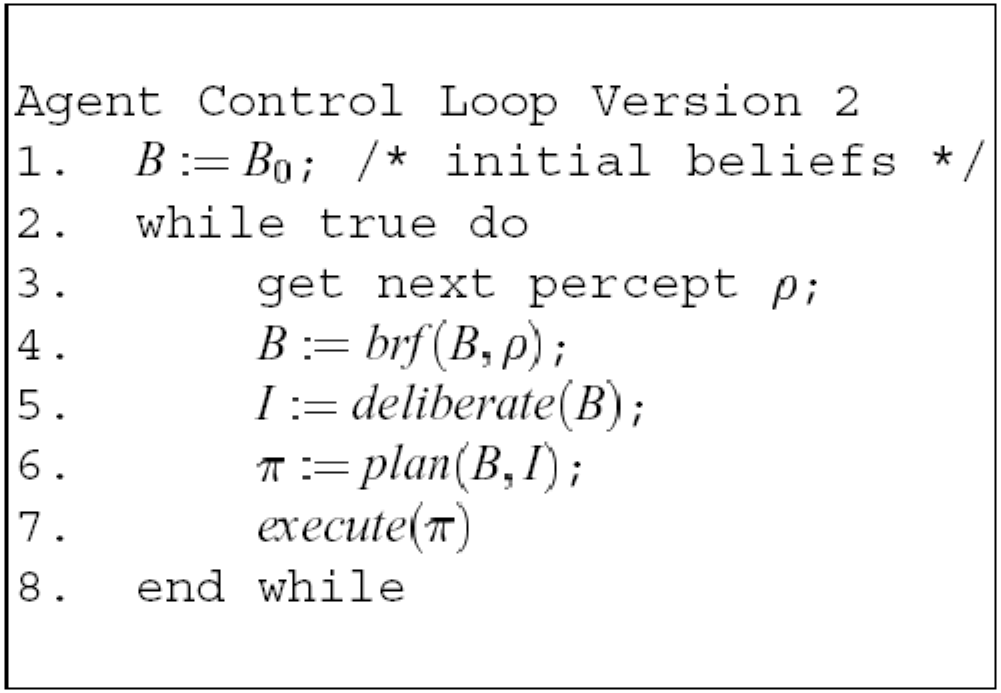
```
while true
    observe the world;
    update internal world model;
    deliberate about what intention to achieve next;
    use means-ends reasoning to get a plan for the
  intention;
    execute the plan
end while
```

- what are the **options** (**desires**) ?
- how to **choose** an option ?
- incl. **filter**
- chosen option ➔ **intention** …

- when to reconsider intentions !?

## Implementing Practical Reasoning Agents

- Let’s make the algorithm more formal:



```
Agent Control Loop Version 2
1.  $B := B_0$; /* initial beliefs */
2.  while true do
3.      get next percept $\rho$;
4.      $B := brf(B, \rho)$;
5.      $I := deliberate(B)$;
6.      $\pi := plan(B, I)$;
7.      $execute(\pi)$
8.  end while
```

## Procedural Reasoning System (PRS)

- Framework for symbolic reactive control systems in dynamic environments
  - Eg Mobile robot control
  - Eg diagnosis of the Space Shuttle’s Reaction Controls System

## Implementing Practical Reasoning Agents

- optimal behaviour if
  - deliberation and means-ends reasoning take a small amount of time;
  - the world is guaranteed to remain static while the agent is deliberating and performing means-ends reasoning;
  - an intention that is optimal when achieved at time t0 (the time at which the world is observed) is guaranteed to remain optimal until time t2 (the time at which the agent has found a course of action to achieve the intention).

## Deliberation

- The *deliberate* function can be decomposed into two distinct functional components:
  - *option generation*
    the agent generates a set of possible alternatives. A function, *options*, takes the agent's current beliefs and current intentions, and from them determines a set of options (= *desires*)
  - *filtering*
    the agent chooses between competing alternatives, and commits to achieving them. In order to select between competing options, an agent uses a *filter* function.

## Deliberation

```
Agent Control Loop Version 3
1.
2.    $B := B_0$;
3.    $I := I_0$;
4.    while true do
5.        get next percept $\rho$;
6.        $B := brf(B, \rho)$;
7.        $D := options(B, I)$;
8.        $I := filter(B, D, I)$;
9.        $\pi := plan(B, I)$;
10.       $execute(\pi)$
11.   end while
```

# Practical Reasoning Agents

If an option has successfully passed trough the filter function and is chosen by the agent as an intention, we say that **the agent has made a commitment to that option.**

Commitment implies temporal persistence of intentions; once an intention is adopted, it should not be immediately dropped out.

How committed an agent should be to its intentions?

