Reinforcement Learning

Robotica Probabilistica

Reinforcement Learning

- Task
  - Learn how to behave successfully to achieve a goal while interacting with an external environment
  - Learn through experience from trial and error

- Examples
  - Game playing: The agent knows it has won or lost, but it doesn't know the appropriate action in each state
  - Control: a traffic system can measure the delay of cars, but not know how to decrease it.

Reinforcement Learning

- No knowledge of environment
  - Can only act in the world and observe states and reward

- Many factors make RL difficult:

  - Actions have non-deterministic effects
    - Which are initially unknown

  - Rewards / punishments are infrequent
    - Often at the end of long sequences of actions
    - How do we determine what action(s) were really responsible for reward or punishment?
      (credit assignment)

  - World is large and complex

- Nevertheless learner must decide what actions to take
  - We will assume the world behaves as an MDP

Reinforcement Learning

- Something is unknown
- Learning and Planning at the same time
- Ultimate learning and planning paradigm
- Scalability is a big issue, Very Challenging!

  - Zhang, W., Dietterich, T. G., (1995). **A Reinforcement Learning Approach to Job-shop Scheduling**
  - *G. Tesauro* (1994). "TD-Gammon, A Self-Teaching Backgammon Program Achieves Master-level Play" in Neural Computation
  - Reinforcement Learning for Vulnerability Assessment in Peer-to-Peer Networks, IAAI 2008
    - Policy Gradient Update
  - DeepQ Learning AlphaGo (2015/2016)

Reinforcement Learning

passive

Learner

active

Sequential decision problems

- Approaches:
  - Learn values of states (or state histories) & try to maximize utility of their outcomes.
    - Need a model of the environment: what ops & what states they lead to
  - Learn values of state-action pairs
    - Does not require a model of the environment (except legal moves)
    - Cannot look ahead

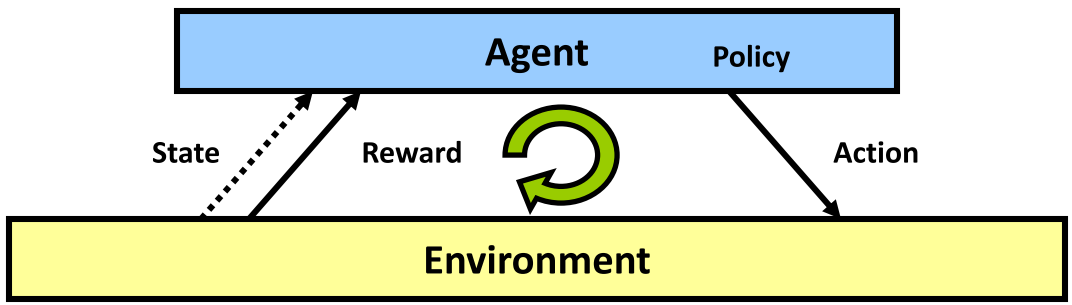
Two Key Aspect in RL

- How we update the value function or policy?
  - How do we form training data
  - Sequence of (s,a,r)….
- How we explore?
  - Exploit or Exploration

Category of Reinforcement Learning

- Model-based RL
  - Constructs domain transition model, MDP
    - E^3 – Kearns and Singh
- Model-free RL
  - Only concerns policy
    - Q-Learning - Watkins
- Active Learning (Off-Policy Learning)
  - Q-Learning
- Passive Learning (On-Policy learning)
  - Sarsa - Sutton

Elements of RL



$$s_0 \xrightarrow{a_0 : r_0} s_1 \xrightarrow{a_1 : r_1} s_2 \xrightarrow{a_2 : r_2} \cdots$$

- **Transition model,** how action influence states
- **Reward R**, immediate value of state-action transition
- **Policy** π**,** maps states to actions

RL task (restated)

- Execute actions in environment,

  observe results.

- Learn action policy π : *state* $\rightarrow$ *action* that

  maximizes <u>expected discounted reward</u>

  $\boldsymbol{E\,[r(t) + \gamma r(t+1) + \gamma^2 r(t+2) + \ldots]}$

  from any starting state in *S*

Reinforcement Learning

- Target function is $\pi : \mathit{state} \rightarrow \mathit{action}$
- However…
  - We have no training examples of form <*state*, *action*>
  - Training examples are of form <<*state*, *action*>, *reward*>

Policy Evaluation

- Given the formula

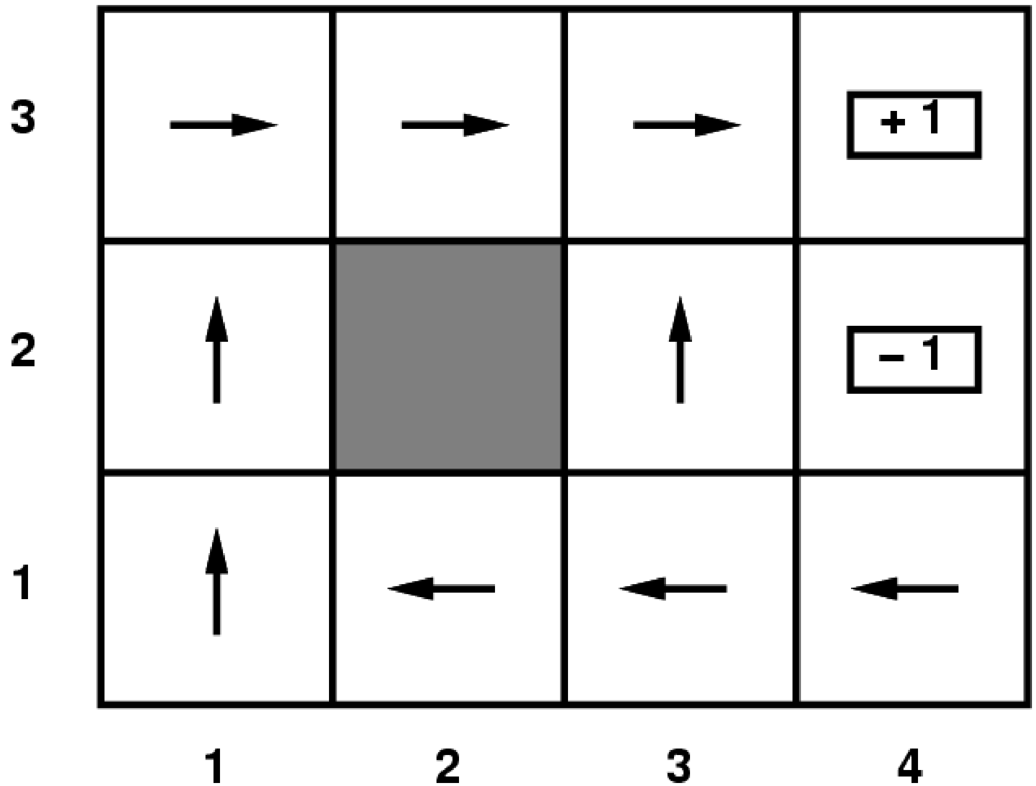
$$V_\pi(s) = R(s) + \beta \sum\nolimits_{s'} T(s, \pi(s), s') \cdot V_\pi(s')$$

- Can we exploit this with RL?
  - What is missing?
  - What needs to be done?

- What do we do after policy evaluation?
  - Policy Update

Example: Passive RL

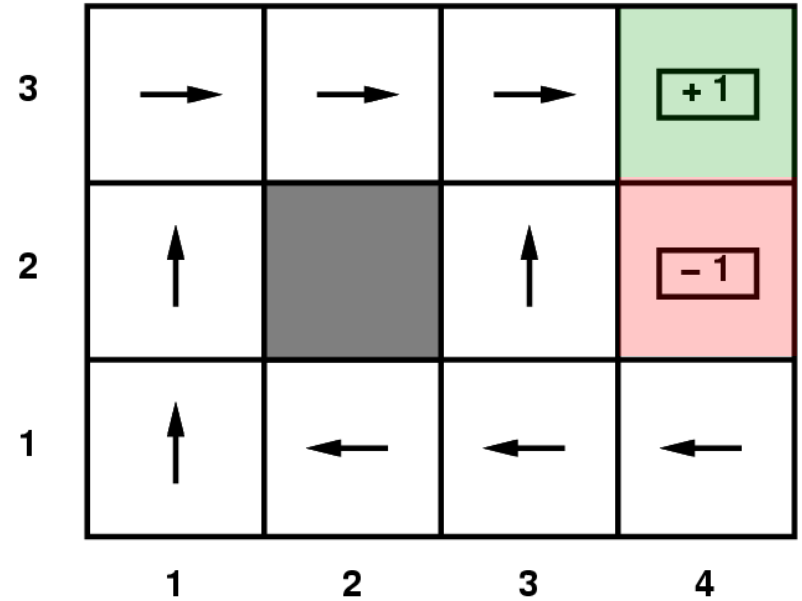
- Suppose given policy
- Want to determine how good it is



Objective: Value Function

3	0.812	0.868	0.918	+1
2	0.762		0.660	– 1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Passive RL



- Given policy π,
  - estimate $V^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Simply follow the policy for many epochs
- Epochs: training sequences

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

Direct Estimation

- Direct estimation (model free)
  - Estimate $V^{\pi}(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)

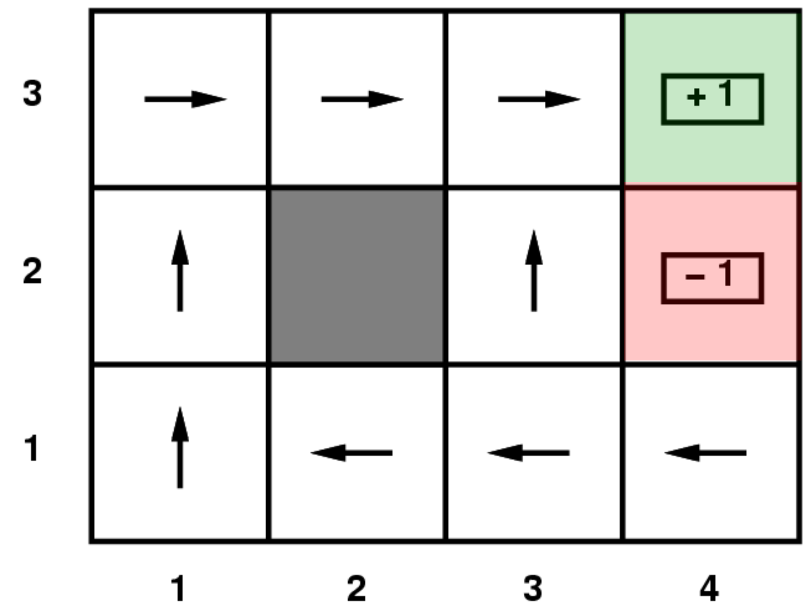
- ***Reward to go*** of a state s

  the sum of the (discounted) rewards from that state until a terminal state is reached

- Key: use observed ***reward to go*** of the state as the direct evidence of the actual expected utility of that state

- Averaging the reward to go samples will converge to true value at state

Passive RL



- Given policy π,
  - estimate $V^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Simply follow the policy for many epochs
- Epochs: training sequences

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) **+1**

0.57 0.64 0.72 0.81 0.9

(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) **+1**

(1,1)→(2,1)→(3,1)→(3,2)→(4,2) **-1**

Direct Estimation

- Converge very slowly to correct utilities values (requires a lot of sequences)

- Does not exploit Bellman on policy values

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

How can we incorporate constraints?

Adaptive Dynamic Programming (ADP)

- ADP is a model based approach
  - Follow the policy for awhile
  - Estimate transition model based on observations
  - Learn reward function
  - Use estimated model to compute utility of policy

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

learned

- How can we estimate transition model T(s,a,s')?
  - Simply the fraction of times we see s' after taking a in state s.

Temporal Difference Learning (TD)

- Can we avoid the computational expense of full DP policy evaluation?
- Temporal Difference Learning
  - Do local updates of utility/value function on a per-action basis
  - Don’t try to estimate entire transition function!
  - For each transition from s to s’, we perform the following update:

$$V^{\pi}(s) = V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}(R(s) + \underbrace{\beta}_{\text{discount factor}} V^{\pi}(s') - V^{\pi}(s))$$

- Intutively, moves us closer to satisfying Bellman constraint

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

Temporal Difference Learning (TD)

- TD update for transition from s to s':

$$V^{\pi}(s) = V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R(s) + \beta V^{\pi}(s')}_{\text{(noisy) sample of utility based on next state}} - V^{\pi}(s))$$

- So the update is maintaining a "mean" of the (noisy) utility samples
- If the learning rate decreases with the number of samples (e.g. 1/n) then the utility estimates will converge to true values

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

Temporal Difference Learning (TD)

- TD update for transition from s to s':

$$V^{\pi}(s) = V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R(s) + \beta V^{\pi}(s')}_{\text{(noisy) sample of utility based on next state}} - V^{\pi}(s))$$

- When V satisfies Bellman constraints then **expected** update is 0.

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

Comparisons

- Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model free)
  - Update speed and implementation similar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to ‘agree’ with observed successor
    - Not ***all*** possible successors
  - Convergence in between direct estimation and ADP

Active Reinforcement Learning

- So far, we have assumed agent with a policy
  – We try to learn how good it is
- Now, suppose agent must learn a good policy (optimal)
  – While acting in uncertain world

Exploration versus Exploitation

- Two reasons to take an action in RL
  - **<u>Exploitation</u>**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **<u>Exploration</u>**: Get more information about the world. How do we know if there is not a pot of gold around the corner.

- To explore we typically need to take actions that do not seem best according to our current model.

- Managing the trade-off between exploration and exploitation is a critical issue in RL

- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s')V(s')$$

  - where V is current value function estimate, and R, T are current estimates of model
  - Q(s,a) is the expected value of taking action a in state s and then getting the estimated value V(s’) of the next state s’

- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE) if it satisfies the following two properties:
  - 1. If a state is visited infinitely often, then each action in that state is chosen infinitely often (with probability 1).
  - 2. In the limit (as t $\rightarrow \infty$), the learning policy is greedy with respect to the learned Q-function (with probability 1).
  - Guarantees convergence

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s')V(s')$$

  - where V is current value function estimate, and R, T are current estimates of model
  - Q(s,a) is the expected value of taking action a in state s and then getting the estimated value V(s’) of the next state s’
- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
  - Guarantees convergence
- Solution 1:
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but it is slow

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s')V(s')$$

  - where V is current value function estimate, and R, T are current estimates of model

- Solution 2: Boltzmann Exploration

  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

TD-based Active RL

1. Start with initial utility/value function
2. Take action according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update

$$V(s) \leftarrow V(s) + \alpha(R(s) + \beta V(s') - V(s))$$

   V(s) is new estimate of optimal value function at state s.

5. Goto 2

Just like TD for passive RL, but we follow explore/exploit policy

Given the usual assumptions about learning rate and GLIE, TD will converge to an optimal value function!

TD-based Active RL

1. Start with initial utility/value function
2. Take action according to an explore/exploit policy
   (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update

$$V(s) \leftarrow V(s) + \alpha(R(s) + \beta V(s') - V(s))$$

   V(s) is new estimate of optimal value function at state s.

5. Goto 2

Requires an estimated model. Why?

To compute Q(s,a) for greedy policy execution

Can we construct a model-free variant?

Q-Learning: Model-Free RL

- Instead of learning the optimal value function V, directly learn the optimal Q function.
  - Recall Q(s,a) is expected value of taking action a in state s and then following the optimal policy thereafter
- The optimal Q-function satisfies $V(s) = \max_{a'} Q(s,a')$ which gives:

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s') V(s')$$

$$= R(s) + \beta \sum_{s'} T(s,a,s') \max_{a'} Q(s,a')$$

- Given the Q function we can act optimally by select action greedily according to Q(s,a)

How can we learn the Q-function directly?

Q-Learning: Model-Free RL

Bellman constraints on optimal Q-function:

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s') \max_{a'} Q(s,a')$$

- We can perform updates after each action just like in TD.
  - After taking action a in state s and reaching state s' do:
    (note that we directly observe reward R(s))

$$Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{R(s) + \beta \max_{a'} Q(s',a')}_{\text{(noisy) sample of Q-value based on next state}} - Q(s,a))$$

Q-Learning

1. Start with initial Q-function (e.g. all zeros)
2. Take action according to an explore/exploit policy
   (should converge to greedy policy, i.e. GLIE)
3. Perform TD update

   $$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s) + \beta \max_{a'} Q(s',a') - Q(s,a)\right)$$

   Q(s,a) is current estimate of optimal Q-function.
4. Goto 2

- Does not require model since we learn Q directly
- Uses explicit |S|x|A| table to represent Q
- Explore/exploit policy directly uses Q-values
  - E.g. use Boltzmann exploration.

Q-Learning Algorithmic Components

- Q-learning:

$$Q_t(s,a) := (1-\alpha_t)Q_{t-1}(s,a) + \alpha_t[r + \gamma \max_{a'} Q_{t-1}(s',a')],$$

  – If infinitely often and $\lim_{T\to\infty}\sum_{t=1}^{T}\alpha_t = \infty \quad \text{and} \quad \lim_{T\to\infty}\sum_{t=1}^{T}\alpha_t^2 < \infty$
  then converngence [Jaakkola,Jordan,Singh 94]

- SARSA(0):

$$Q_t(s,a) := (1-\alpha_t)Q_{t-1}(s,a) + \alpha_t[r + \gamma Q_{t-1}(s',a')],$$

  – Convergence if GLIE policy: infinitely often, in the limit action chosen w.r.t. Q

SARSA

- SARSA(state-action-reward-state-action) equation

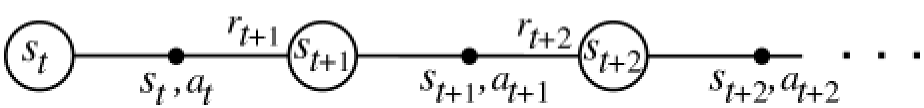
$$Q_t(s,a) := (1-\alpha_t)Q_{t-1}(s,a) + \alpha_t[r + \gamma Q_{t-1}(s',a')],$$

where a' is the action actually taken in state s'.

- The rule is applied at the end of each s,a,r,s',a'.

- **Difference with Q learning**:

Q-learning backs up the *best* Q-value from the state reached while SARSA waits until an action is taken and then backs up the Q-value from that action.

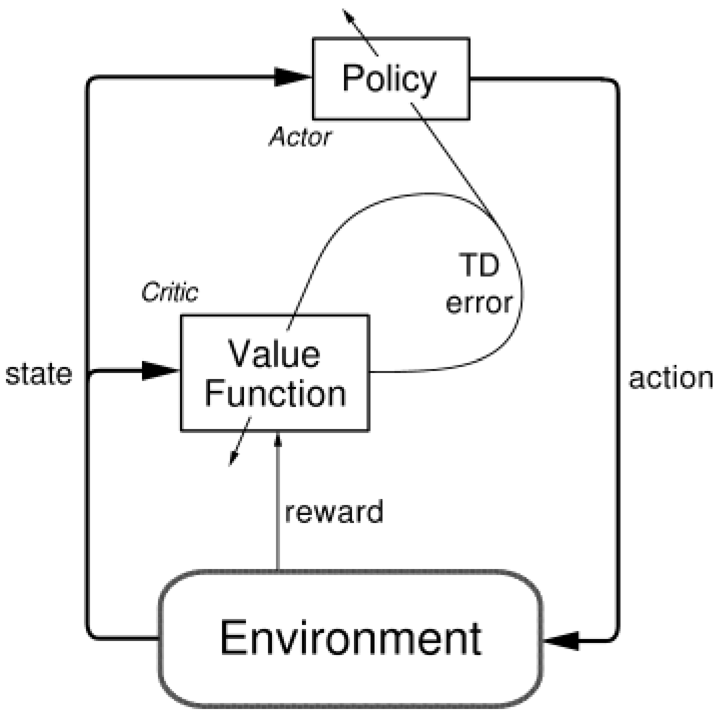


SARSA

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        $Q(s, a) \leftarrow Q(s, a) + \alpha \big[r + \gamma Q(s', a') - Q(s, a) \big]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

Actor Critic Method

- Policy structure (*actor)*: it selects the actions,
- Value function (*critic)*: it criticizes the actions made by the actor.



- Explicit representation of policy as well as value function
- Minimal computation to select actions
- Can learn an explicit stochastic policy
- Can put constraints on policies
- Appealing as psychological and neural models

Actor-Critic Details

TD-error is used to evaluate actions:

$$\delta_t = r_{t+1} + V(s_{t+1}) - V(s_t)$$

If actions are determined by preferences, $p(s,a)$, as follows:

$$\pi_t(s,a) = \Pr\left\{a_t = a \middle| s_t = s\right\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}},$$

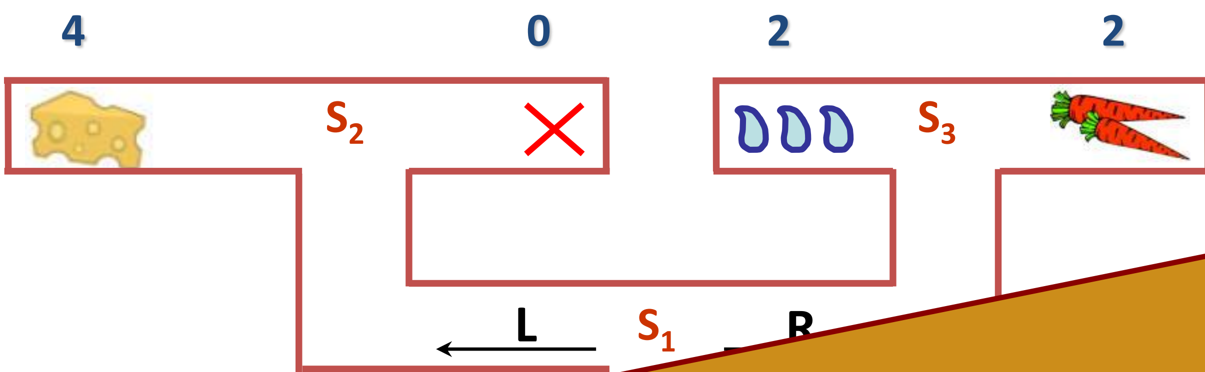
(softmax)

then you can update the preferences like this:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$$

p(s, a) tendency to select
(*preference* for) each action

RL in real world tasks…



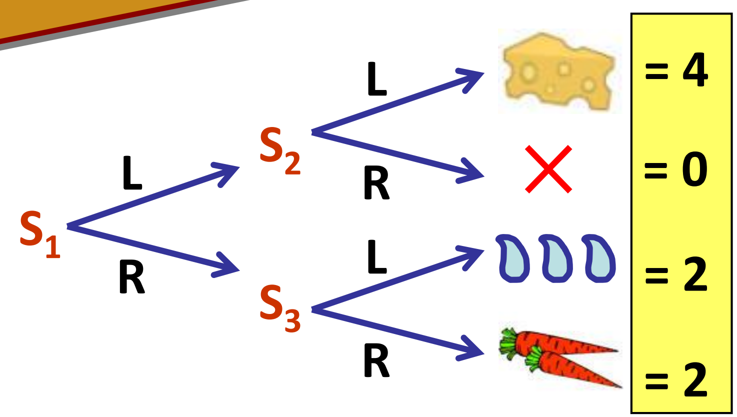
Scaling problem!

$Q(S_3,L) \rightarrow 2$

$Q(S_3,R) \rightarrow 2$

$Q(S_2,L) \rightarrow 4$

$Q(S_2,R) \rightarrow 0$

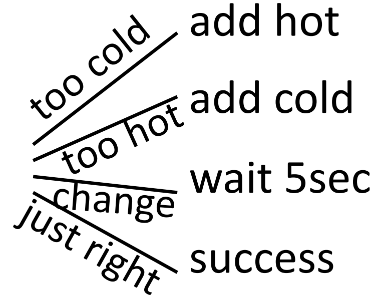


Real-world behavior is hierarchical

1. pour coffee
2. add sugar
3. add milk
4. stir



1. set water temp
2. get wet
3. shampoo
4. soap
5. turn off water
6. dry off

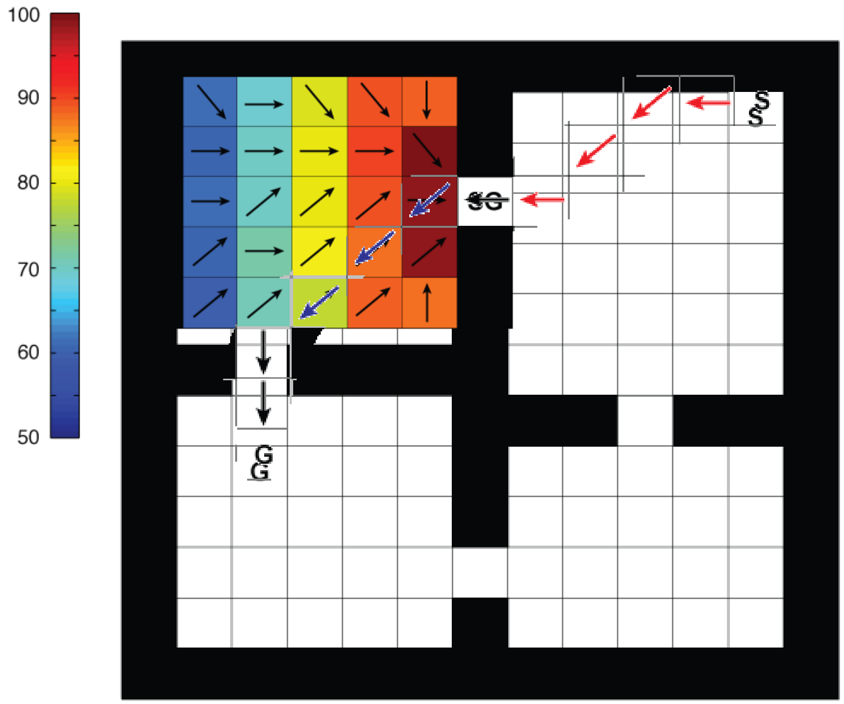


simplified control, disambiguation, encapsulation

Hierarchical Reinforcement Learning

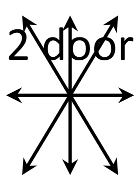
- Exploits domain structure to facilitate learning
  - Policy constraints
  - State abstraction
- Paradigms: Options, HAMs, MaxQ
- MaxQ task hierarchy
  - Directed acyclic graph of subtasks
  - Leaves are the primitive MDP actions
- Traditionally, task structure is provided as prior knowledge to the learning agent

HRL: a toy example



S: start   G: goal

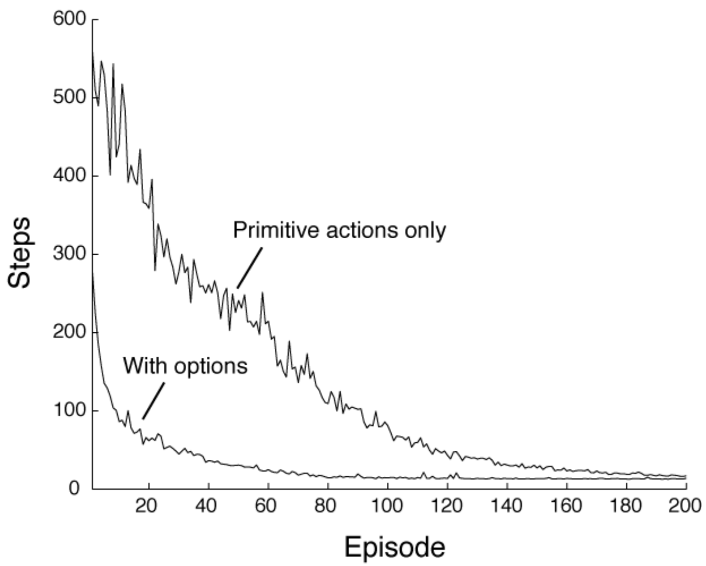
Options: going to doors

Actions:  + 2 door options

Advantages of HRL

1. Faster learning
   (mitigates scaling problem)

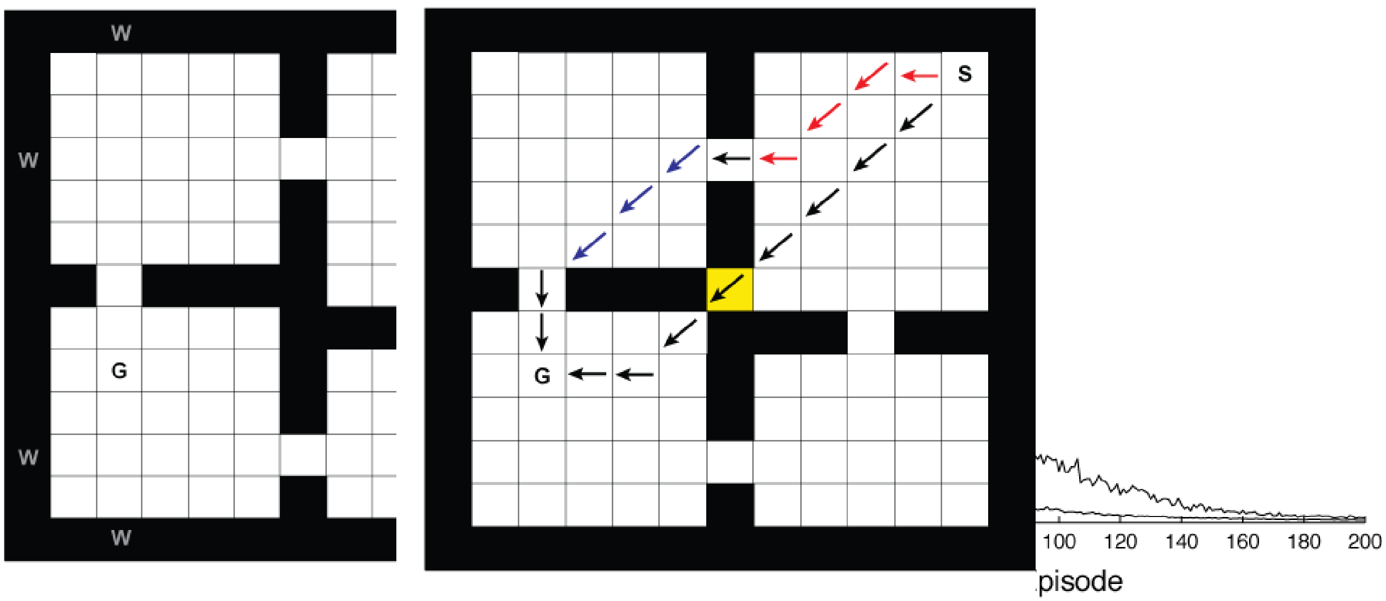
2. *Transfer of knowledge* from previous tasks
   (generalization, shaping)



RL: no longer 'tabula rasa'

Disadvantages (or: the cost) of HRL

1. Need ‘right’ options - how to learn them?
2. Suboptimal behavior (“negative transfer”; habits)
3. More complex learning/control structure



no free lunches…

Semi-Markov Decision Process

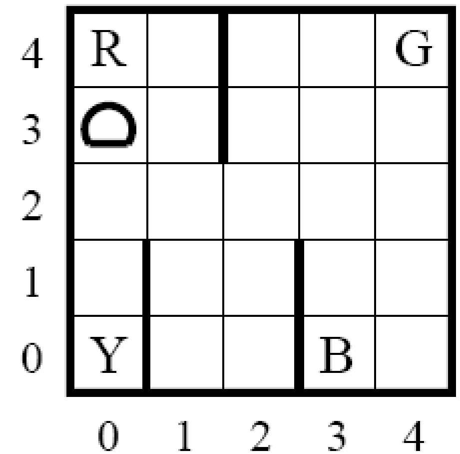
- Generalizes MDPs
- Action **a** takes **N** steps to complete in **s**
- P(**s'**,**n** | **a**, **s**), R(**s'**, **N** | **a**, **s**)
- Bellman equation:

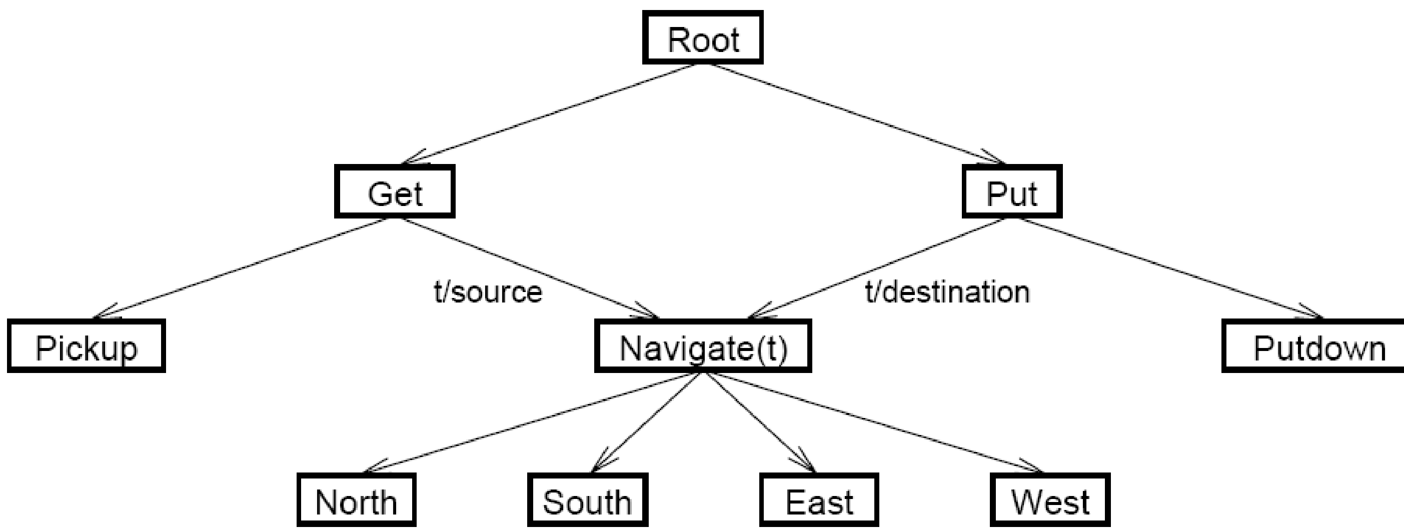
$$V^{\pi}(s) = \sum_{s',N} P(s',N|s,\pi(s)) \left[R(s',N|s,\pi(s)) + \gamma^N V^{\pi}(s') \right].$$

$$V^{\pi}(s) = \overline{R}(s,\pi(s)) + \sum_{s',N} P(s',N|s,\pi(s)) \gamma^N V^{\pi}(s').$$

Taxi Domain

- Motivational Example
- Reward: -1 actions,
  -10 illegal, 20 mission.
- 500 states
- Task Graph:





HSMQ Alg. (Task Decomposition)

function HSMQ(state s, subtask p) **returns** float

Let $TotalReward = 0$

while p is not terminated **do**

Choose action $a = \pi_x(s)$ according to exploration policy π_x

Execute a.

if a is primitive, Observe one-step reward r

else $r := HSMQ(s, a)$, which invokes subroutine a and returns the total reward received while a executed.

$TotalReward := TotalReward + r$

Observe resulting state s'

Update $Q(p, s, a) := (1 - \alpha)Q(p, s, a) + \alpha \left[r + \max_{a'} Q(p, s', a') \right]$

end // **while**

return $TotalReward$

end

MAXQ Alg. (Value Fun. Decomposition)

- Want to obtain some sharing (compactness) in the representation of the value function.
- Re-write Q(p, s, a) as

$$Q(p, s, a) = V(a, s) + C(p, s, a)$$

$$V(p, s) = \max_a \left[V(a, s) + C(p, s, a)\right]$$

where *V(a, s)* is the expected total reward while executing action a, and *C(p, s, a)* is the expected reward of completing parent task *p* after *a* has returned

Hierarchical Structure

- MDP decomposed in task M0, … , Mn

Theorem 1 *Given a task graph over tasks $M_0, \ldots, M_n$ and a hierarchical policy π, each subtask M_i defines a semi-Markov decision process with states S_i, actions A_i, probability transition function $P_i^\pi(s', N|s, a)$, and* expected *reward function $\overline{R}(s, a) = V^\pi(a, s)$, where $V^\pi(a, s)$ is the projected value function for child task M_a in state s. If a is a primitive action, $V^\pi(a, s)$ is defined as the expected immediate reward of executing a in s: $V^\pi(a, s) = \sum_{s'} P(s'|s, a)R(s'|s, a)$.*

- Q for the subtask i

$$Q^\pi(i, s, a) = V^\pi(a, s) + \sum_{s', N} P_i^\pi(s', N|s, a)\gamma^N Q^\pi(i, s', \pi(s')),$$

$$Q^\pi(i, s, a) = V^\pi(a, s) + C^\pi(i, s, a).$$

Value Decomposition

Definition 6 *The* completion function, $C^\pi(i,s,a)$, *is the expected discounted cumulative reward of* completing *subtask* M_i *after invoking the subroutine for subtask* M_a *in state* s. *The reward is discounted back to the point in time where* a *begins execution.*

$$C^\pi(i,s,a) = \sum_{s',N} P_i^\pi(s',N|s,a)\gamma^N Q^\pi(i,s',\pi(s')) \tag{9}$$

With this definition, we can express the Q function recursively as

$$Q^\pi(i,s,a) = V^\pi(a,s) + C^\pi(i,s,a). \tag{10}$$

Finally, we can re-express the definition for $V^\pi(i,s)$ as

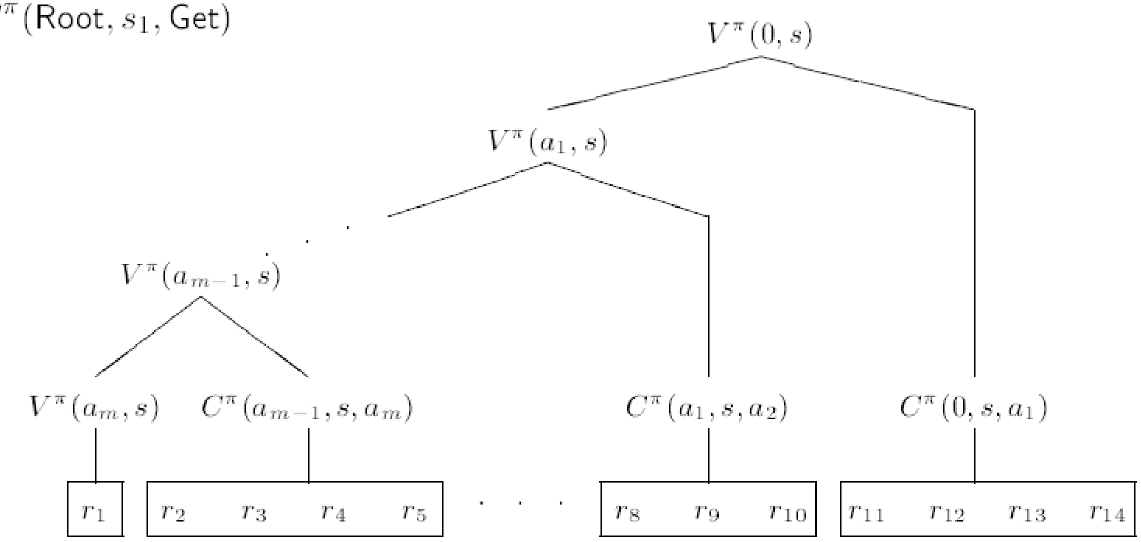
$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)) & \text{if } i \text{ is composite} \\ \sum_{s'} P(s'|s,i)R(s'|s,i) & \text{if } i \text{ is primitive} \end{cases} \tag{11}$$

Value Decomposition

- The value function can be decomposed as follows

$$V^\pi(0,s) = V^\pi(a_m,s) + C^\pi(a_{m-1},s,a_m) + \ldots + C^\pi(a_1,s,a_2) + C^\pi(0,s,a_1)$$

$$\begin{aligned} V^\pi(\mathsf{Root},s_1) &= V^\pi(\mathsf{North},s_1) + C^\pi(\mathsf{Navigate}(R),s_1,\mathsf{North}) + \\ &\quad C^\pi(\mathsf{Get},s_1,\mathsf{Navigate}(R)) + C^\pi(\mathsf{Root},s_1,\mathsf{Get}) \\ &= -1 + 0 + -1 + 12 \\ &= 10 \end{aligned}$$



MAXQ Alg. (cont'd)

- An example

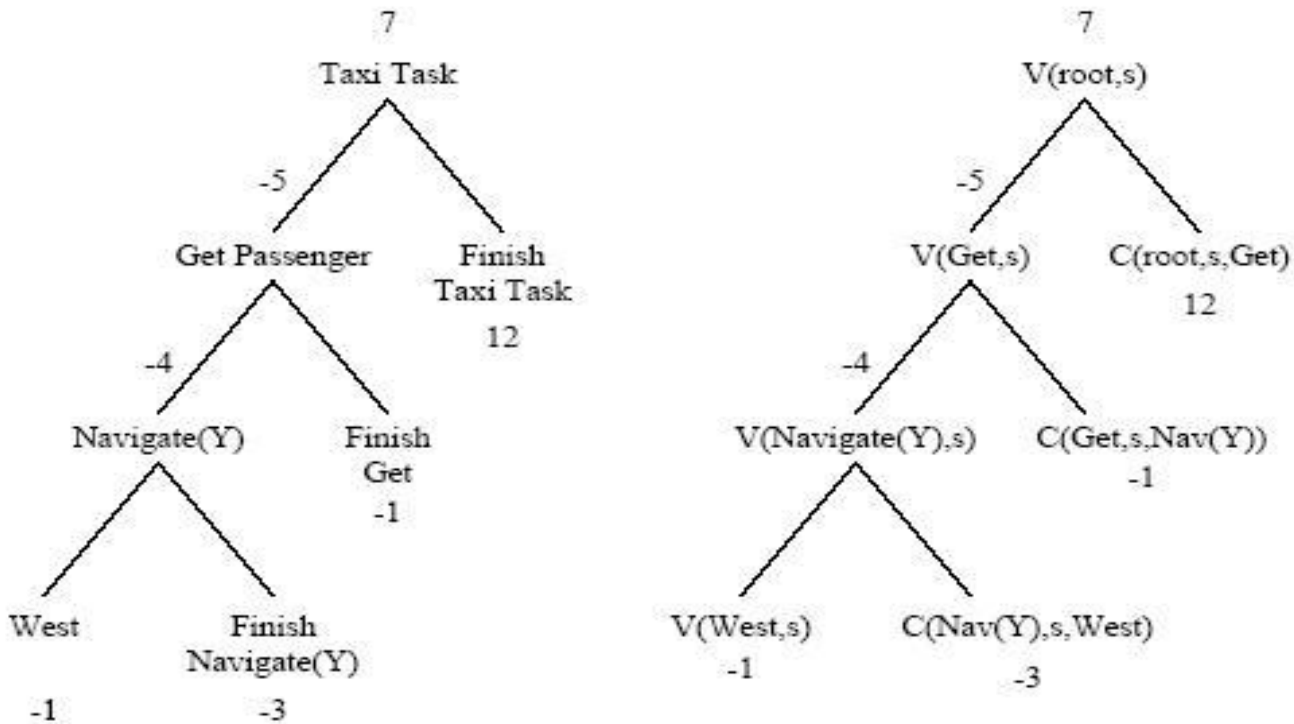


Fig. 5. An example of the MAXQ value function decomposition for the state in which the taxi is at location (2,2), the passenger is at (0,0), and wishes to get to (3,0). The left tree gives English descriptions, and the right tree uses formal notation.

MAXQ Alg. (cont'd)

$$\begin{aligned} V(\text{root}, s) = V(\text{west}, s) &+ C(\text{navigate}(Y), s, \text{west}) \\ &+ C(\text{get}, s, \text{navigate}(Y)) \\ &+ C(\text{root}, s, \text{get}). \end{aligned}$$

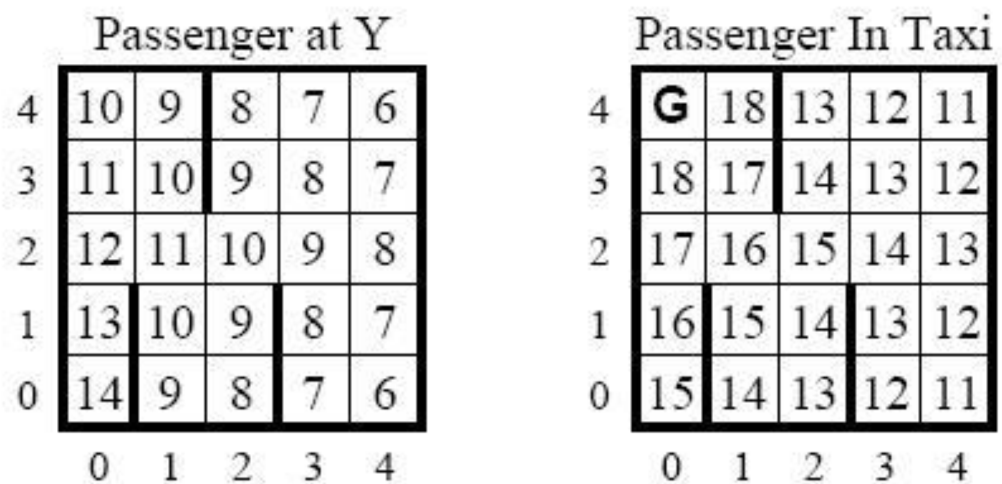


Fig. 4. Value function for the case where the passenger is at (0,0) (location Y) and wishes to get to (0,4) (location R).

MAXQ Alg. (cont'd)

function MAXQQ(state s, subtask p) **returns** float
    Let $TotalReward = 0$
    while p is not terminated **do**
        Choose action $a = \pi_x(s)$ according to exploration policy π_x
        Execute a.
            if a is primitive, Observe one-step reward r
            else $r := MAXQQ(s, a)$, which invokes subroutine a and
                returns the total reward received while a executed.
        $TotalReward := TotalReward + r$
        Observe resulting state s'
        if a is a primitive
            $V(a, s) := (1 - \alpha)V(a, s) + \alpha r$
        else a is a subroutine
            $C(p, a, s) := (1 - \alpha)C(p, s, a) + \alpha \max_{a'} [V(a', s') + C(p, s', a')]$
    end // **while**
    return $TotalReward$
end

State Abstraction

Three fundamental forms

- Irrelevant variables

e.g. passenger location is irrelevant for the **navigate** and **put** subtasks and it thus could be ignored.

- Funnel abstraction

A funnel action is an action that causes a larger number of initial states to be mapped into a small number of resulting states. E.g., the ***navigate(t)*** action maps any state into a state where the taxi is at location *t*. This means the completion cost is independent of the location of the taxi—it is the same for all initial locations of the taxi.

State Abstraction (cont'd)

- Structure constraints

  - E.g. if a task is terminated in a state s, then there is no need to represent its completion cost in that state

  - Also, in some states, the termination predicate of the child task implies the termination predicate of the parent task

Effect

- reduce the amount memory to represent the Q-function.