- degrees of commitments
  - blind commitment
    - » ≈ fanatical commitment: continue until achieved
  - single-minded commitment
    - » continue until achieved or no longer possible
  - open-minded commitment
    - » continue until no longer believed possible

## Commitment Strategies

- An agent has commitment both
  - to *ends* (i.e., the wishes to bring about)
  - and *means* (i.e., the mechanism via which the agent wishes to achieve the state of affairs)

- current version of agent control loop is overcommitted, both to means and ends

  ➔ modification: *replan* if ever a <u>plan</u> goes wrong

## Agent Control Loop Version 4

```
1.
2.   $B := B_0$;
3.   $I := I_0$;
4.   while true do
5.       get next percept $\rho$;
6.       $B := brf(B, \rho)$;
7.       $D := options(B, I)$;
8.       $I := filter(B, D, I)$;
9.       $\pi := plan(B, I)$;
10.      while not $empty(\pi)$ do
11.          $\alpha := hd(\pi)$;
12.          $execute(\alpha)$;
13.          $\pi := tail(\pi)$;
14.          get next percept $\rho$;
15.          $B := brf(B, \rho)$;
16.          if not $sound(\pi, I, B)$ then
17.              $\pi := plan(B, I)$
18.          end-if
19.      end-while
20.  end-while
```

$\longleftarrow$ *Reactivity, replan*

“Blind commitment”

## Commitment Strategies

- this version still overcommitted to intentions:
  - never stops to consider whether or not its intentions are appropriate

➔ modification: stop for determining whether
intentions have succeeded or whether they are impossible:
"*Single-minded commitment*"

# Single-minded Commitment

```
Agent Control Loop Version 5
2.   $B := B_0$;
3.   $I := I_0$;
4.   while true do
5.       get next percept $\rho$;
6.       $B := brf(B, \rho)$;
7.       $D := options(B, I)$;
8.       $I := filter(B, D, I)$;
9.       $\pi := plan(B, I)$;
10.      while not $empty(\pi)$
                 or $succeeded(I, B)$
                 or $impossible(I, B)$) do
11.          $\alpha := hd(\pi)$;
12.          $execute(\alpha)$;
13.          $\pi := tail(\pi)$;
14.          get next percept $\rho$;
15.          $B := brf(B, \rho)$;
16.          if not $sound(\pi, I, B)$ then
17.              $\pi := plan(B, I)$
18.          end-if
19.      end-while
20.  end-while
```

*Dropping intentions that are impossible or have succeeded*

*Reactivity, replan*

# Intention Reconsideration

- Our agent gets to reconsider its intentions when:
  - it has <u>completely executed a plan</u> to achieve its current intentions; or
  - it believes it has <u>achieved its current intentions</u>; or
  - it believes its <u>current intentions are no longer possible</u>.

→ This is limited in the way that it permits an agent to *reconsider* its intentions

➔ modification:

Reconsider intentions after executing every action

"*Open-minded commitment*"

## Open-minded Commitment

```
Agent Control Loop Version 6

$B := B_0$;
$I := I_0$;
while true do
    get next percept $\rho$;
    $B := brf(B, \rho)$;
    $D := options(B, I)$;
    $I := filter(B, D, I)$;
    $\pi := plan(B, I)$;
    while not ($empty(\pi)$
            or $succeeded(I, B)$
            or $impossible(I, B)$) do
        $\alpha := hd(\pi)$;
        $execute(\alpha)$;
        $\pi := tail(\pi)$;
        get next percept $\rho$;
        $B := brf(B, \rho)$;
        $D := options(B, I)$;
        $I := filter(B, D, I)$;
        if not $sound(\pi, I, B)$ then
            $\pi := plan(B, I)$
        end-if
    end-while
end-while
```