  14,000 q values required for flat Q-learning

  3,000 for HSMQ (with the irrelevant-variable abstraction

  632 for C() and V() in MAXQ

- learning faster

State Abstraction (cont'd)

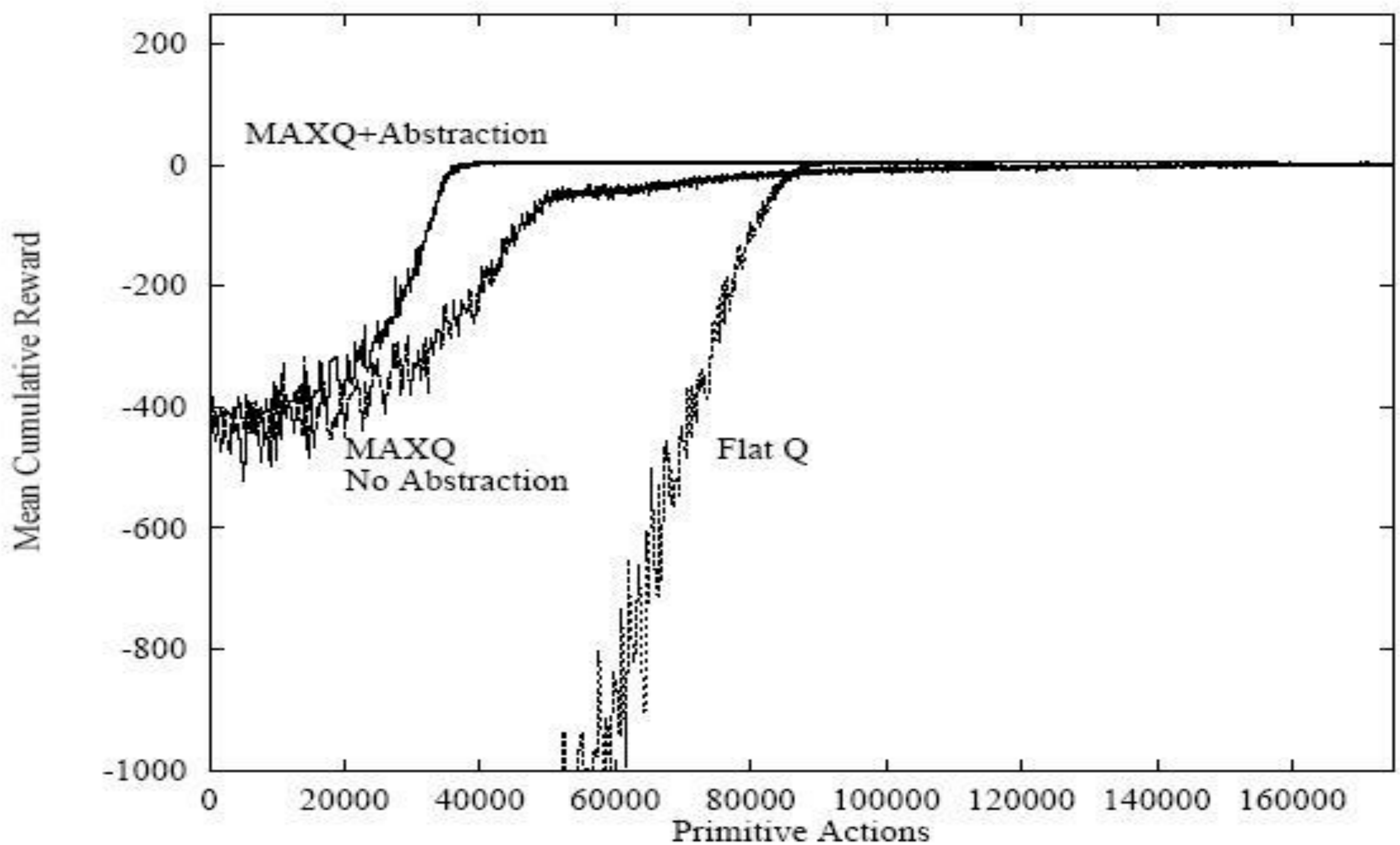
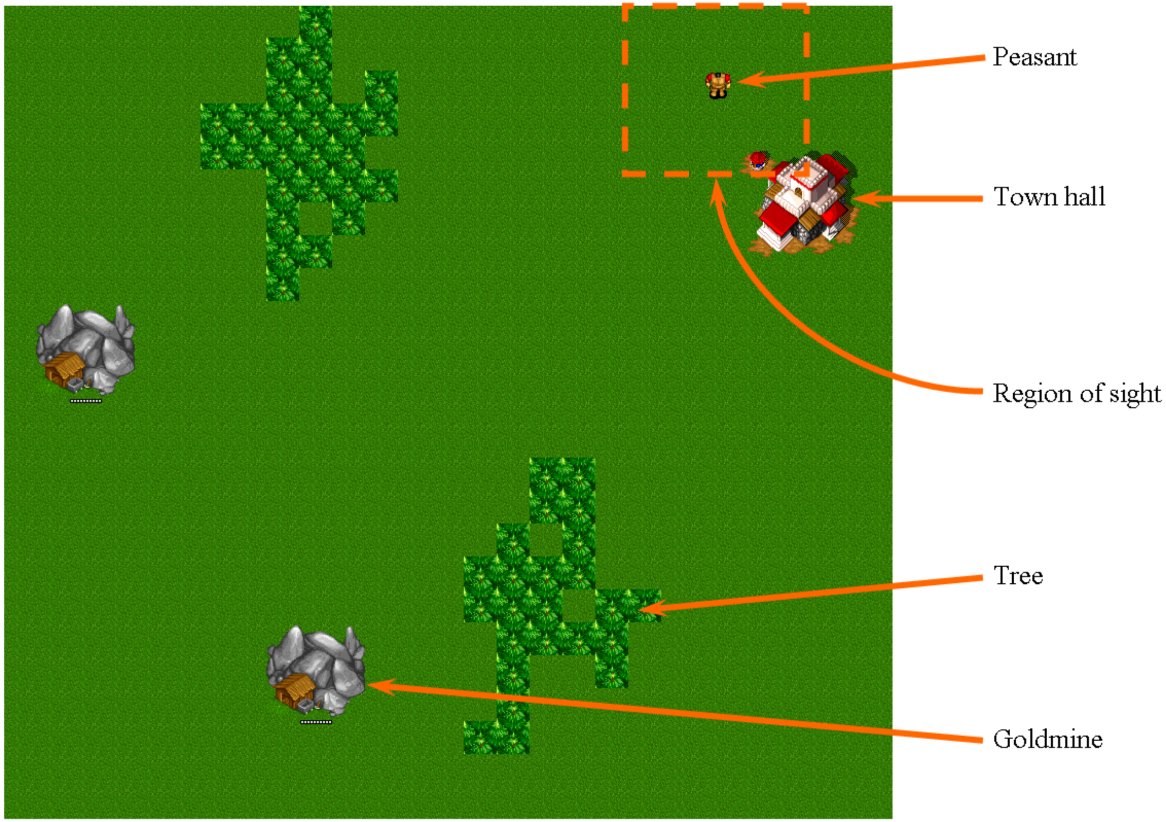


Fig. 7. Comparison of Flat Q learning, MAXQ Q learning with no state abstraction, and MAXQ Q learning with state abstraction on a noisy version of the taxi task.

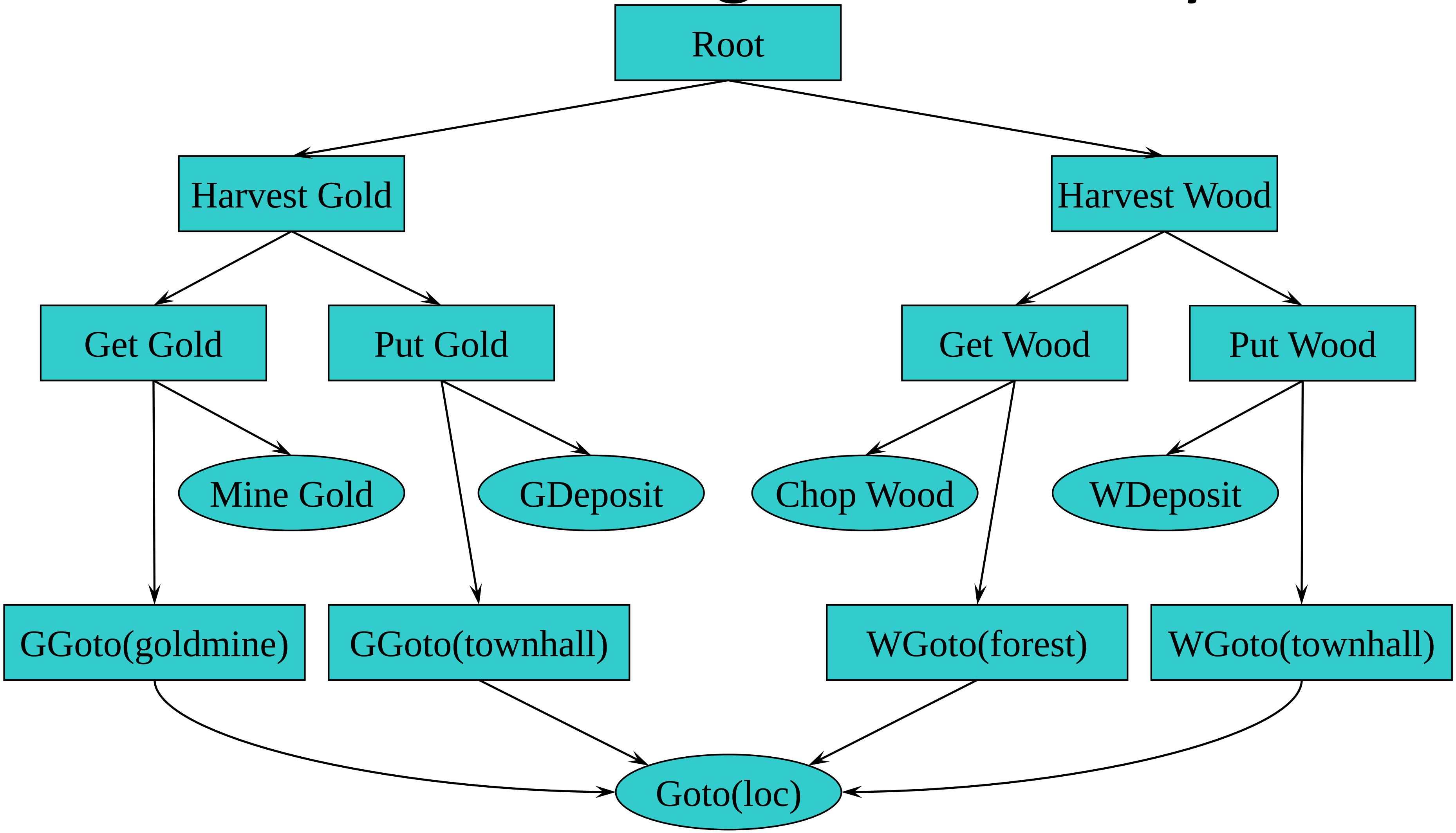
Wargus Resource-Gathering Domain



State variables
Peasant location: `a.l`
Peasant resource: `a.r`
Gold mine within sight radius: `reg.gold`
Trees within sight radius: `reg.wood`
Town hall within sight radius: `reg.townhall`
Required gold quota: `req.gold`
Required wood quota: `req.wood`

Primitive actions
Mine gold: `MG`
Chop wood: `CW`
Deposit: `Dep`
Navigate: `Goto(loc)`

Induced Wargus Hierarchy



Induced Abstraction & Termination

Task Name	State Abstraction	Termination Condition
Root	req.gold, req.wood	req.gold = 1 && req.wood = 1
Harvest Gold	req.gold, agent.resource, region.townhall	req.gold = 1
Get Gold	agent.resource, region.goldmine	agent.resource = gold
Put Gold	req.gold, agent.resource, region.townhall	agent.resource = 0
GGoto(goldmine)	agent.x, agent.y	agent.resource = 0 && region.goldmine = 1
GGoto(townhall)	agent.x, agent.y	req.gold = 0 && agent.resource = gold && region.townhall = 1
Harvest Wood	req.wood, agent.resource, region.townhall	req.wood = 1
Get Wood	agent.resource, region.forest	agent.resource = wood
Put Wood	req.wood, agent.resource, region.townhall	agent.resource = 0
WGoto(forest)	agent.x, agent.y	agent.resource = 0 && region.forest = 1
WGoto(townhall)	agent.x, agent.y	req.wood = 0 && agent.resource = wood && region.townhall = 1
Mine Gold	agent.resource, region.goldmine	NA
Chop Wood	agent.resource, region.forest	NA
GDeposit	req.gold, agent.resource, region.townhall	NA
WDeposit	req.wood, agent.resource, region.townhall	NA
Goto(loc)	agent.x, agent.y	NA

Note that because each subtask has a unique terminal state,
Result Distribution Irrelevance applies

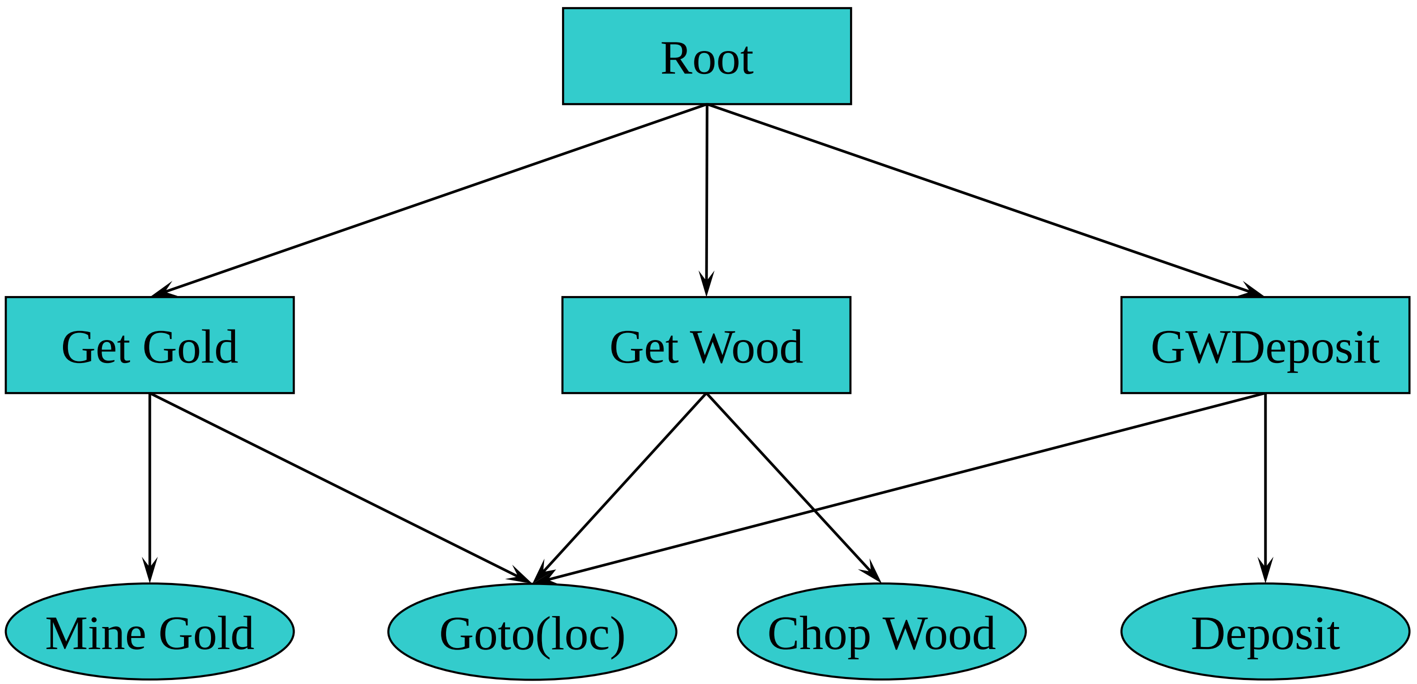
Claims

- The resulting hierarchy is unique
  - Does not depend on the order in which goals and trajectory sequences are analyzed
- All state abstractions are safe
  - There exists a hierarchical policy within the induced hierarchy that will reproduce the observed trajectory
  - Extend MaxQ Node Irrelevance to the induced structure
- Learned hierarchical structure is “locally optimal”
  - No local change in the trajectory segmentation can improve the state abstractions (very weak)

Experimental Setup

- Randomly generate pairs of source-target resource-gathering maps in Wargus
- Learn the optimal policy in source
- Induce task hierarchy from a single (near) optimal trajectory
- Transfer this hierarchical structure to the MaxQ value-function learner for target
- Compare to direct Q learning, and MaxQ learning on a manually engineered hierarchy within target

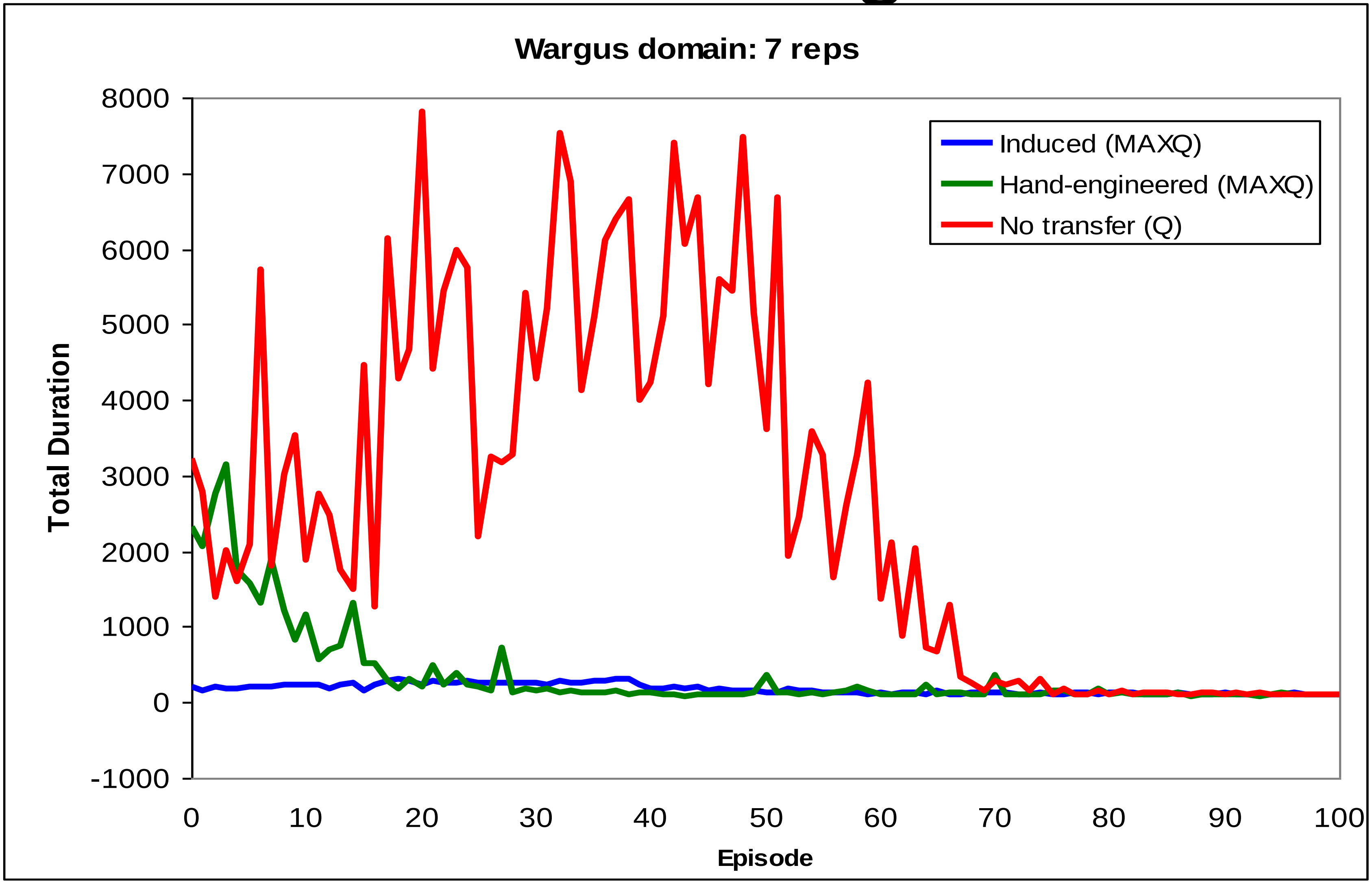
Hand-Built Wargus Hierarchy



Hand-Built Abstractions & Terminations

Task Name	State Abstraction	Termination Condition
Root	req.gold, req.wood, agent.resource	req.gold = 1 && req.wood = 1
Harvest Gold	agent.resource, region.goldmine	agent.resource ≠ 0
Harvest Wood	agent.resource, region.forest	agent.resource ≠ 0
GWDeposit	req.gold, req.wood, agent.resource, region.townhall	agent.resource = 0
Mine Gold	region.goldmine	NA
Chop Wood	region.forest	NA
Deposit	req.gold, req.wood, agent.resource, region.townhall	NA
Goto(loc)	agent.x, agent.y	NA

Results: Wargus



Limitations

- Recursively optimal not necessarily optimal
- Model-free Q-learning

  Model-based algorithms (that is, algorithms that try to learn *P(s'|s,a)* and *R(s'|s,a))* are generally much more efficient because they remember past experience rather than having to re-experience it.

References and Further Reading

- Sutton, R., Barto, A., (2000) *Reinforcement Learning: an Introduction*, The MIT Press

  http://www.cs.ualberta.ca/~sutton/book/the-book.html

- Kaelbling, L., Littman, M., Moore, A., (1996) Reinforcement Learning: a Survey, *Journal of Artificial Intelligence Research*, **4**:237-285

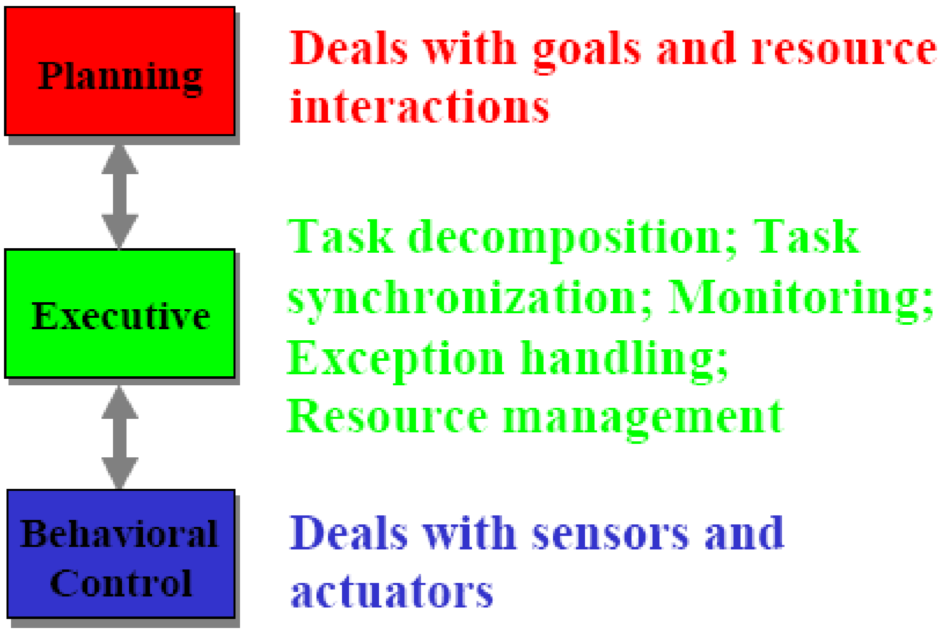
- Barto, A., Mahadevan, S., (2003) Recent Advances in Hierarchical Reinforcement Learning, *Discrete Event Dynamic Systems: Theory and Applications*, **13**(4):41-77

Task Planning

Architetture Robotiche

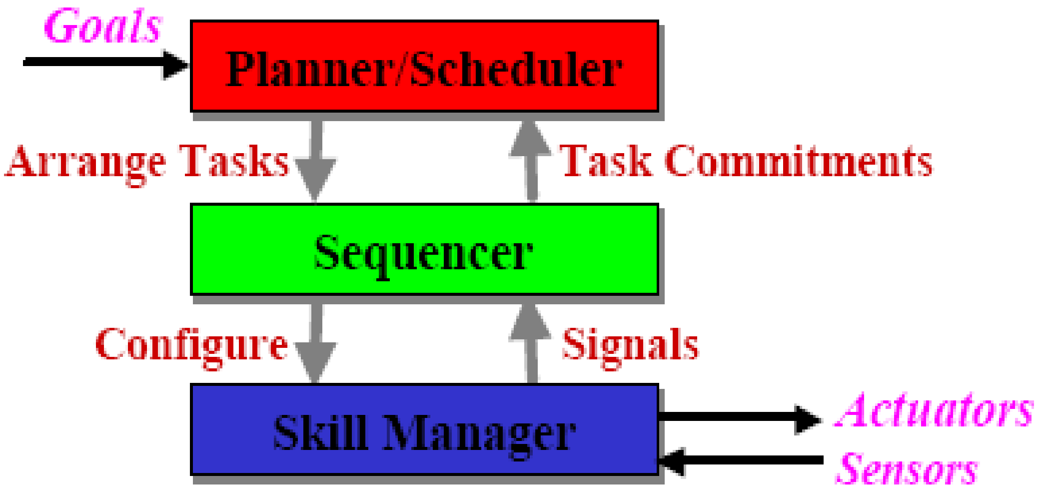
Architetture a 3 Livelli

- Deliberativo:
  pianificazione, ragionamento, decisione
- Esecutivo:
  monitoraggio dell’esecuzione,
  sequenziamento dei comandi
- Funzionale:
  funzionalità di controllo attuative e percettive



Architetture a 3 Livelli: ATLANTIS

- Explicit Separation of Planning, Sequencing, and Control
  - Upper layers provide *control flow* for lower layers
  - Lower layers provide *status* (state change) and *synchronization* (success/failure) for upper layers
- Heterogeneous Architecture
  - Each layer utilizes algorithms tuned for its particular role
  - Each layer has a representation to support its reasoning

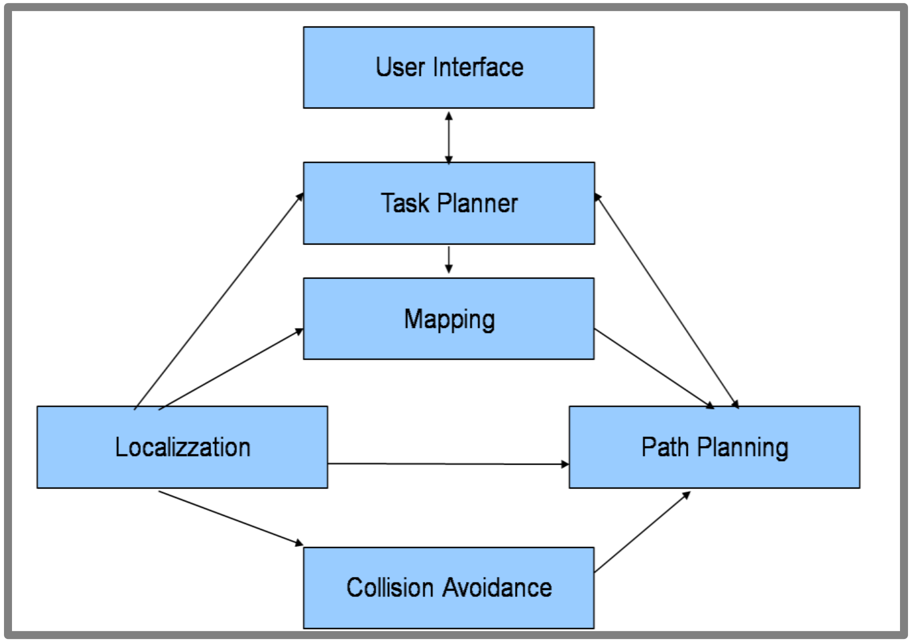


Esempio: RHINO Architettura

Architettura di RHINO la guida robotica del museo di Bonn (1995); simile MINERVA (1998) ad Atlanta

Architettura a 3 Livelli per un robot mobile:

1. Funzionale:
   Mapping, Localizzazione, Avoidance
2. Esecutivo:
   Sequencer, monitor
3. Deliberativo:
   Task Planner



Architetture di RIHINO

Rhino, 1997

Minerva, 1998

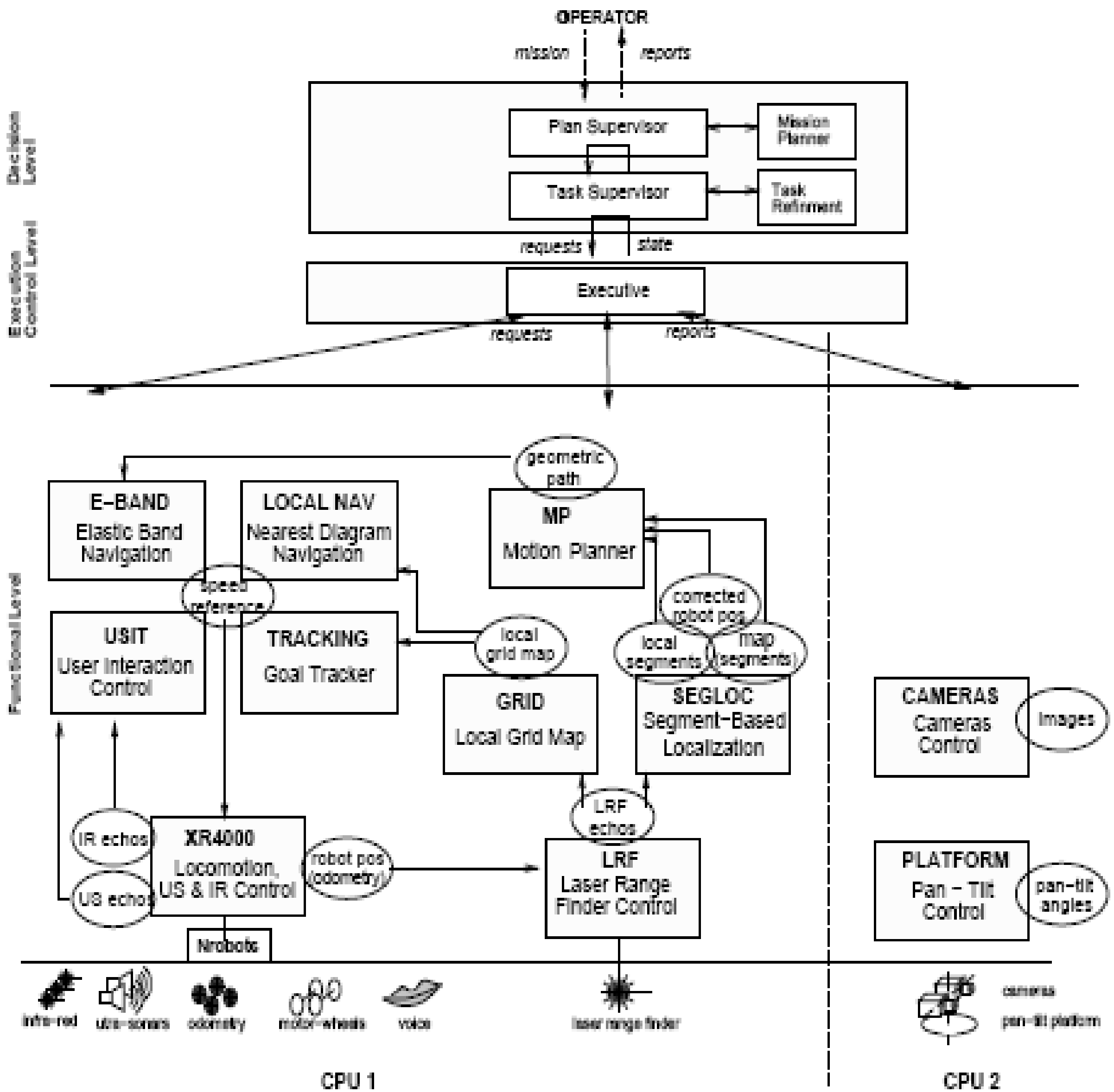
Architetture a 3 Livelli

• LAAS architecture:

Tre Livelli:

1. Deliberativo
   (temporal planner)
2. Esecutivo
   (PRS)
3. Funzionale
   (GENOME)

Controllo di Rover



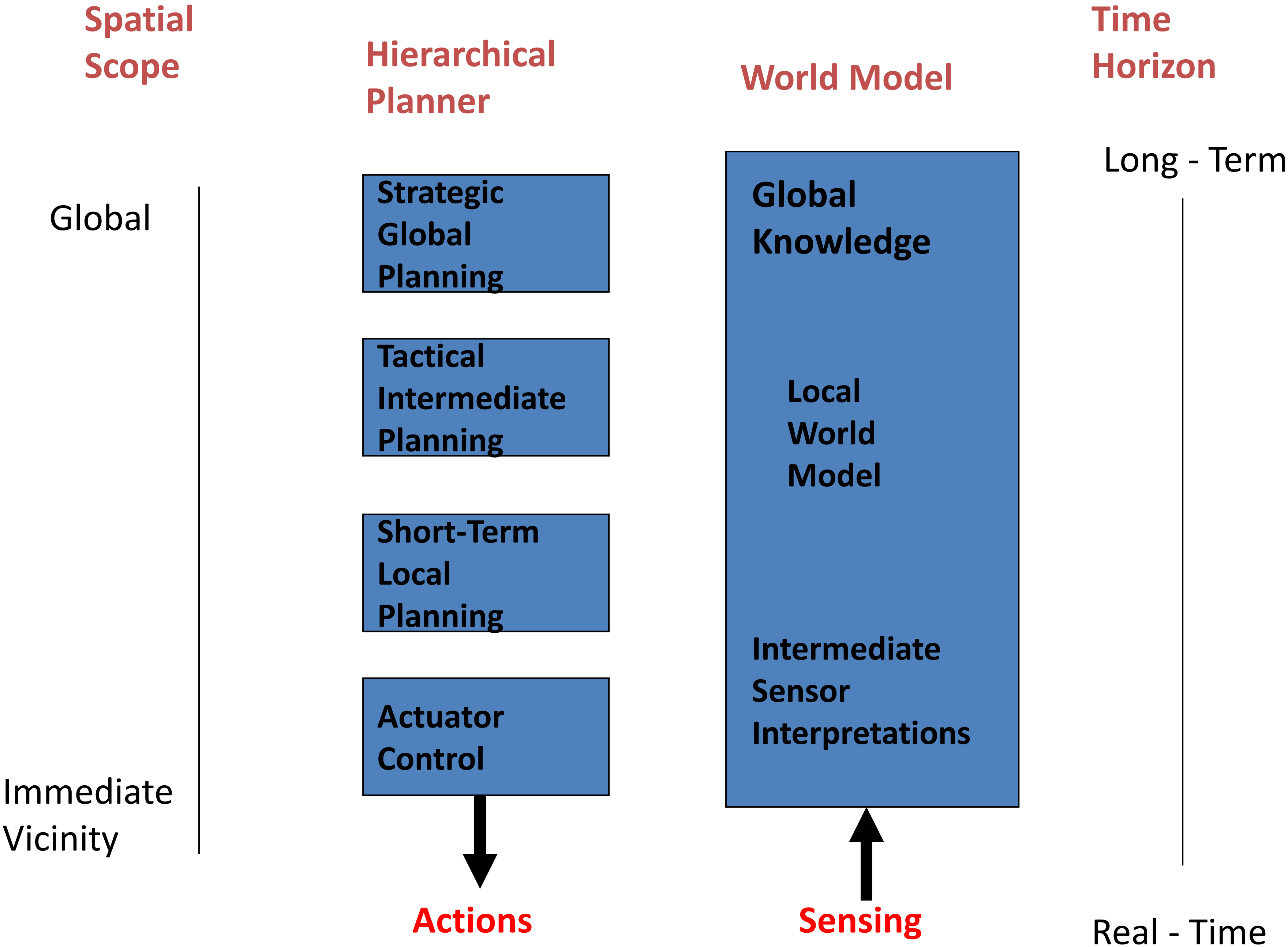
Architetture a 3 Livelli

Xavier Architecture (1995)

Task Planning (Prodigy)
Path Planning (Decision-Theoretic)
Map-Based Navigation (POMDPs)
Local Obstacle Avoidance (Curvature Velocity Method)
Servo-Control (Commercial)

Pianificazione Deliberativa

- Are often aligned with hierarchical control community within robotics.
- **Hierarchical planning** systems typically share a structured and clearly identifiable subdivision of functionality regarding to **distinct program modules** that communicate with each other in a **predictable and predetermined manner**.
- At a hierarchical planner's highest level, the **most global and least specific** plan is formulated.
- At the lowest levels, **rapid real-time** response is required, but **the planner is concerned only** with its immediate surroundings and has lost the sight of the big picture.


Spatial Scope
Hierarchical Planner
World Model
Time Horizon
Global
Immediate Vicinity
Strategic Global Planning
Tactical Intermediate Planning
Short-Term Local Planning
Actuator Control
Global Knowledge
Local World Model
Intermediate Sensor Interpretations
Long - Term
Real - Time
Actions
Sensing

Planning as Search

- Planning is looking ahead, **searching**
- The goal is a state.
- The robot's entire state space is enumerated, and searched, from the current state to the goal state.
- Different paths are tried until one is found that reaches the goal.
- If the optimal path is desired, **then all possible paths must be considered** in order to find the best one.

SPA = Planner-based

- Planner-based (deliberative) architectures typically involve three generic sequential steps or functional modules:

  1) **sensing (S)**

  2) **planning (P)**

  3) **acting (A), executing the plan**

- Thus, they are called SPA architectures.
- **SPA has serious drawbacks.**

Problem 1:
Time Scale

- It takes a **very** (prohibitively) long time to search in a real robot's state space, as that space is typically very large.
- Real robots may **have collections of simple digital sensors** (e.g., switches, IRs), a **few more complex ones** (e.g., cameras), or **analog sensors** (e.g., encoders, gauges, etc.)
- => "too much information"
- => **Generating a plan is slow.**

SPA = Planner-based

Problem 2: Space

- It takes a **lot of** space (memory) **to represent** and manipulate the robot's state space representation.
- The representation must **contain all information needed for planning.**
- => Generating a plan can be large.
- Space is not nearly as much of a problem as time, in practice.

Problem 3: Information

- The **planner assumes** that the representation of the state space **is accurate and up-to-date**.
- => The representation must be **constantly updated and checked**
- The more information, the better.
- => "too little information"

SPA = Planner-based

Problem 4: Use of Plans

The resulting plan is only useful if:

a) the environment **does not change** during the **execution of a plan** in a way that affects the plan.

b) the representation **was accurate enough** to generate a correct plan.

c) the robot's effectors **are accurate enough** to perfectly execute each **step of the plan** in order to make the **next step possible**

Deliberation in Summary

- In short, deliberative (SPA, planner-based) approaches:
  - require **search and planning**, which are slow
  - encourage open-loop plan execution, which is limiting and dangerous
- Note that **if planning were not slow** (computationally expensive) then execution would not need to be open-loop, since re-planning could be done.

Hierarchical Planners vs. BBS

Hierarchical Planners

- Rely heavily on world models,
- Can readily integrate world knowledge,
- Have a broad perspective and scope.

BB Control Systems

- afford modular development,
- Real-time robust performance within a changing world,
- Incremental growth
- are tightly coupled with arriving sensory data.

Hybrid Control

- **The basic idea is simple**: we want the best of both worlds (if possible).
- The goal is to **combine** closed-loop and open-loop **execution.**
- That means to **combine reactive and deliberative control.**
- This implies combining the **different time-scales and representations.**
- This mix is called hybrid control.

Hybrid robotic architectures believe that a union of deliberative and behavior-based approaches can potentially yield the best of both worlds.

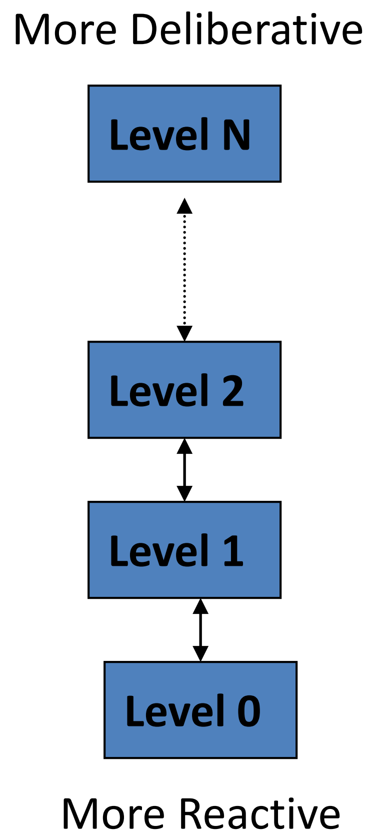
Organizing Hybrid Systems

Planning and reaction can be tied:

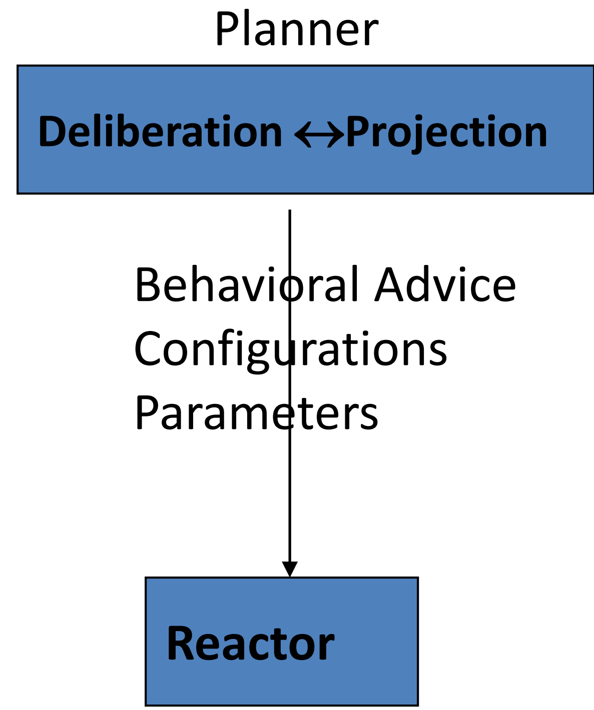
A: hierarchical integration - planning and reaction are involved with different activities, time scales

B: Planning to guide reaction - configure and set parameters for the reactive control system.

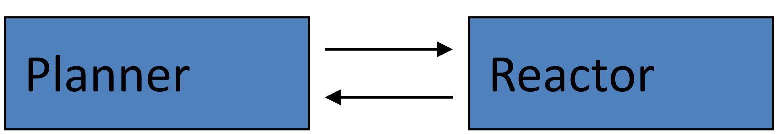
C: coupled - concurrent activities



A



B



C

Organizing Hybrid Systems

It was observed that the emerging architectural design of choice is:

– multi-layered hybrid comprising of

* a **top-down** planning system and

* a **lower-level** reactive system.

– **the interface** (middle layer between the two components) **design** is a central issue in differentiating different hybrid architectures.

In summary, a modern hybrid system typically consists of three components:

- ♦ **a reactive layer**
- ♦ **a planner**
- ♦ **a layer that puts the two together.**

=> Hybrid architectures are often called **three-layer architectures**.

The Magic Middle: Executive Control

- The middle layer has a hard job:
  1) compensate for the limitations of both the planner and the reactive system
  2) reconcile their different time-scales.
  3) deal with their different representations.
  4) reconcile any contradictory **commands** between the two.
- This is **the challenge** of hybrid systems

  => **achieving the right compromise between the two ends.**

The middle layer services.

Reusing Plans

- **Some frequently useful planned decisions may need to be reused**, so to avoid planning, an intermediate layer **may cache** and look those up. These can be:
  - **intermediate-level actions** (ILAs): stored in contingency tables.
  - **macro operators**: plans compiled into more general operators for future use.

Dynamic Re-planning

- Reaction **can influence** planning.
- Any **"important" changes** discovered by the low-level controller are passed back to the planner **in a way that** the planner can use to re-plan.
- The planner is interrupted when **even a partial answer** is needed in real-time.
- The reactive controller (and thus the robot) **is stopped** if it must wait for the planner to tell it *where to go*.

The middle layer services.

Planner - Driven Reaction

- Planning can also influence reaction.
- Any **"important" optimizations** the planner discovers are passed down to the reactive controller.
- The planner’s suggestions **are used if they are** possible and safe.

=> Who has priority, planner or reactor? It depends, as we will see...

Types of “Reaction ↔ Planning” Interaction

♦ **Selection**: Planning is viewed as configuration.

♦ **Advising**: Planning is viewed as advice giving.

♦ **Adaptation**: Planning is viewed as adaptation of controller.

♦ **Postponing**: Planning is viewed as a least commitment process.

Universal Plans

- Suppose for a given problem, **all possible plans are generated for all possible situations in advance**, and stored.
- If for each situation a robot has a pre-existing optimal plan, **it can react optimally**, be reactive and optimal.
- It has a universal plan (These are complete reactive mappings).

Viability of Universal Plans

- A system with a universal plan **is reactive**; the planning is done at compile-time, **not at run-time**.
- Universal plans are **not viable in most domains**, because:
  - the **world** must be deterministic.
  - the **world** must not change.
  - the **goals** must not change.
  - the **world** is too complex (state space is too large).

Planning & Execution

- Planning
  - *Generate* a set of *actions* – **a plan** – that can transform an *initial state* of the world to a *goal state*

    [Newell and Simon, 1950s]
- Execution
  - Start at the initial state, and *perform* each action of a generated plan

Planning Problem

Newell and Simon 1956

- Given the *actions* available in a task domain.
- Given a problem specified as:
  - an initial *state* of the world,
  - a set of *goals* to be achieved.

- Find a *solution* to the problem, i.e., a *way* to transform the initial state into a new state of the world where the goal statement is true.

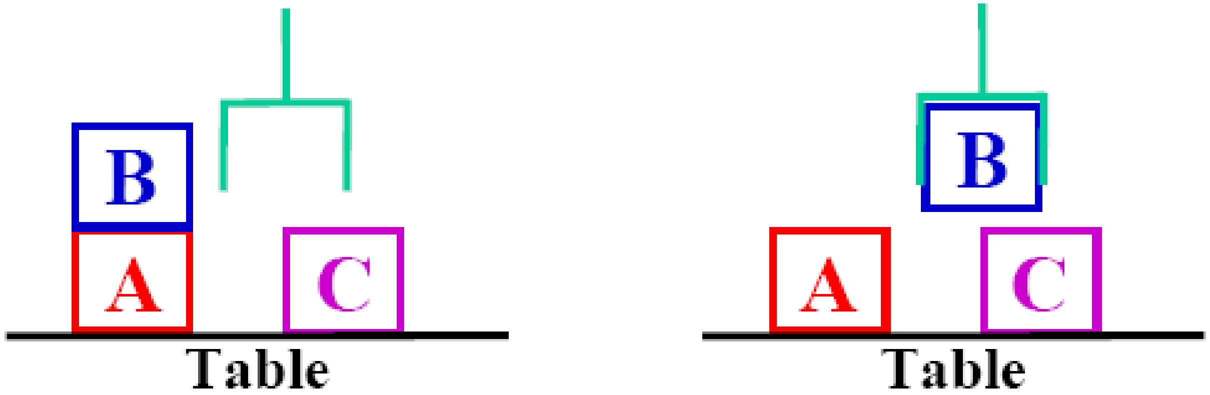
Action Model, State, Goals

Classical Planning

- Action Model: complete, deterministic, correct, rich representation
- State: single initial state, fully known
- Goals: complete satisfaction

Several different planning algorithms

Esempio: Blocks World



- Blocks are picked up and put down by the arm
- Blocks can be picked up only if they are clear, i.e., without any block on top
- The arm can pick up a block only if the arm is empty, i.e., if it is not holding another block, i.e., the arm can be pick up only one block at a time
- The arm can put down blocks on blocks or on the table

STRIPS Model

Pickup_from_table(b)

Pre: Block(b), Handempty
Clear(b), On(b, Table)

Add: Holding(b)

Delete: Handempty,
On(b, Table)

Pickup_from_block(b, c)

Pre: Block(b), Handempty
Clear(b), On(b, c), Block(c)

Add: Holding(b), Clear(c)

Delete: Handempty,
On(b, c)

Putdown_on_table(b)

Pre: Block(b), Holding(b)

Add: Handempty,
On(b, Table)

Delete: Holding(b)

Putdown_on_block(b, c)

Pre: Block(b), Holding(b)
Block(c), Clear(c), $b \neq c$

Add: Handempty, On(b, c)

Delete: Holding(b), Clear(c)

Init: On(a,Table), On(b,table), On(c,table)

Goal: On(a,table),On(b,a), On(c,b)

Spacecraft Domain

Observation-1
- target — pointing
- instruments — calibrated

Observation-2

Observation-3

Observation-4

...

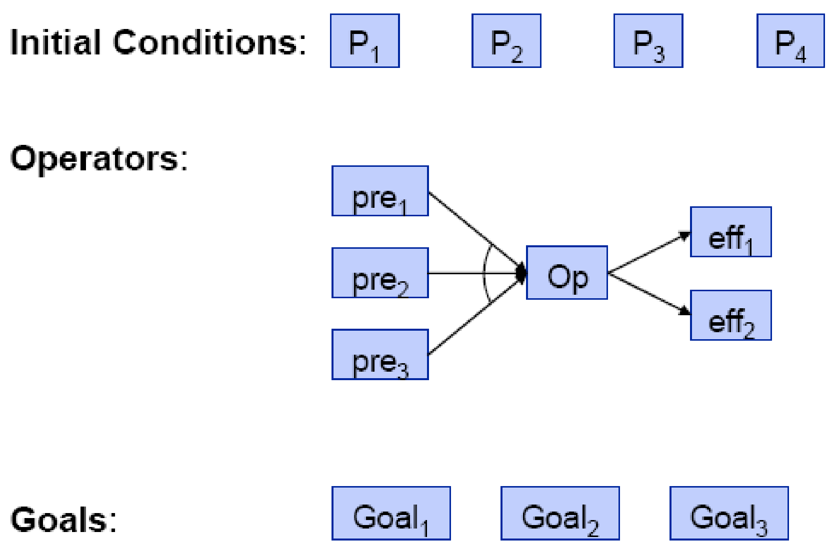
TakeImage (?target, ?instr):
- Pre: Status(?instr, Calibrated), Pointing(?target)
- Eff: Image(?target)

Calibrate (?instrument):
- Pre: Status(?instr, On), Calibration-Target(?target), Pointing(?target)
- Eff: ¬Status(?inst, On), Status(?instr, Calibrated)

Turn (?target):
- Pre: Pointing(?direction), ?direction ≠ ?target
- Eff: ¬Pointing(?direction), Pointing(?target)

Planning Problem

- **Planning Domain:** Descrizione degli operatori in termini di precondizioni ed effetti
- **Planning Problem:** Stato iniziale, Dominio, Goals



Tipi di Planning

- Classical Planning
- Temporal Planning
- Conditional Planning
- Decision Theoretic Planning
- …
- Least-Commitment Planning
- HTN planning
- …

Paradigms

Classical planning

(STRIPS, operator-based, first-principles)

"generative"

Hierarchical Task Network planning

"practical" planning

MDP & POMDP planning

planning under uncertainty

State Space vs. Plan Space

- Planning in the state space:
  - sequence of actions, from the initial state to the goal state
- Planning in the plan space:
  - Sequence of plan transformations, from an initial plan to the final one

Plan-State Search

- Search space is set of *partial plans*
- Plan is tuple <*A*, *O*, *B*>
  - *A*: Set of ***actions***, of the form $(a_i : Op_j)$
  - *O*: Set of ***orderings***, of the form $(a_i < a_j)$
  - *B*: Set of ***bindings***, of the form $(v_i = C)$, $(v_i \neq C)$, $(v_i = v_j)$ or $(v_i \neq v_j)$
- Initial plan:
  - $<\{start, finish\}, \{start < finish\}, \{\}>$
  - *start* has no preconditions; Its effects are the initial state
  - *finish* has no effects; Its preconditions are the goals

State-Space vs Plan-Space

Planning problem

Find a sequence of actions that make instance of the goal true

Nodes in search space

Standard search: node = concrete world state

Planning search: node = partial plan

(Partial) Plan consists of

- Set of operator applications S_i
- Partial (temporal) order constraints $S_i \prec S_j$
- Causal links $S_i \xrightarrow{c} S_j$

  Meaning: "S_i achieves $c \in precond(S_j)$" (record purpose of steps)