# Intention Reconsideration

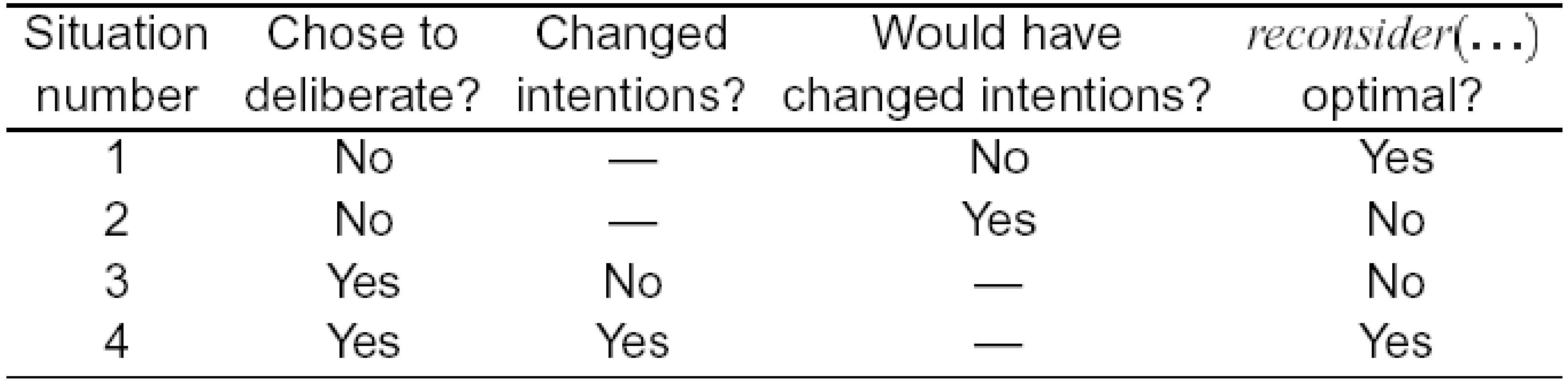
- But intention reconsideration is *costly*!
  A dilemma:
  - an agent that does not stop to reconsider its intentions sufficiently often will continue attempting to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them
  - an agent that *constantly* reconsiders its attentions may spend insufficient time actually working to achieve them, and hence runs the risk of never actually achieving them
- Solution: incorporate an explicit ***<u>meta-level control</u>*** component, that decides whether or not to reconsider

```
Agent Control Loop Version 7
1.
2.   $B := B_0$;
3.   $I := I_0$;
4.   while true do
5.       get next percept $\rho$;
6.       $B := brf(B, \rho)$;
7.       $D := options(B, I)$;
8.       $I := filter(B, D, I)$;
9.       $\pi := plan(B, I)$;
10.      while not ($empty(\pi)$
                   or $succeeded(I, B)$
                   or $impossible(I, B)$) do
11.          $\alpha := hd(\pi)$;
12.          $execute(\alpha)$;
13.          $\pi := tail(\pi)$;
14.          get next percept $\rho$;
15.          $B := brf(B, \rho)$;
16.          if $reconsider(I, B)$ then
17.              $D := options(B, I)$;
18.              $I := filter(B, D, I)$;
19.          end-if
20.          if not $sound(\pi, I, B)$ then
21.              $\pi := plan(B, I)$
22.          end-if
23.      end-while
24.  end-while
```

*meta-level control*

## Possible Interactions

- The possible interactions between meta-level control and deliberation are:



| Situation number | Chose to deliberate? | Changed intentions? | Would have changed intentions? | *reconsider*(...) optimal? |
|---|---|---|---|---|
| 1 | No | — | No | Yes |
| 2 | No | — | Yes | No |
| 3 | Yes | No | — | No |
| 4 | Yes | Yes | — | Yes |

# Intention Reconsideration

- Situations
  - In situation (1), the agent did not choose to deliberate, and as consequence, did not choose to change intentions.
    Moreover, if it *had* chosen to deliberate, it would not have changed intentions.
    the *reconsider(…)* function is behaving optimally.
  - In situation (2), the agent did not choose to deliberate, but if it had done so, it *would* have changed intentions.
    the *reconsider(…)* function is not behaving optimally.
  - In situation (3), the agent chose to deliberate, but did not change intentions.
    the *reconsider(…)* function is not behaving optimally.
  - In situation (4), the agent chose to deliberate, and did change intentions.
    the *reconsider(…)* function is behaving optimally.
- An important assumption: cost of *reconsider(…)* is *much* less than the cost of the deliberation process itself.

# Optimal Intention Reconsideration

- Kinny and Georgeff's experimentally investigated effectiveness of intention reconsideration strategies
- Two different types of reconsideration strategy were used:
  - *bold* agents
    never pause to reconsider intentions, and
  - *cautious* agents
    stop to reconsider after every action
- *Dynamism* in the environment is represented by the *rate of world change*, $\gamma$

# Optimal Intention Reconsideration

- Results (not surprising):
  - If $\gamma$ is low (i.e., the environment does not change quickly),
    bold agents do well compared to cautious ones.
    - cautious ones waste time reconsidering their commitments while bold agents are busy working towards — and achieving — their intentions.
  - If $\gamma$ is high (i.e., the environment changes frequently), cautious agents tend to outperform bold agents.
    - they are able to recognize when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities when they arise.

# Implemented BDI Agents: IRMA

- IRMA – Intelligent Resource-bounded Machine Architecture – Bratman, Israel, Pollack

- IRMA has four key symbolic data structures:
  - a *plan library*
  - explicit representations of
    - *beliefs*: information available to the agent — may be represented symbolically, but may be simple variables
    - *desires*: those things the agent would *like* to make true — think of desires as *tasks* that the agent has been allocated;
    - *intentions*: desires that the agent has *chosen* and *committed to*

## IRMA

- Additionally, the architecture has:
  - a *reasoner*
    - for reasoning about the world; an inference engine
  - a *means-ends analyzer*
    - determines which plans might be used to achieve intentions
  - an *opportunity analyzer*
    - monitors the environment, and as a result of changes, generates new options
  - a *filtering process*
    - determines which options are compatible with current intentions
  - a *deliberation process*
    - responsible for deciding upon the ‘best’ intentions to adopt