Search in the Plan-Space

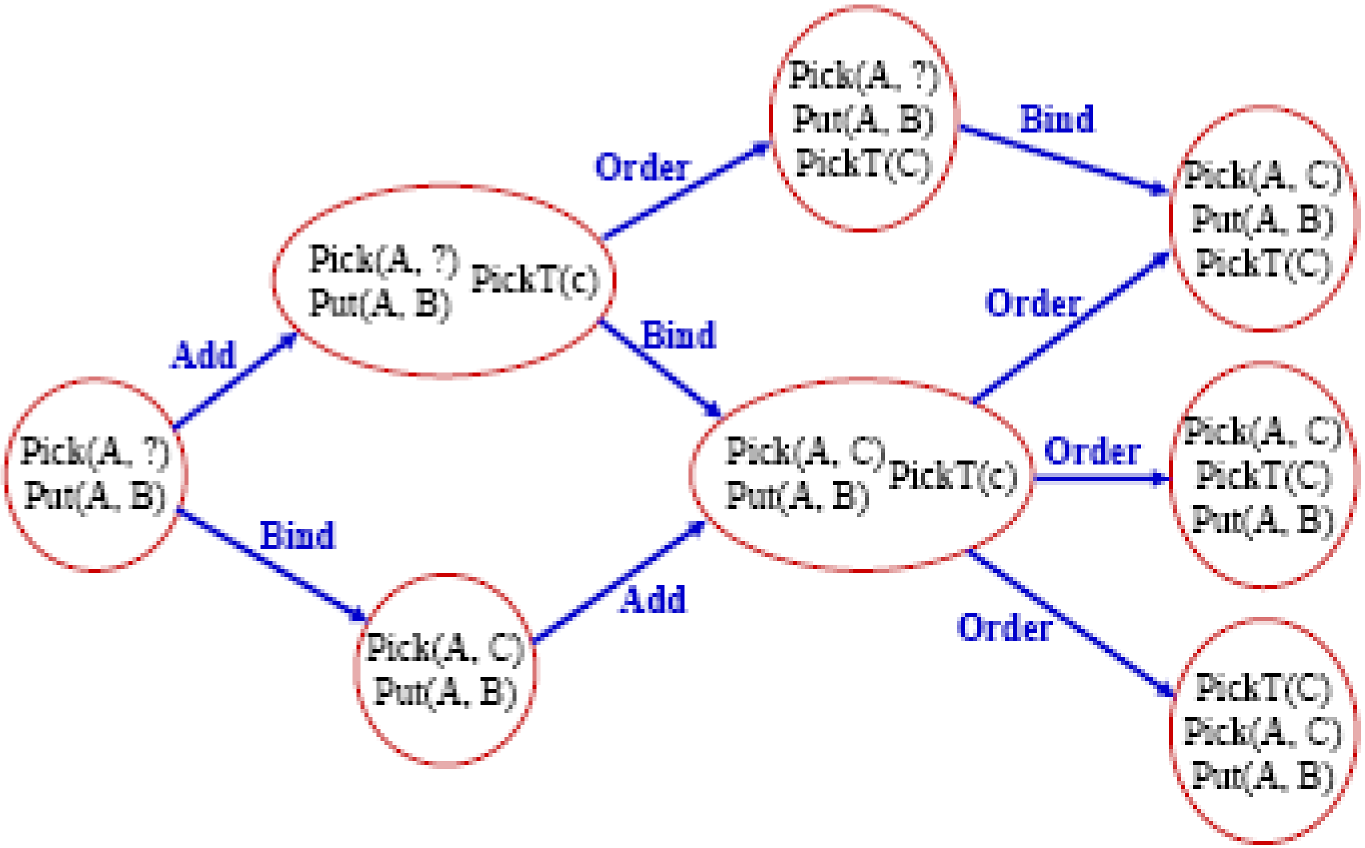
Operators on partial plans

- add an action and a causal link to achieve an open condition
- add a causal link from an existing action to an open condition
- add an order constraint to order one step w.r.t. another

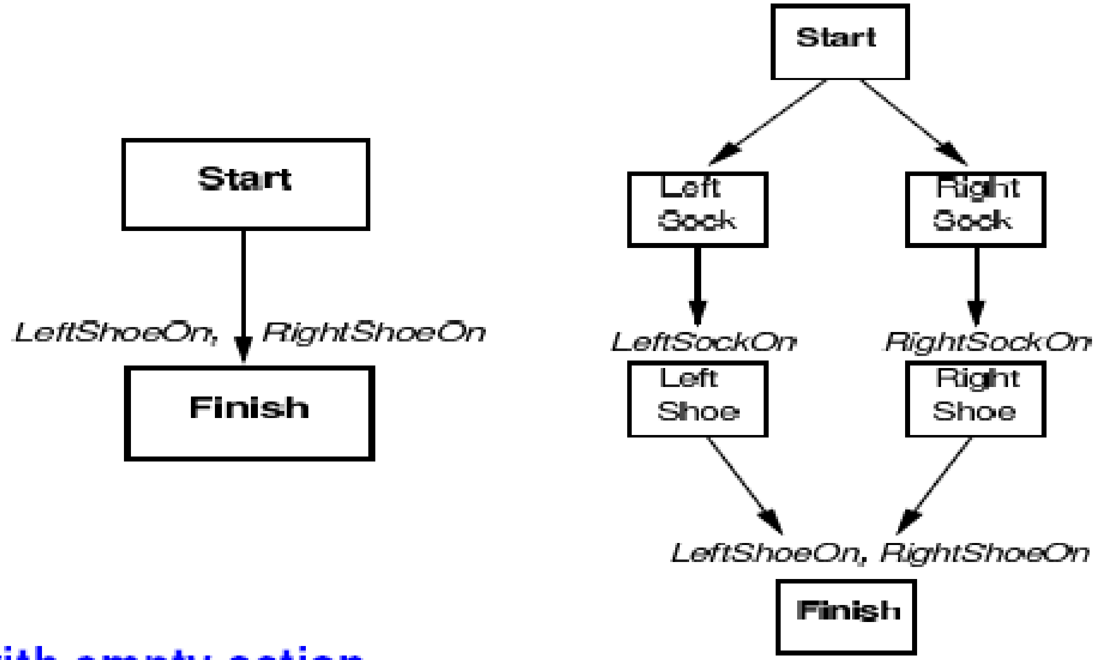
Open condition

A precondition of an action not yet causally linked

Plan-State Search



Partially-Ordered Plans



Special steps with empty action

Start no precond, initial assumptions as effect)

Finish goal as precond, no effect

Partial-Order Plans

Complete plan

A plan is complete iff every precondition is achieved

A precondition c of a step S_j is achieved (by S_i) if

- $S_i \prec S_j$
- $c \in effect(S_i)$
- there is no S_k with $S_i \prec S_k \prec S_j$ and $\neg c \in effect(S_k)$
  (otherwise S_k is called a clobberer or threat)

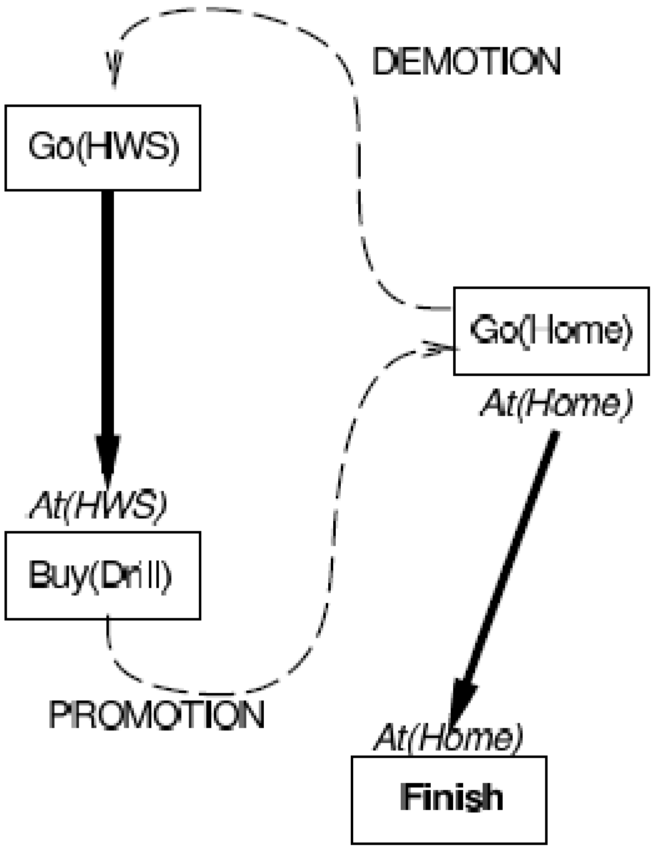
Clobberer / threat

A potentially intervening step that destroys the condition achieved by a causal link

Partial-Order Plans

Example

$Go(Home)$ **clobbers** $At(HWS)$



Demotion

Put before $Go(HWS)$

Promotion

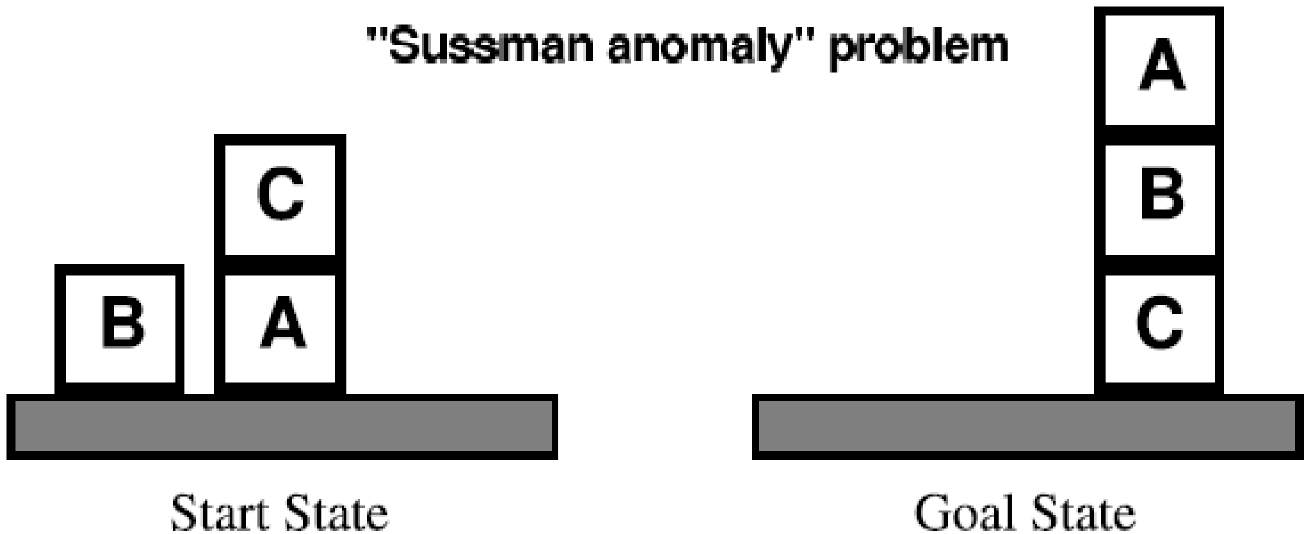
Put after $Buy(Drill)$

General Approach

- General Approach
  - Find unachieved precondition
    - Add new action *or* link to existing action
  - Determine if conflicts occur
    - Previously achieved precondition is "clobbered"
    - Fix conflicts (reorder, bind, …)
- Partial-order planning can easily (and optimally) solve blocks world problems that involve goal interactions (e.g., the "Sussman Anomaly" problem)

Blocks World

"Sussman anomaly" problem



Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

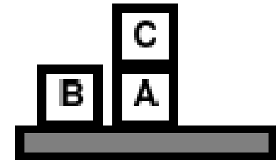
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Blocks World

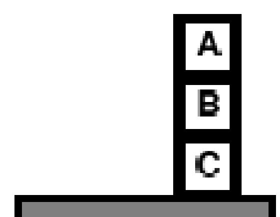
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

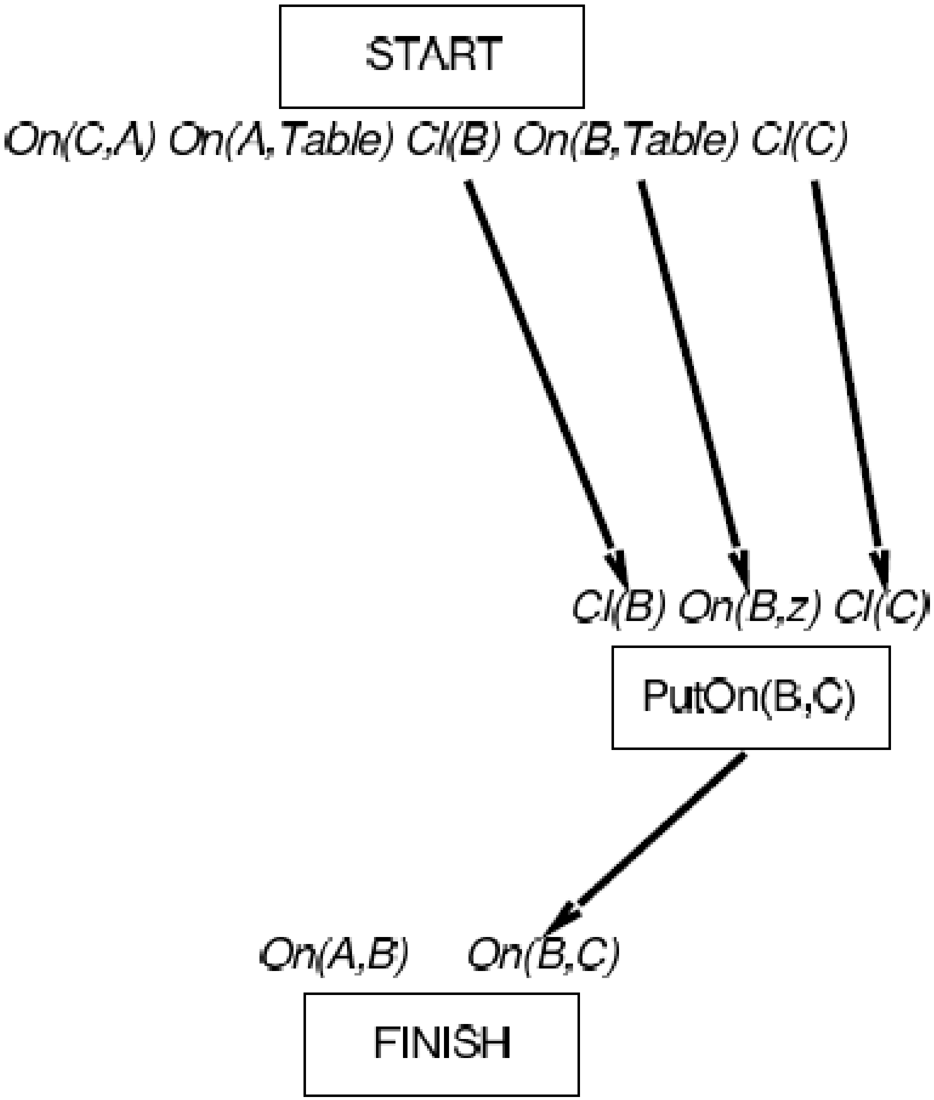


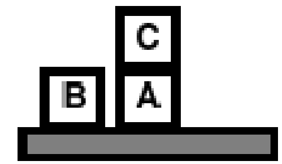
On(A,B) On(B,C)

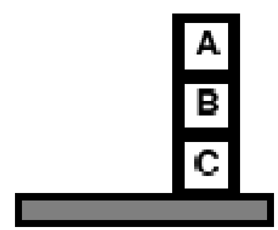
FINISH



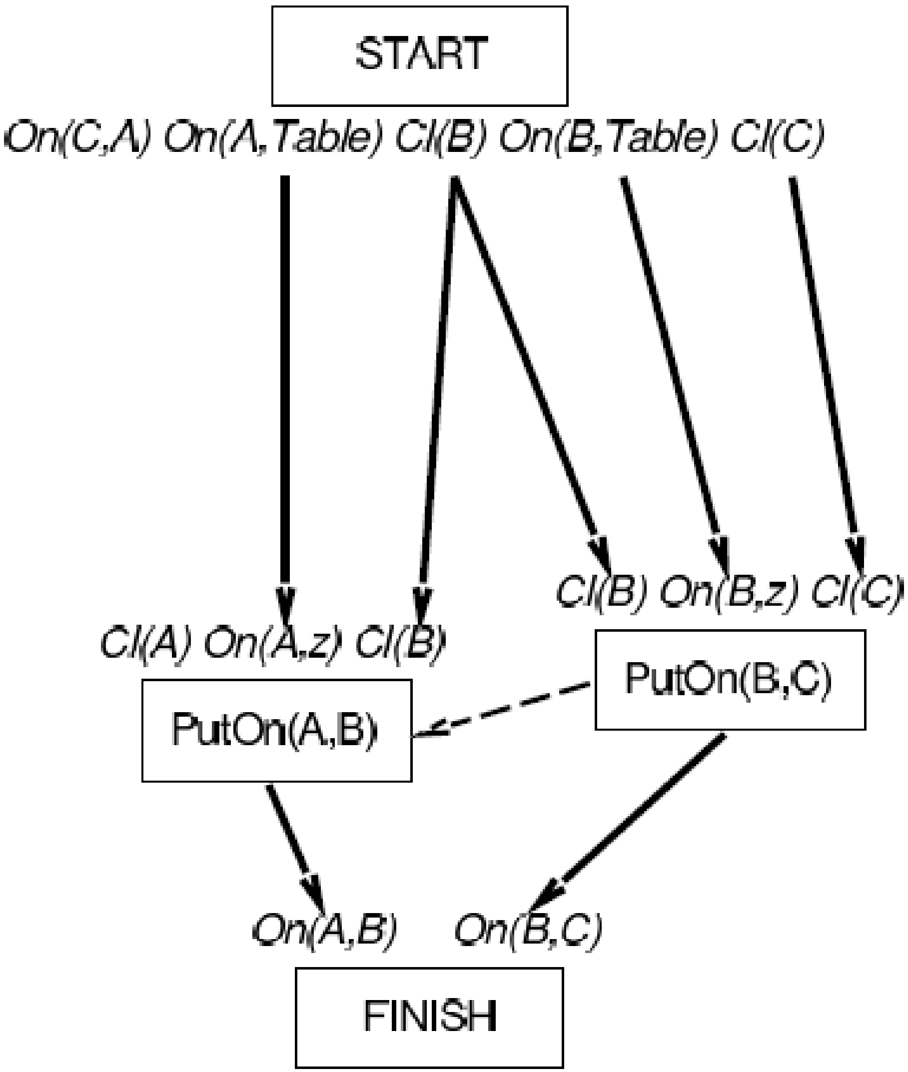
Blocks World

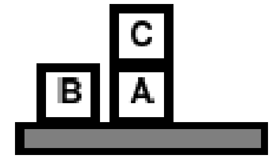




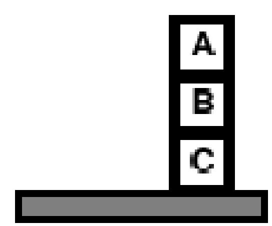


Blocks World

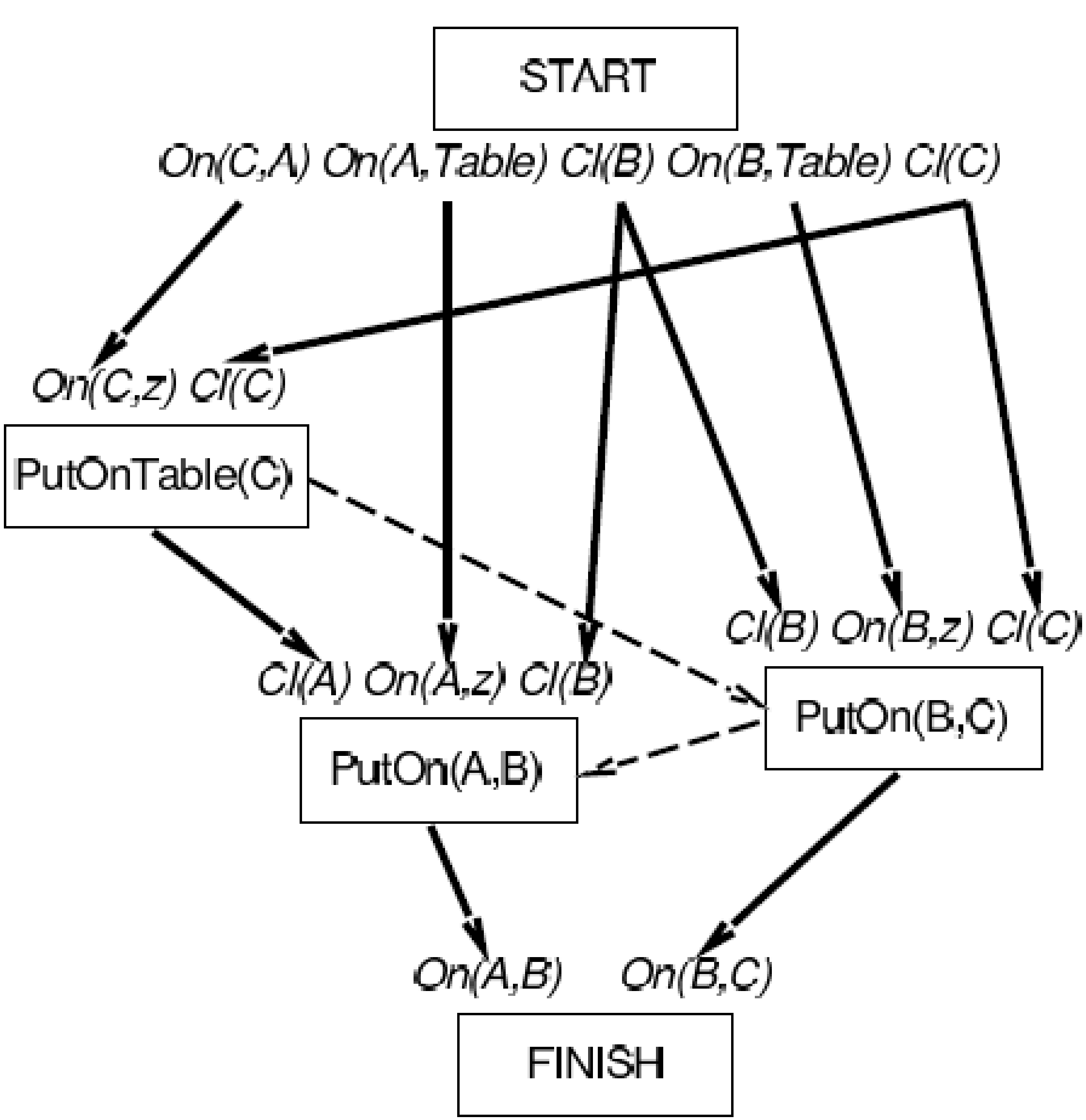


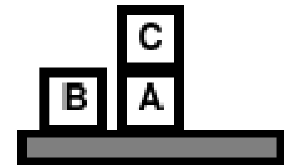


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



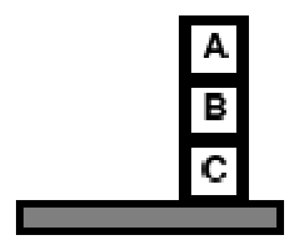
Blocks World



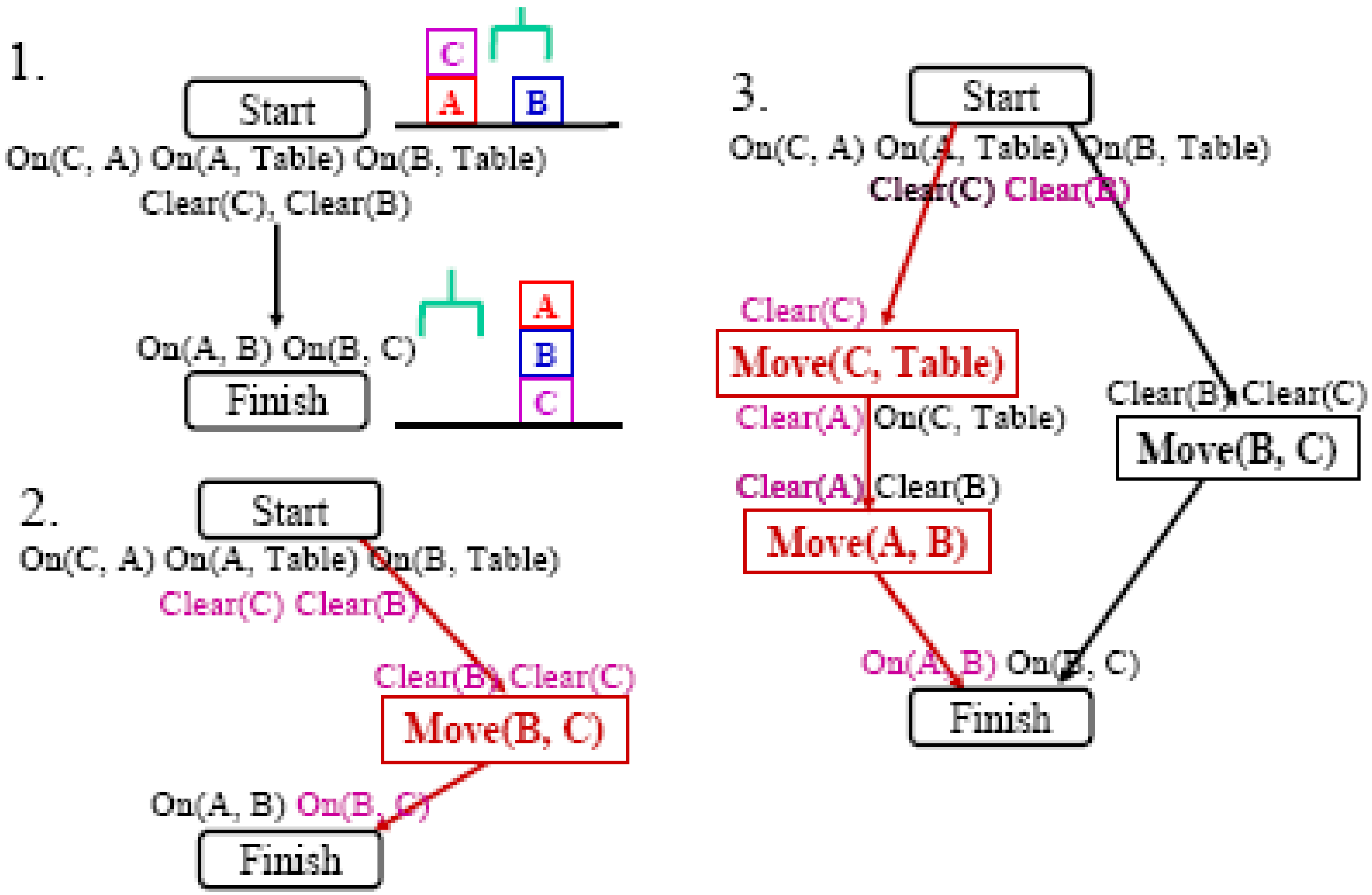


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)

PutOn(B,C)
clobbers Cl(C)
=> order after
PutOnTable(C)

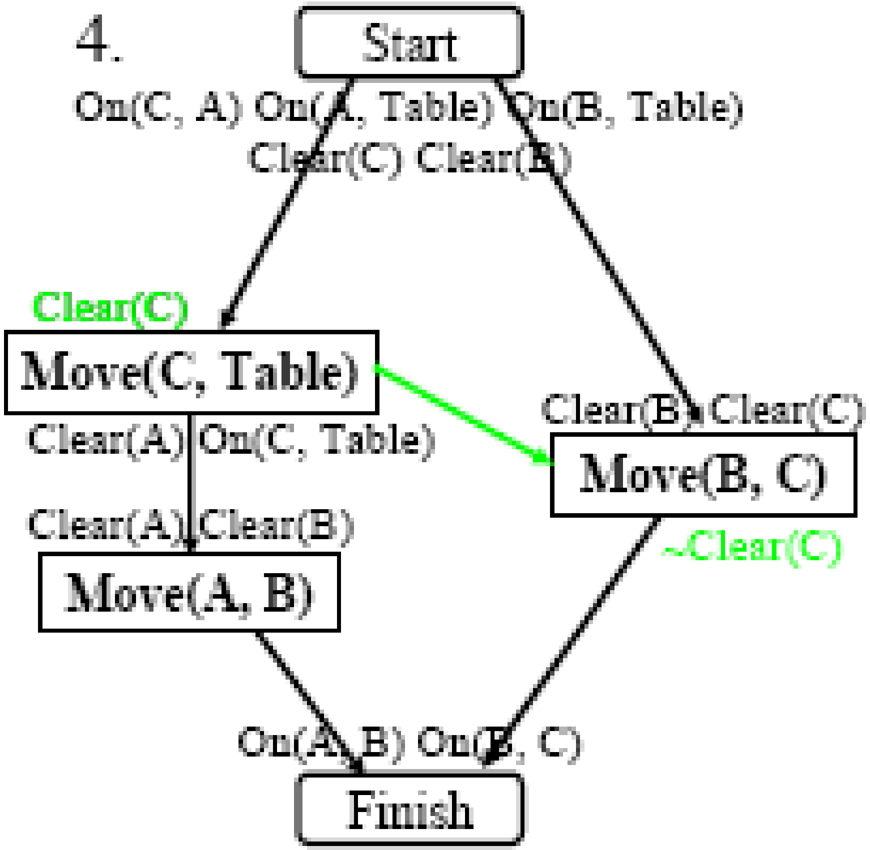


Blocks World

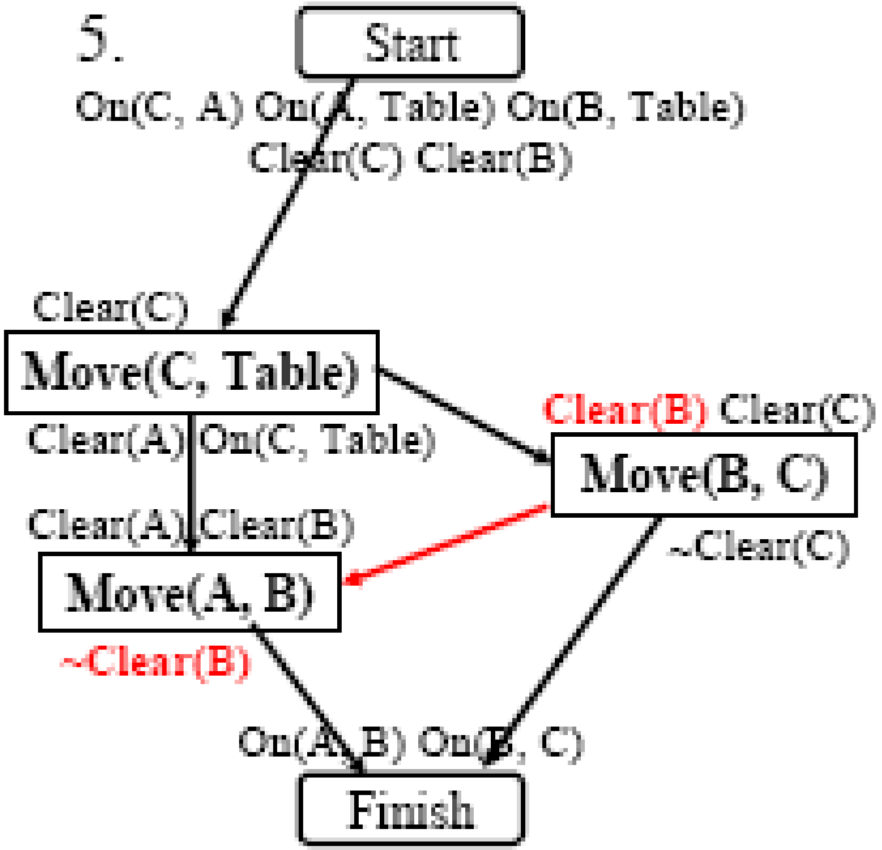


Blocks World

4.



5.



Least Commitment

- Basic Idea
  - *Make choices that are only relevant to solving the current part of the problem*
- Least Commitment Choices
  - **Orderings**: Leave actions unordered, unless they must be sequential
  - **Bindings**: Leave variables unbound, unless needed to unify with conditions being achieved
  - **Actions**: Usually not subject to "least commitment"
- Refinement
  - Only *add* information to the current plan
  - *Transformational* planning can remove choices

Terminology

- ***Totally Ordered*** Plan
  - There exists sufficient orderings O such that all actions in A are ordered with respect to each other
- ***Fully Instantiated*** Plan
  - There exists sufficient constraints in B such that all variables are constrained to be equal to some constant
- ***Consistent*** Plan
  - There are no contradictions in O or B
- ***Complete*** Plan
  - Every precondition p of every action a_i in A is *achieved*: There exists an effect of an action a_j that comes before a_i and unifies with p, and no action a_k that deletes p comes between a_j and a_i

POP-Algorithm

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*   % complete and consistent
  S_{need}, c ← SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return S_{need}, c

POP-Algorithm

procedure CHOOSE-OPERATOR(*plan, operators*, S_{need}, *c*)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

POP-Algorithm

procedure RESOLVE-THREATS(*plan*)

    for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**

        choose either

            Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)

            Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)

        if not CONSISTENT(*plan*) **then fail**

    end

POP-Algorithm

- Non-deterministic search for plan,
  backtracks over choicepoints on failure:
  - choice of S_{add} to achieve S_{need}
  - choice of promotion or demotion for clobberer
- Sound and complete
- There are extensions for:
  disjunction, universal quantification, negation, conditionals
- Efficient with good heuristics from problem description
  But: very sensitive to subgoal ordering
- Good for problems with loosely related subgoals

POP-Algorithm

- **Advantages**
  - Partial order planning is ***sound*** and ***complete***
  - Typically produces ***optimal*** solutions (plan length)
  - Least commitment may lead to shorter search times
- **Disadvantages**
  - Significantly more complex algorithms (higher *per-node* cost)
  - Hard to determine what is true in a state
  - Larger search space (**infinite!**)

Plan Monitoring

Execution monitoring

Failure: Preconditions of remaining plan not met

Action monitoring

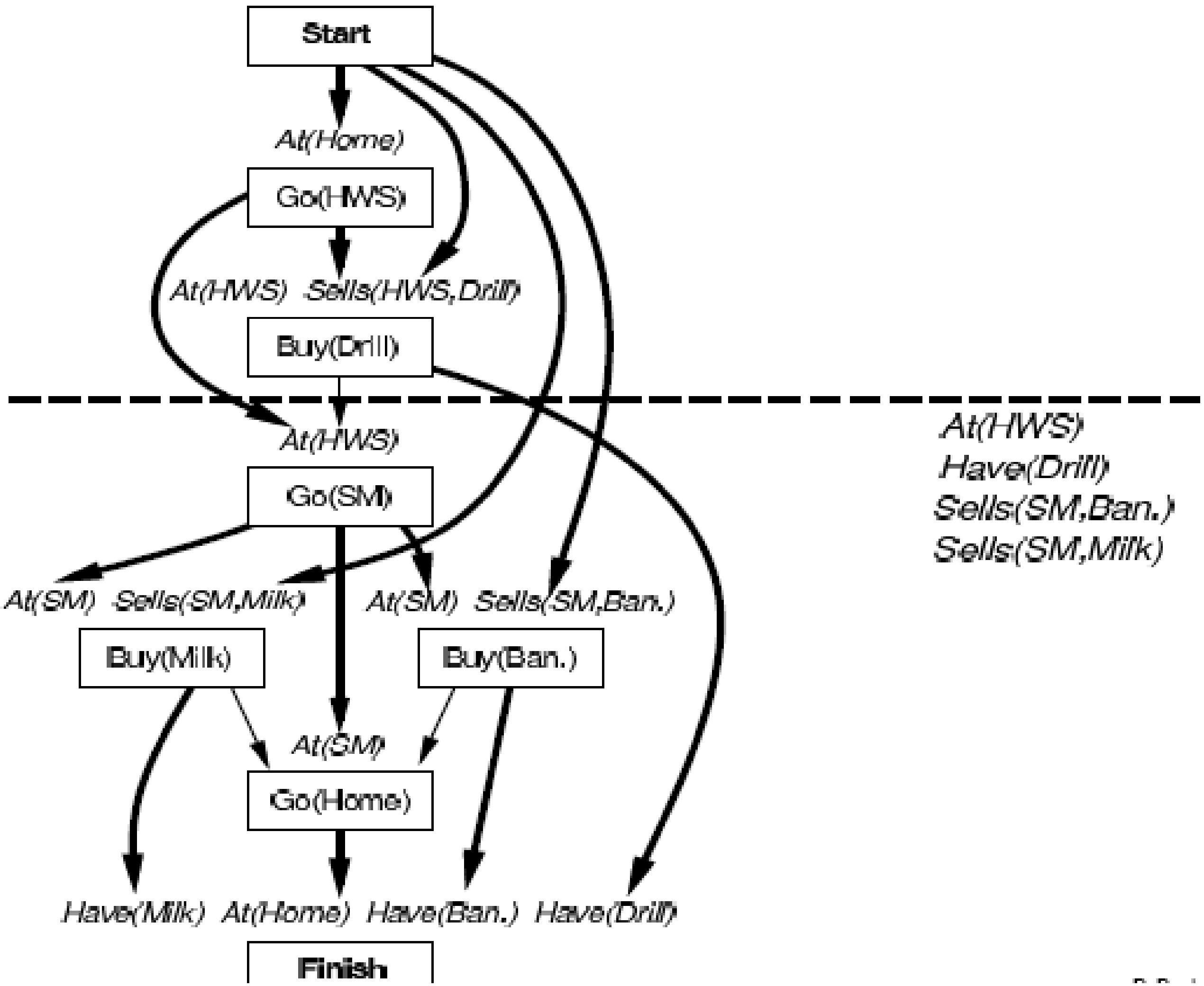
Failure: Preconditions of next action not met

(or action itself fails, e.g., robot bump sensor)

Consequence of failure

Need to **replan**

Preconditions for the rest of the plan



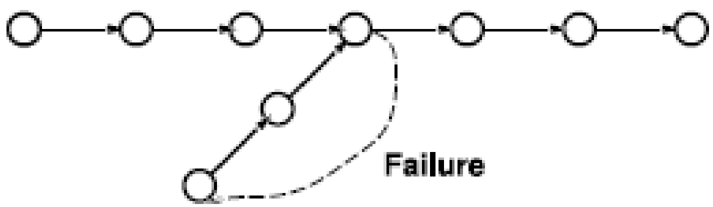
Replanning

Simplest

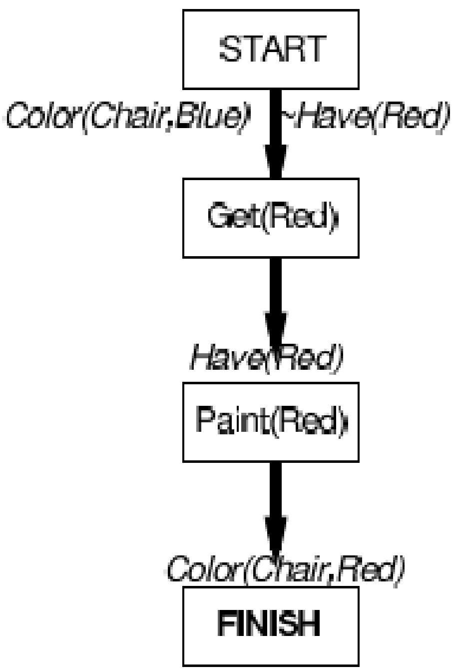
On failure, replan from scratch

Better

Plan to get back on track by reconnecting to best continuation



Replanning



PRECONDITIONS	FAILURE RESPONSE
none	N/A
Have(Red)	Fetch more red
Color(Chair,Red)	Repaint

Classical Planning: Limits

Instantaneous actions

No temporal constraints

No concurrent actions

No continuous quantities

Spacecraft Domain

Observation-1
- priority
- time window
- target
- instruments
- duration

Observation-2

Observation-3

Observation-4

...

Objective:
maximize science return

Spacecraft Domain

Observation-1
- priority
- time window
- target
- instruments
- duration

Observation-2

Observation-3

Observation-4

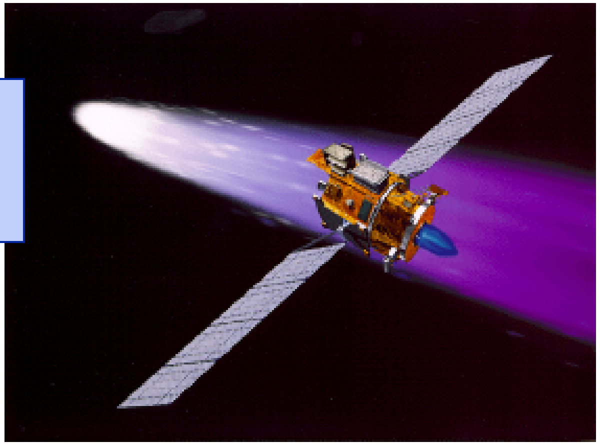
…

linked

angle between targets
$\Rightarrow$ turn duration

calibration
- target1
- target2
- …

consumables:
- fuel
- power
- data storage
- cryogen



Objective:
maximize science return

Based on slides by Dave Smith, NASA Ames

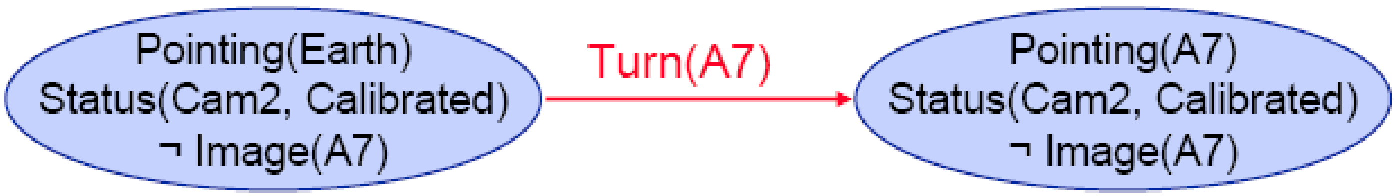
Extensions

- Time
- Resources
- Constraints
- Uncertainty
- Utility
- …

Model

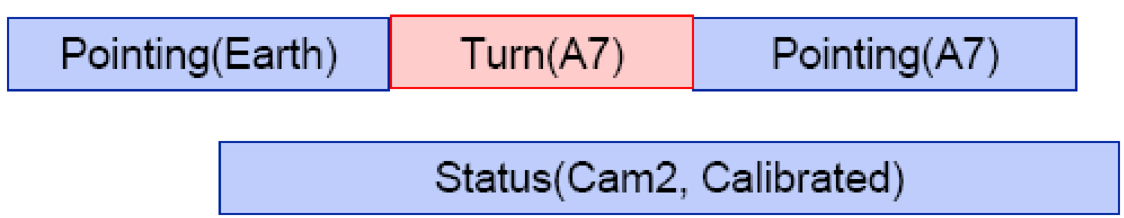
State-centric (Mc Carthy):

for each time describe propositions that are true



History-based (Hayes):

for each proposition describe times it is true



Temporal Interval Relations

A before B

A meets B

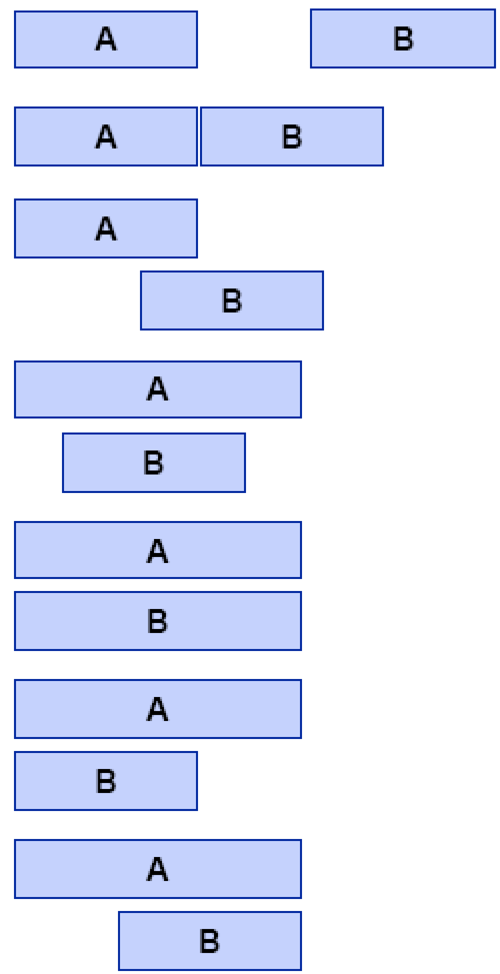
A overlaps B

A contains B

A = B

A starts B

A ends B



Based on slides by Dave Smith, NASA Ames

Temporal Operators

TakeImage (?target, ?instr):
    Pre: Status(?instr, Calibrated), Pointing(?target)
    Eff: Image(?target)

TakeImage (?target, ?instr)

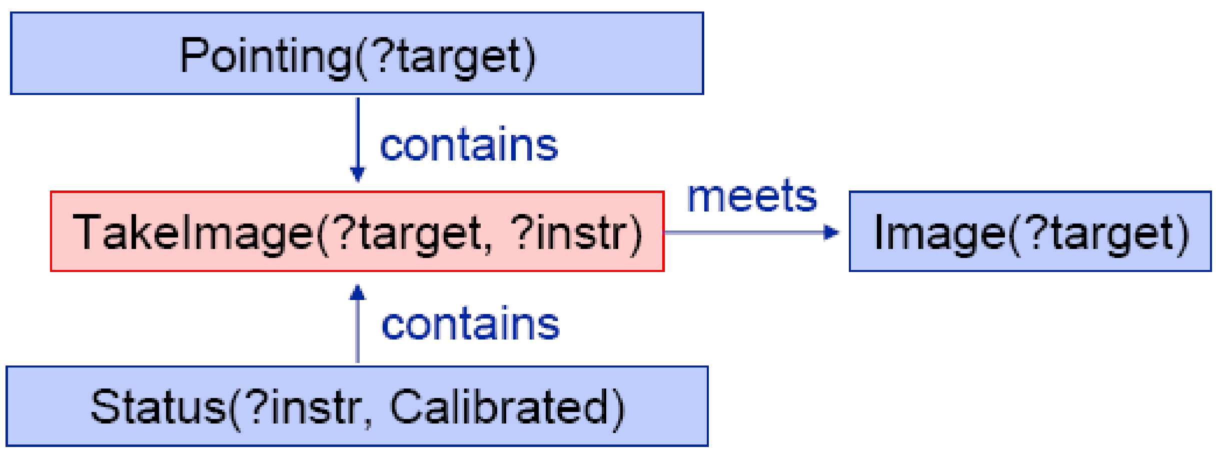
contained-by	Status(?instr, Calibrated)
contained-by	Pointing(?target)
meets	Image(?target)

Based on slides by Dave Smith, NASA Ames

Temporal Operators

TakeImage (?target, ?instr)

contained-by	Status(?instr, Calibrated)
contained-by	Pointing(?target)
meets	Image(?target)



Based on slides by Dave Smith, NASA Ames

Temporal Operators

TakeImage (?target, ?instr)

contained-by	Status(?instr, Calibrated)
contained-by	Pointing(?target)
meets	Image(?target)

$$\Downarrow$$

$$\text{TakeImage}(?\text{target}, ?\text{instr})_A$$

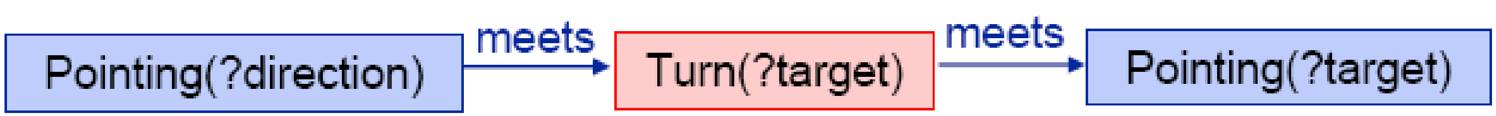
$$\Rightarrow \exists P\ \{\text{Status}(?\text{instr}, \text{Calibrated})_P \ \wedge\ \text{Contains}(P, A)\}$$

$$\wedge\ \exists Q\ \{\text{Pointing}(?\text{target})_Q \ \wedge\ \text{Contains}(Q, A)\}$$

$$\wedge\ \exists R\ \{\text{Image}(?\text{target})_R \ \wedge\ \text{Meets}(A, R)\}$$

Temporal Operators

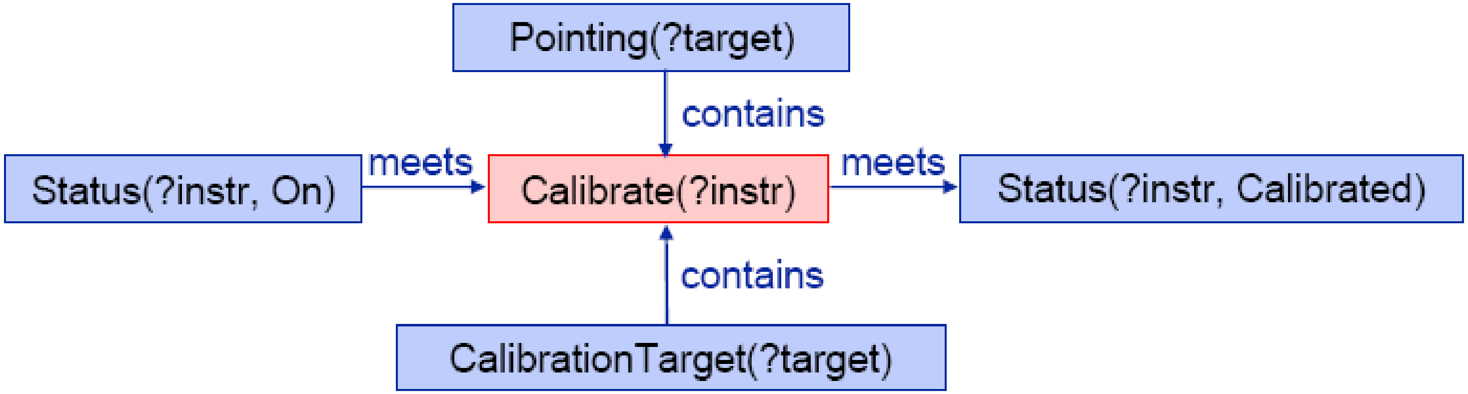
Turn (?target)
met-by Pointing(?direction)
meets Pointing(?target)



Temporal Operators

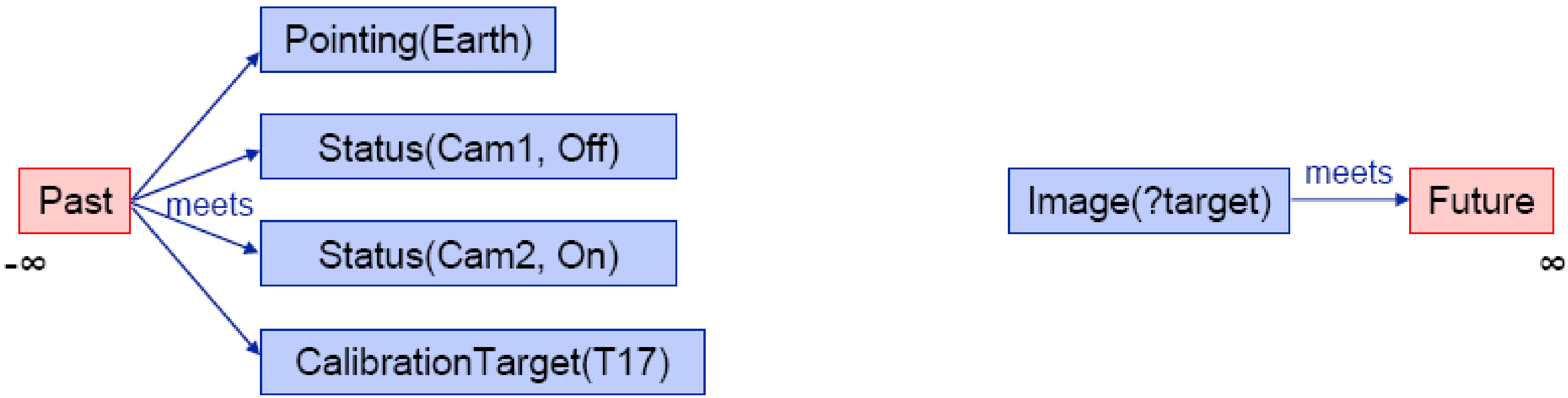
Calibrate (?instr)

met-by	Status(?instr, On)
contained-by	CalibrationTarget(?target)
contained-by	Pointing(?target)
meets	Status(?instr, Calibrated)



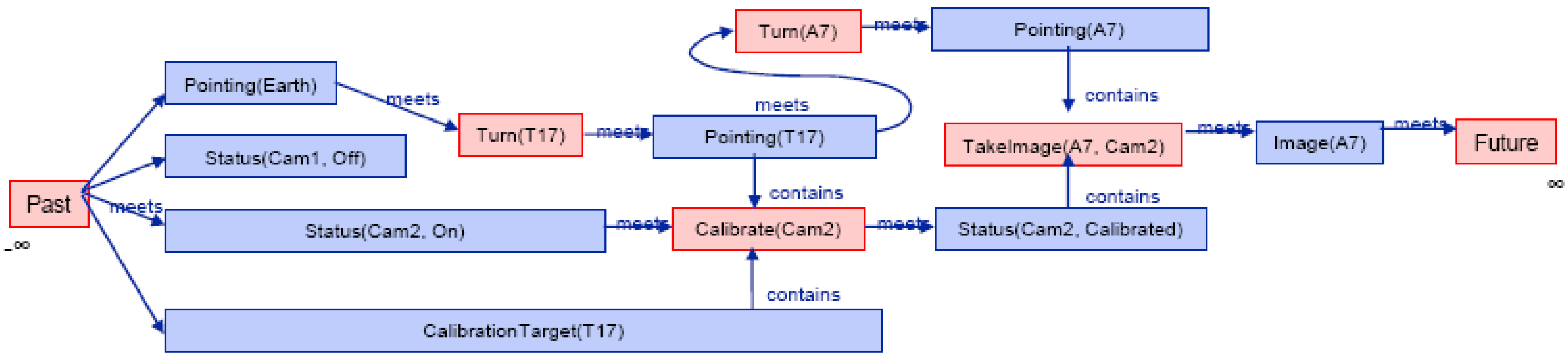
Based on slides by Dave Smith, NASA Ames

Temporal Planning Problem



Based on slides by Dave Smith, NASA Ames

Consistent Complete Plan



Based on slides by Dave Smith, NASA Ames

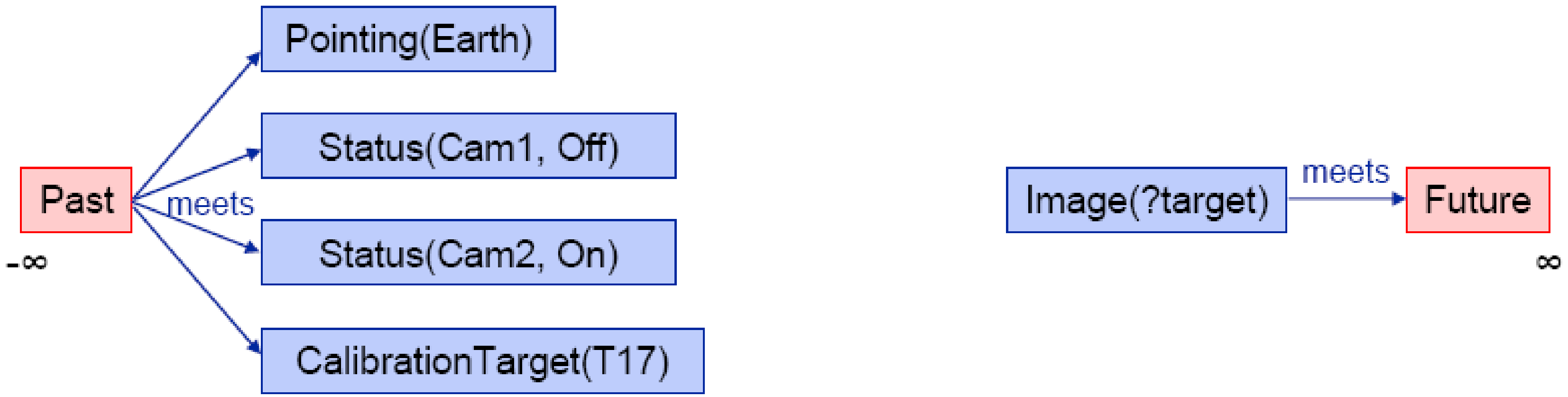
CBI-Planning

Choose:

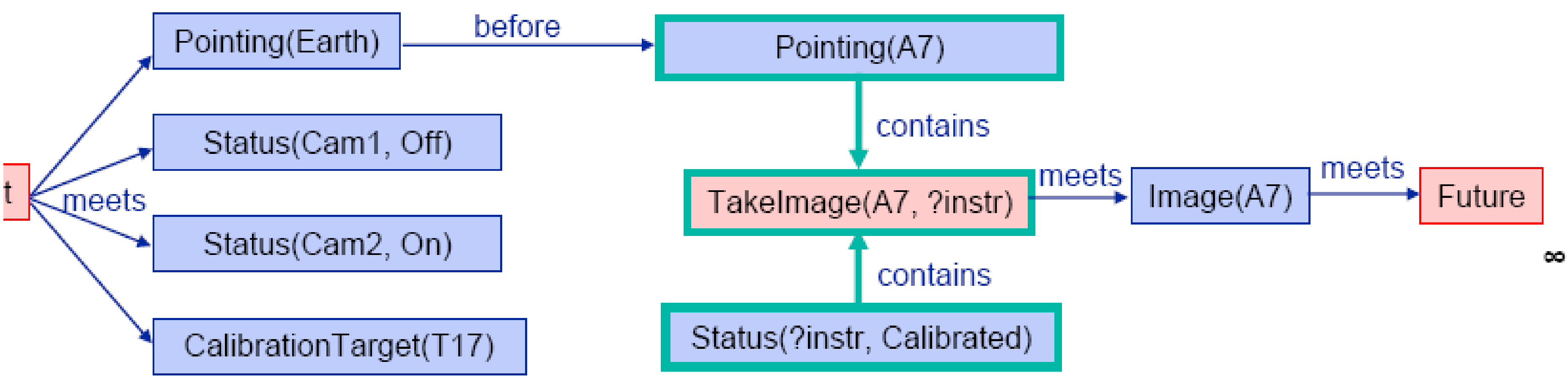
- introduce an action & instantiate constraints
- coalesce propositions

Propagate constraints

Initial Plan

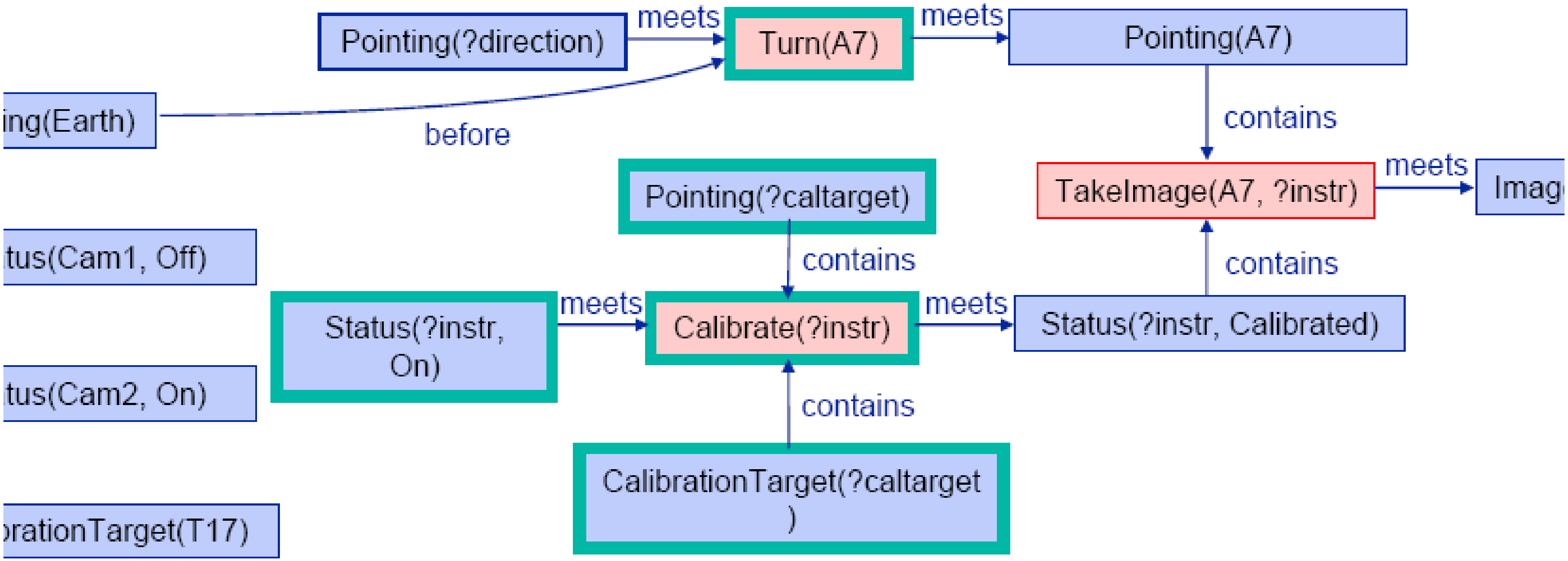


Expansion



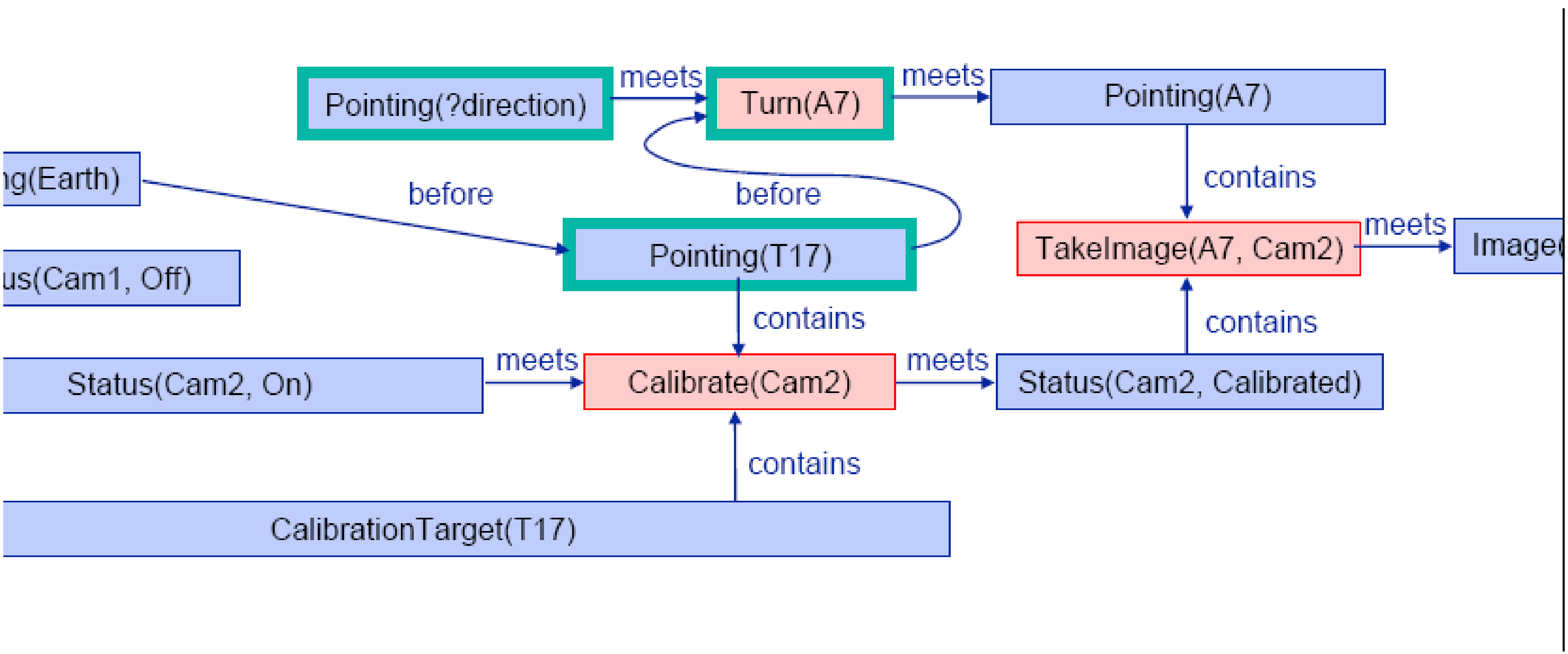
Pointing(Earth)
before
Pointing(A7)
Status(Cam1, Off)
contains
t
meets
Status(Cam2, On)
TakeImage(A7, ?instr)
meets
Image(A7)
meets
Future
∞
contains
CalibrationTarget(T17)
Status(?instr, Calibrated)

Expansion



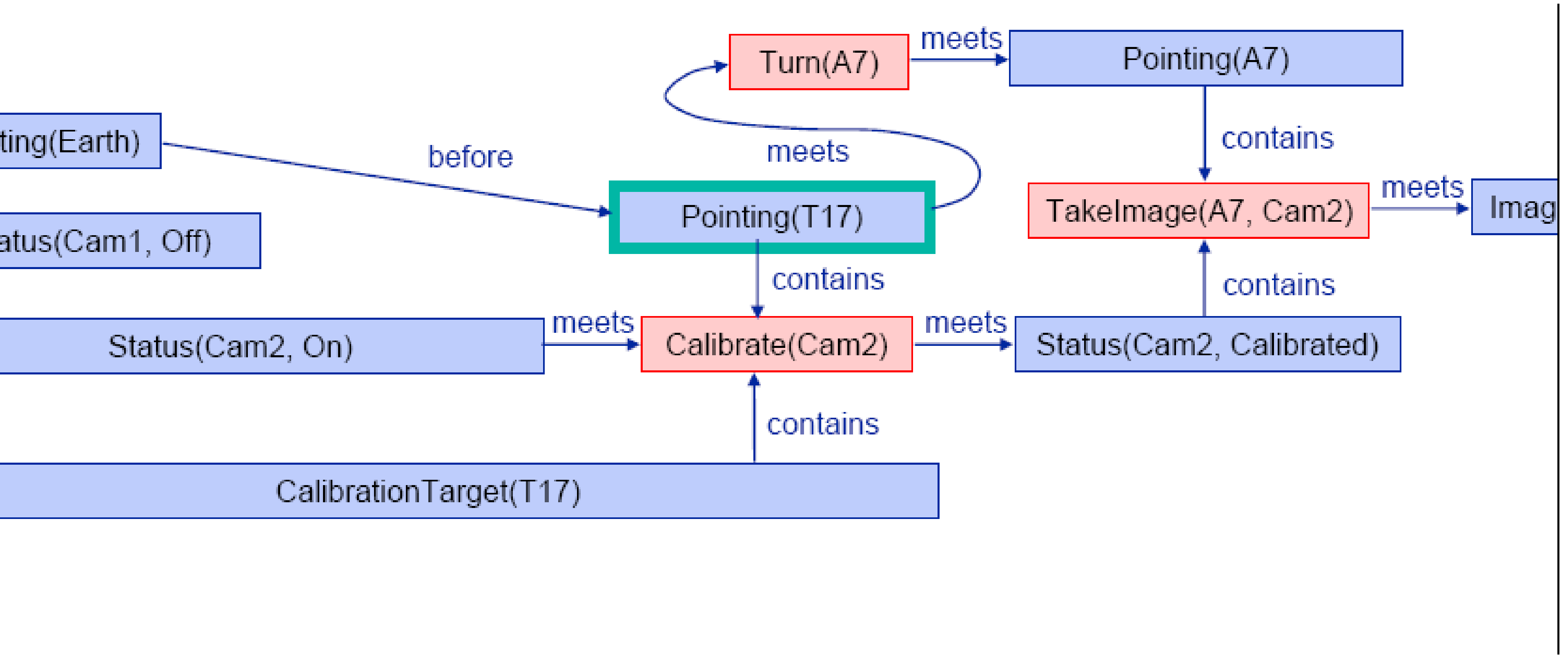
Based on slides by Dave Smith, NASA Ames

Coalescing



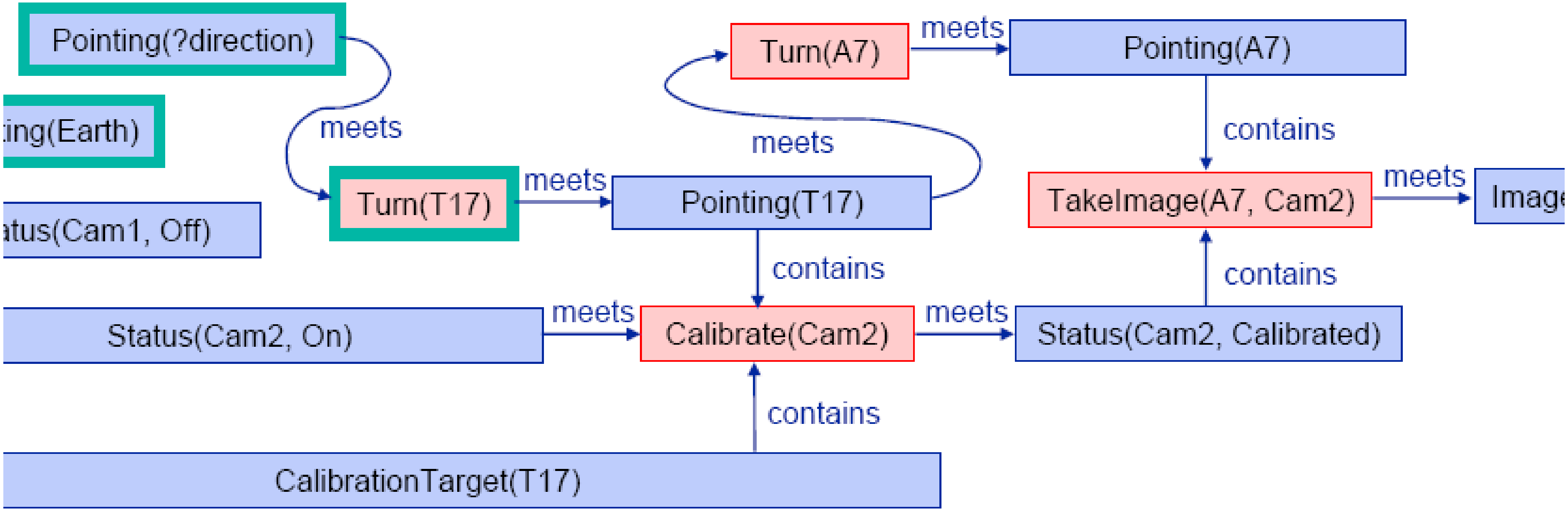
Pointing(?direction)
meets
Turn(A7)
meets
Pointing(A7)
ng(Earth)
before
before
contains
us(Cam1, Off)
Pointing(T17)
TakeImage(A7, Cam2)
meets
Image(
contains
contains
meets
meets
Status(Cam2, On)
Calibrate(Cam2)
Status(Cam2, Calibrated)
contains
CalibrationTarget(T17)

Coalescing



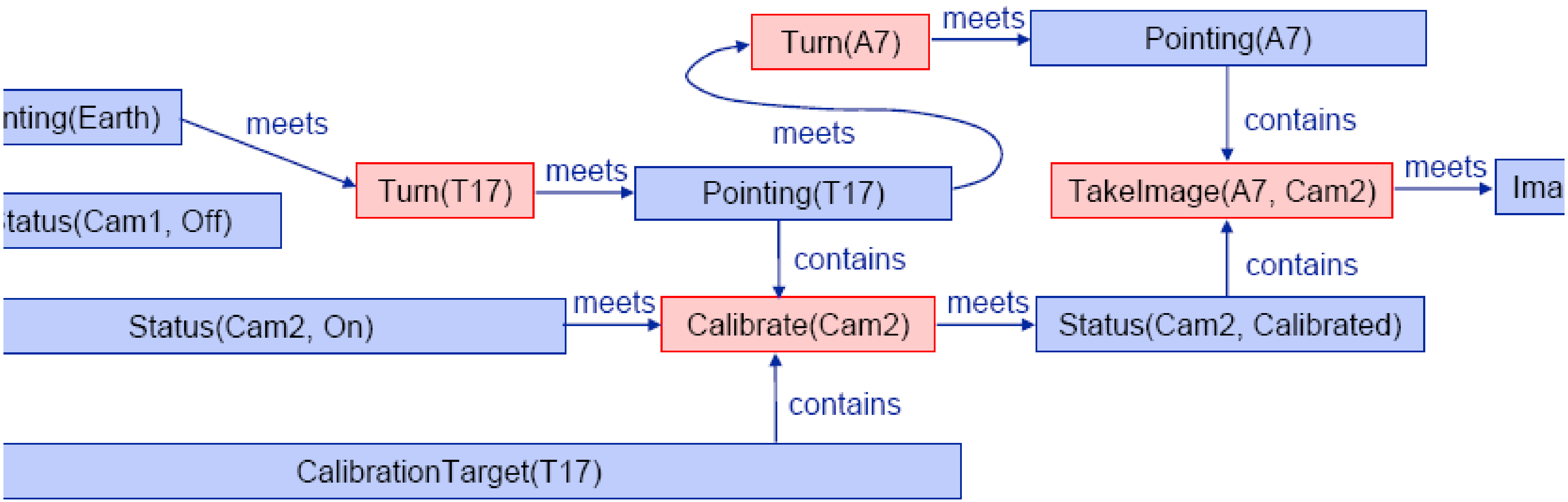
Turn(A7)
meets
Pointing(A7)
ting(Earth)
before
meets
contains
TakeImage(A7, Cam2)
meets
Imag
atus(Cam1, Off)
Pointing(T17)
contains
contains
Status(Cam2, On)
meets
Calibrate(Cam2)
meets
Status(Cam2, Calibrated)
contains
CalibrationTarget(T17)

Expansion



Pointing(?direction)
ing(Earth)
atus(Cam1, Off)
Status(Cam2, On)
CalibrationTarget(T17)
meets
Turn(T17)
meets
Pointing(T17)
meets
Turn(A7)
meets
Pointing(A7)
contains
TakeImage(A7, Cam2)
meets
Image
contains
contains
Calibrate(Cam2)
meets
meets
Status(Cam2, Calibrated)
contains
contains

Coalescing



Turn(A7)
meets
Pointing(A7)
nting(Earth)
meets
meets
contains
Turn(T17)
meets
Pointing(T17)
TakeImage(A7, Cam2)
meets
Ima
tatus(Cam1, Off)
contains
contains
Status(Cam2, On)
meets
Calibrate(Cam2)
meets
Status(Cam2, Calibrated)
contains
CalibrationTarget(T17)

CBI-Algorithm

Expand(TQAs, constraints)

1. If the constraints are inconsistent, **fail**
2. If all TQAs have causal explanations, **return**(TQAs, constraints)
3. Select a $g \in$ TQAs with no causal explanation
4. **Choose**:

    Choose another $p \in$ TQAs such that g can be coalesced with p under constraints C

        Expand(TQAs-g, constraints $\cup$ C)

    Choose an action that would provide a causal explanation for g

        Let A be a new TQA for the action,
          and let R be the set of new TQAs implied by the axioms for A

        Let C be the constraints between A and R

        Expand(TQAs $\cup$ {A} $\cup$ R, constraints $\cup$ C)

CBI-Planners

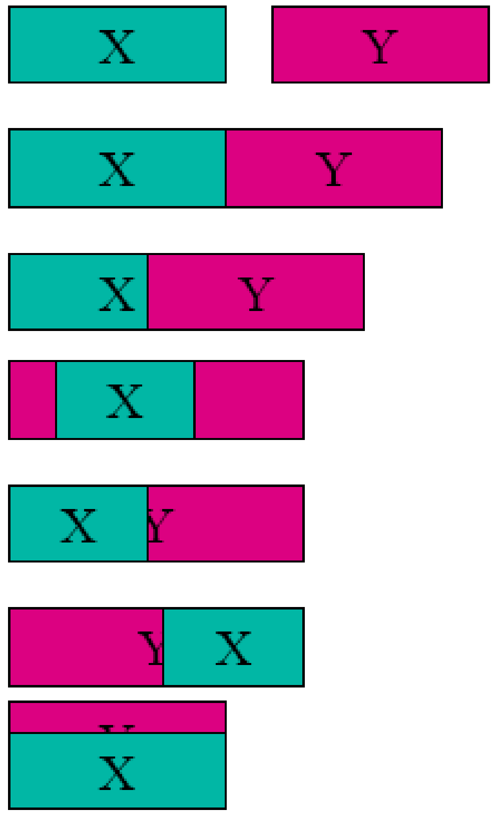
Zeno (Penberthy)	intervals, no CSP
Trains (Allen)	
Descartes (Joslin)	extreme least commitment
IxTeT (Ghallab)	functional rep.
HSTS (Muscettola)	functional rep., activities
EUROPA (Jonsson)	functional rep., activities

CBI vs POP

- CBI is similar to POP because least commitment and partial order
- But, temporal constraints in CBI …
- Contraints Temporal Network associated with a plan
- Constraint propagation

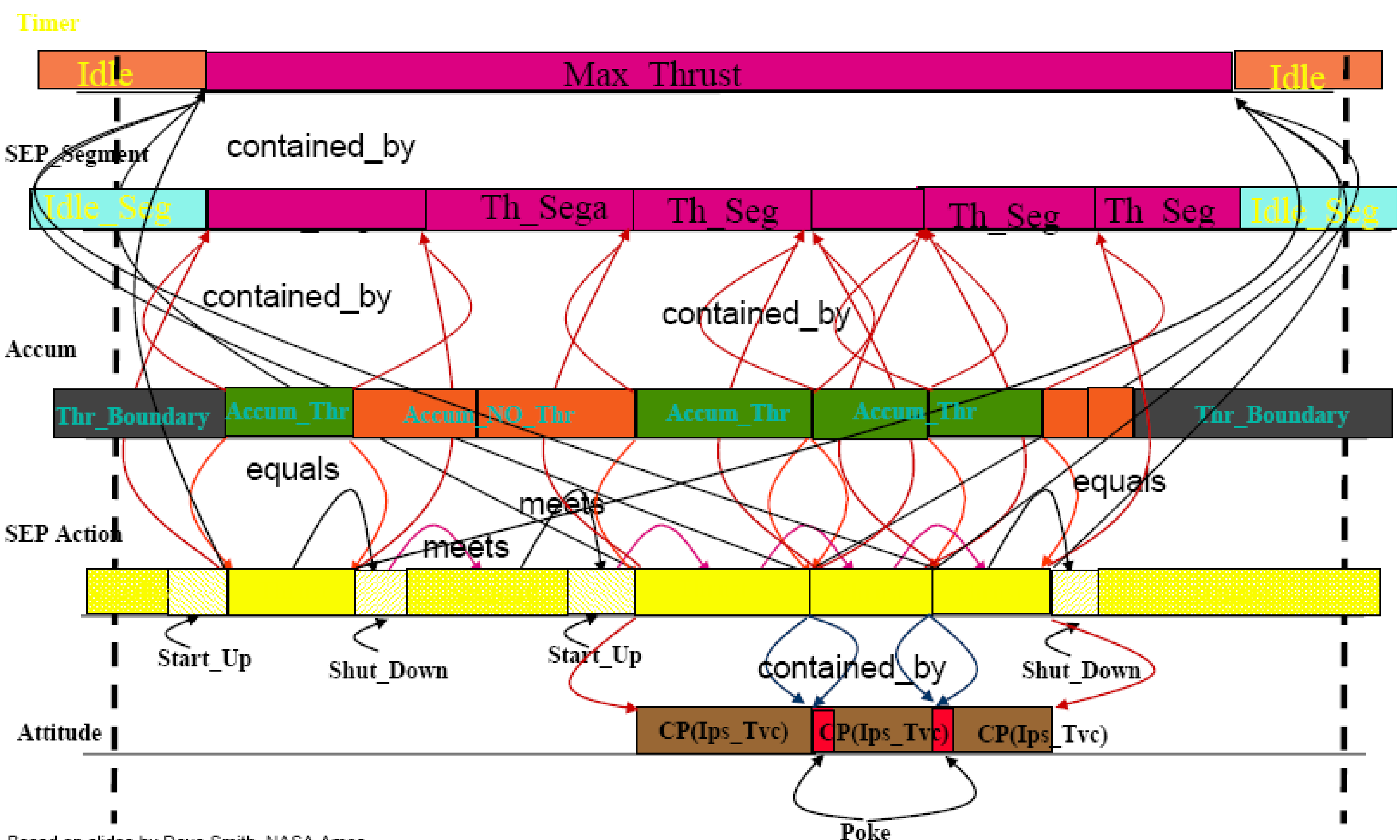
Temporal Constraints

- x before y
- x meets y
- x overlaps y
- x during y
- x starts y
- x finishes y
- x equals y



- y after x
- y met-by x
- y overlapped-by x
- y contains x
- y started-by x
- y finished-by x
- y equals x

RAX Example: DS1



Based on slides by Dave Smith, NASA Ames

Temporal Constraints as Inequalities

- x before y $X^+ < Y^-$
- x meets y $X^+ = Y^-$
- x overlaps y $(Y^- < X^+)$ & $(X^- < Y^+)$
- x during y $(Y^- < X^-)$ & $(X^+ < Y^+)$
- x starts y $(X^- = Y^-)$ & $(X^+ < Y^+)$
- x finishes y $(X^- < Y^-)$ & $(X^+ = Y^+)$
- x equals y $(X^- = Y^-)$ & $(X^+ = Y^+)$

Inequalities may be expressed as binary interval relations:

$$X^+ - Y^- < [-\mathrm{inf}, 0]$$

Metric Constraints

- Going to the store takes at least 10 minutes and at most 30 minutes.
  $\rightarrow$ $10 \leq [T^{+}(\text{store}) - T^{-}(\text{store})] \leq 30$

- Bread should be eaten within a day of baking.
  $\rightarrow$ $0 \leq [T^{+}(\text{baking}) - T^{-}(\text{eating})] \leq 1$ day

- Inequalities, $X^{+} < Y^{-}$, may be expressed as binary interval relations:
  $\rightarrow$ $-\inf < [X^{+} - Y^{-}] < 0$

Temporal Constraint Networks

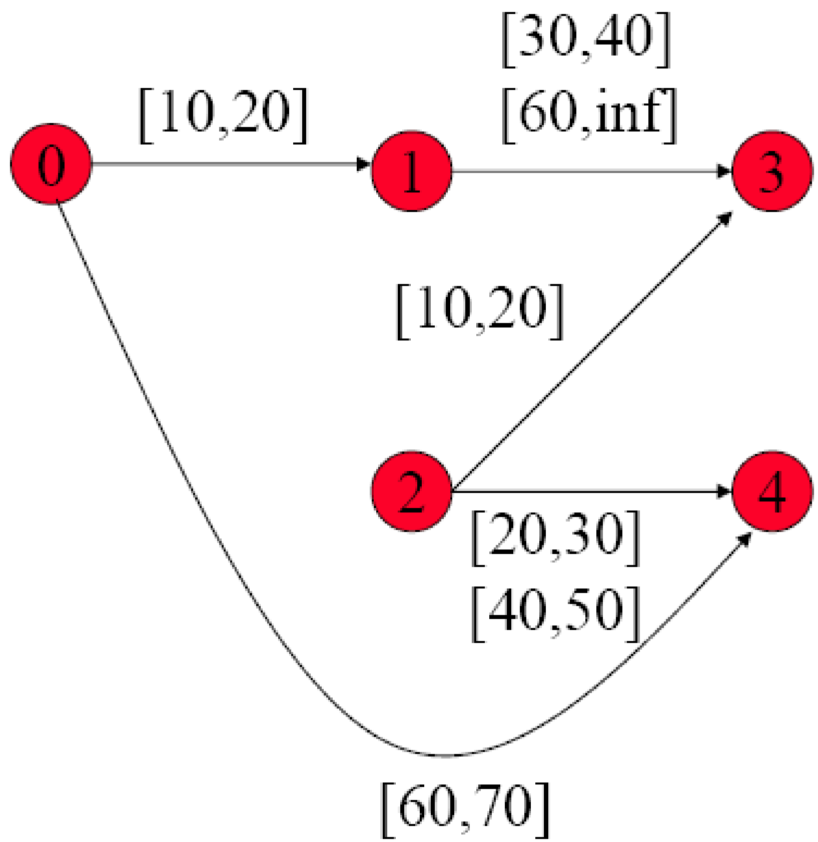
- A set of time points X_i at which events occur.
- Unary constraints

$$(a_0 \leq X_i \leq b_0) \text{ or } (a_1 \leq X_i \leq b_1) \text{ or } \ldots$$

- Binary constraints

$$(a_0 \leq X_j - X_i \leq b_0) \text{ or } (a_1 \leq X_j - X_i \leq b_1) \text{ or } \ldots$$

Temporal Constraint Satisfaction Problem



Simple Temporal Networks

Simple Temporal Networks:

- A set of time points X_i at which events occur.
- Unary constraints

  $(a_0 \le X_i \le b_0)$ ~~or $(a_1 \le X_i \le b_1)$ or . . .~~
- Binary constraints

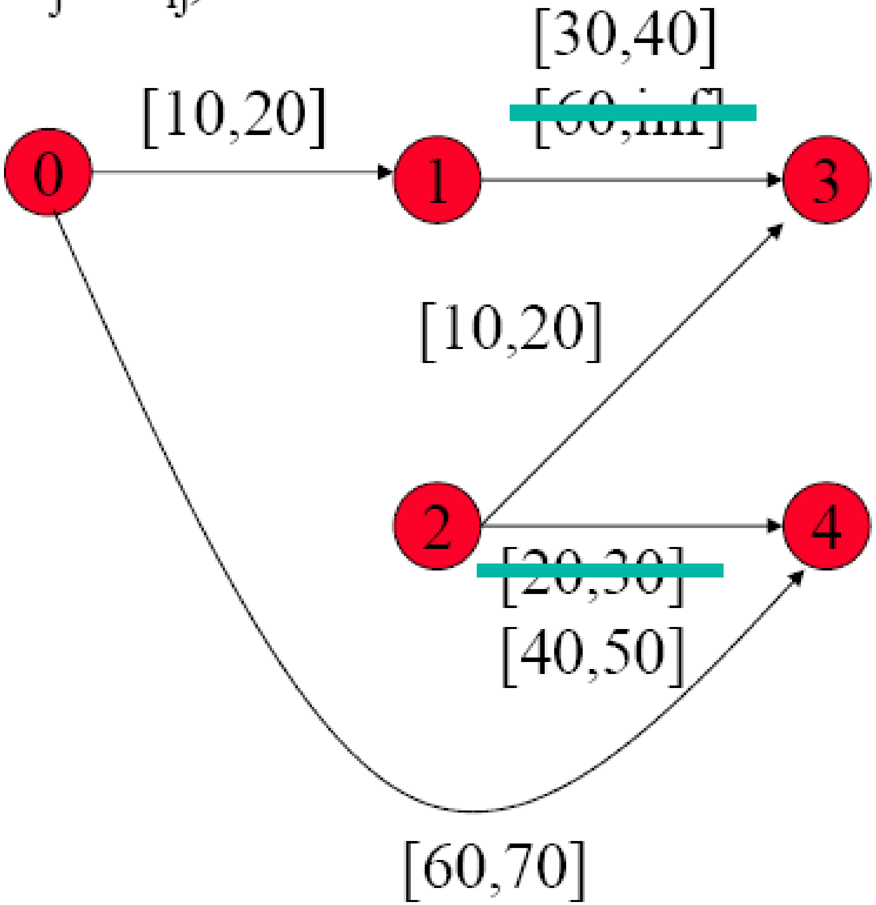
  $(a_0 \le X_j - X_i \le b_0)$ ~~or $(a_1 \le X_j - X_i \le b_1)$ or . . .~~

Sufficient to represent:
- most Allen relations
- simple metric constraints

Can't represent:
- Disjoint activities

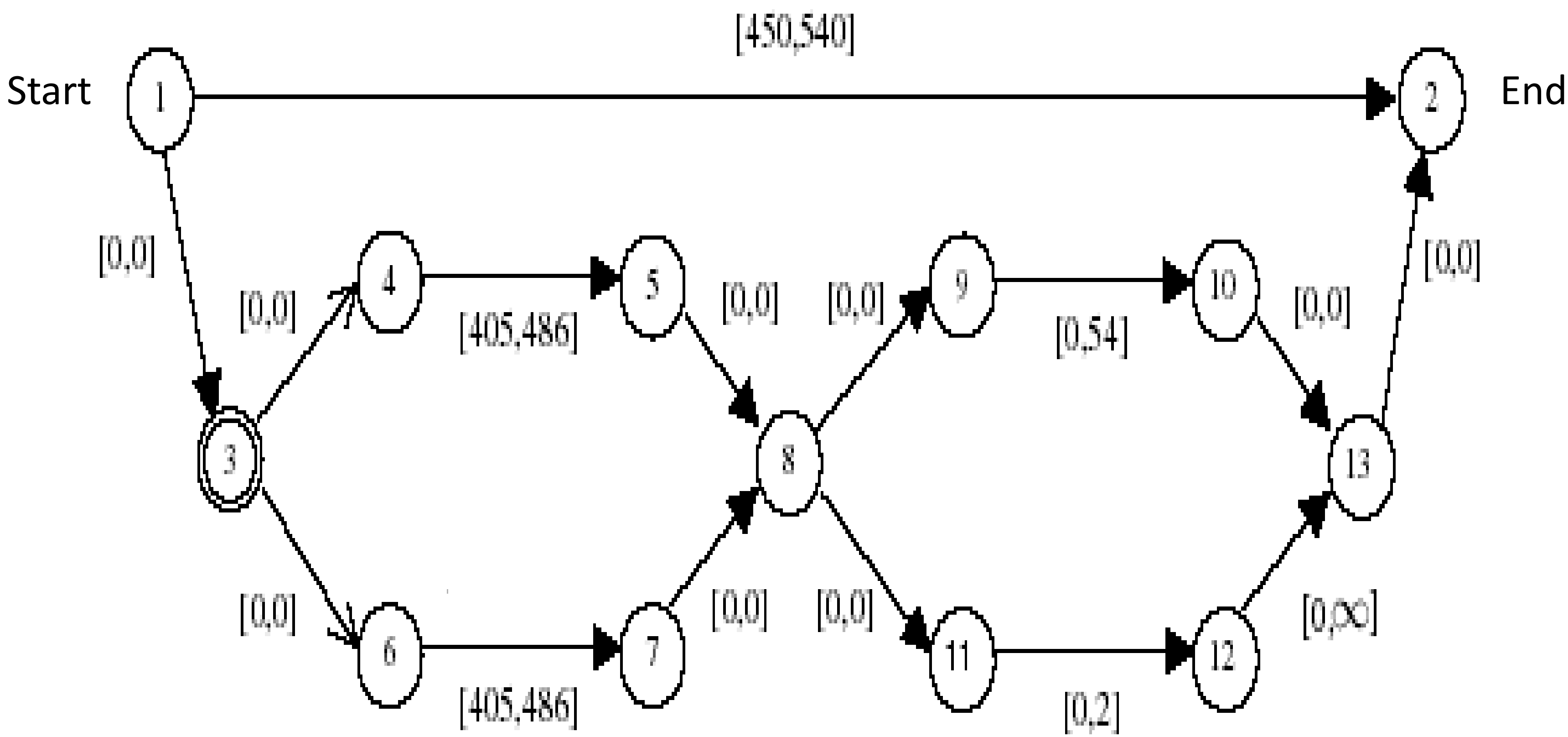
Simple Temporal Networks

- $T_{ij} = (a_{ij} \leq X_i - X_j \leq b_{ij})$

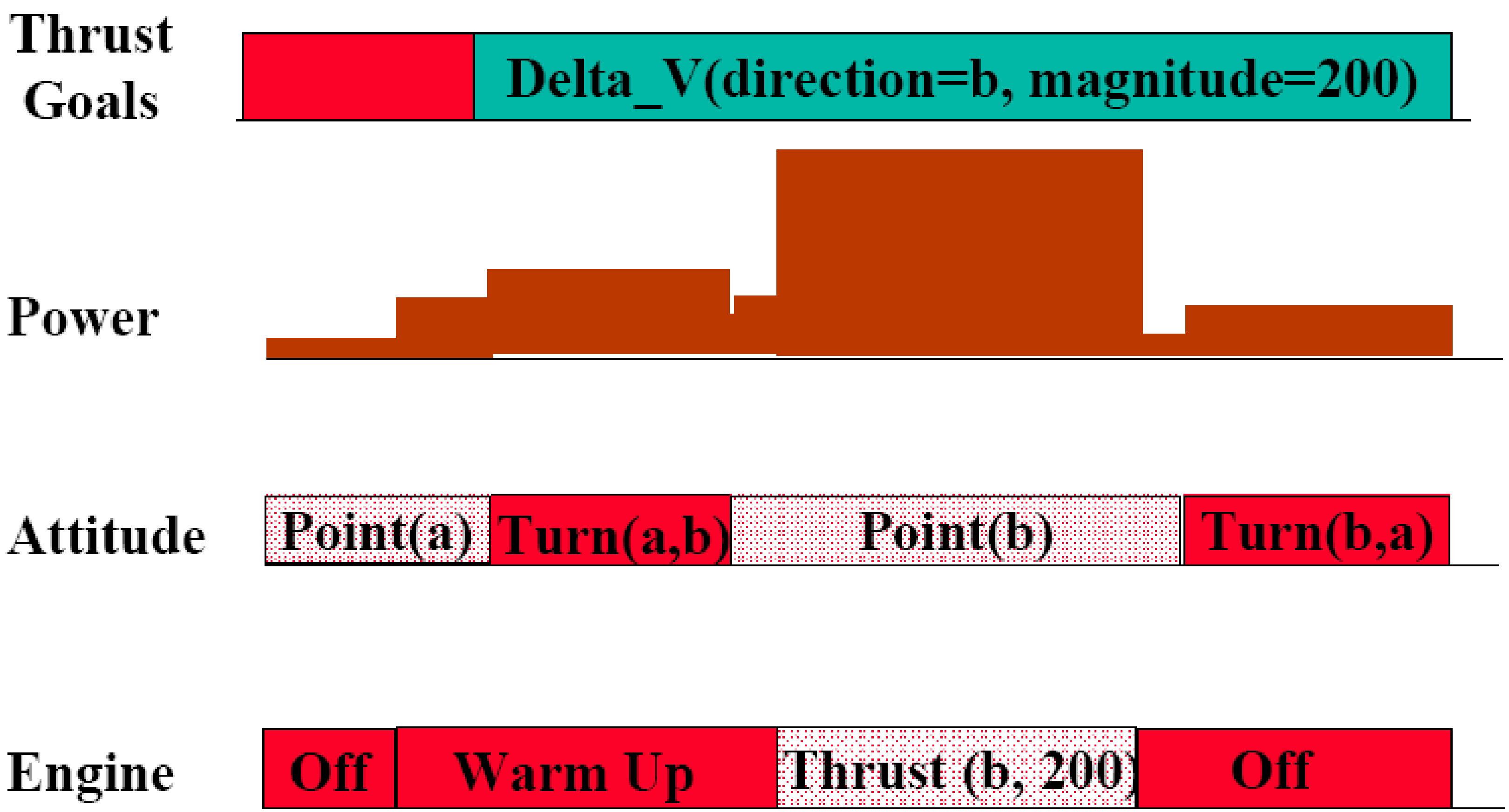


Based on slides by Dave Smith, NASA Ames

STN example

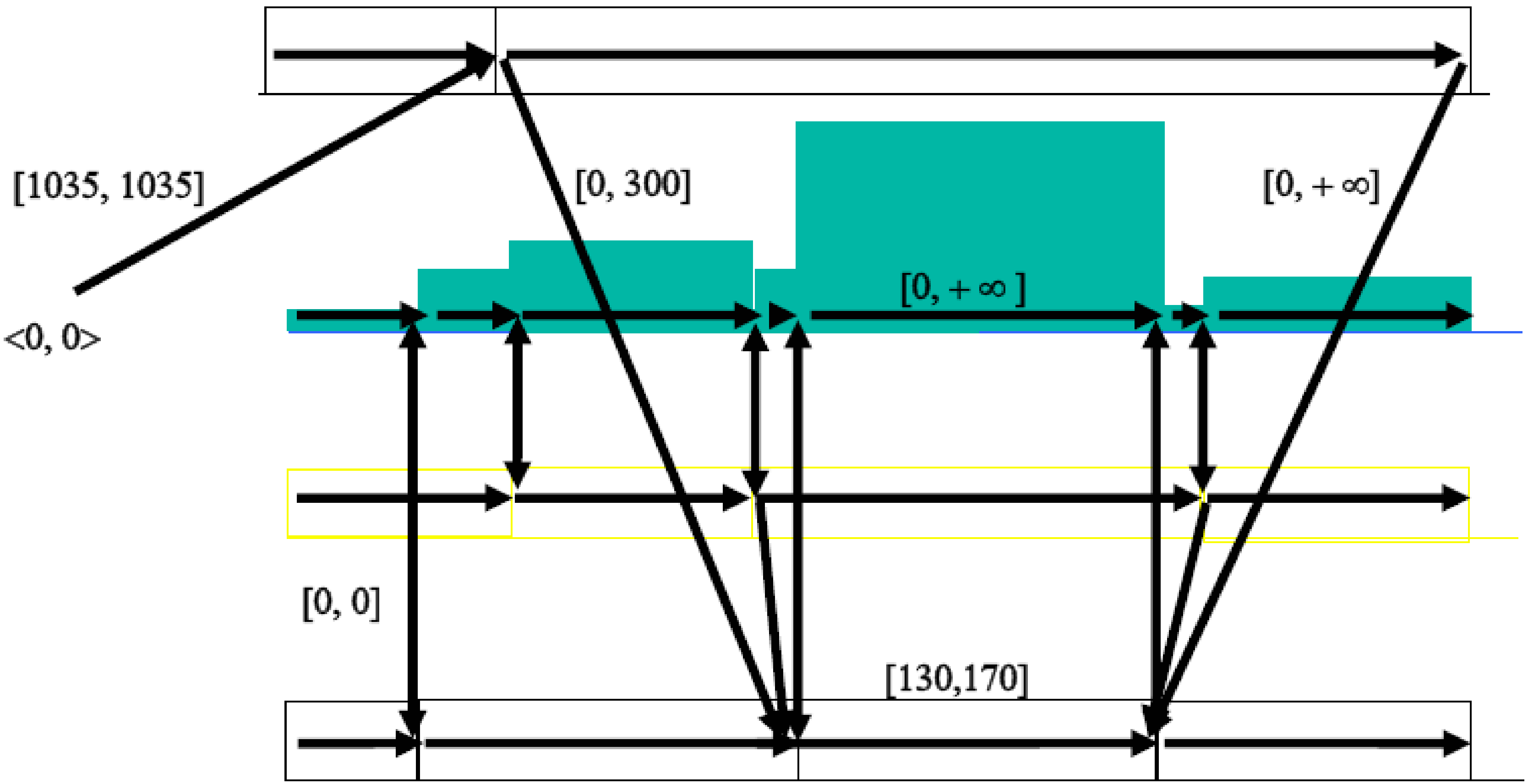

Start
End
[450,540]
1
2
3
4
5
6
7
8
9
10
11
12
13
[0,0]
[0,0]
[405,486]
[0,0]
[0,0]
[0,54]
[0,0]
[0,0]
[0,0]
[405,486]
[0,0]
[0,0]
[0,2]
[0,∞]

A Complete CBI-Plan is a STN



Based on slides by Dave Smith, NASA Ames

A Complete CBI-Plan is a STN



Based on slides by Dave Smith, NASA Ames

DS1: Remote Agent

Remote Agent Experiment: RAX

Remote Agent Experiment

See rax.arc.nasa.gov

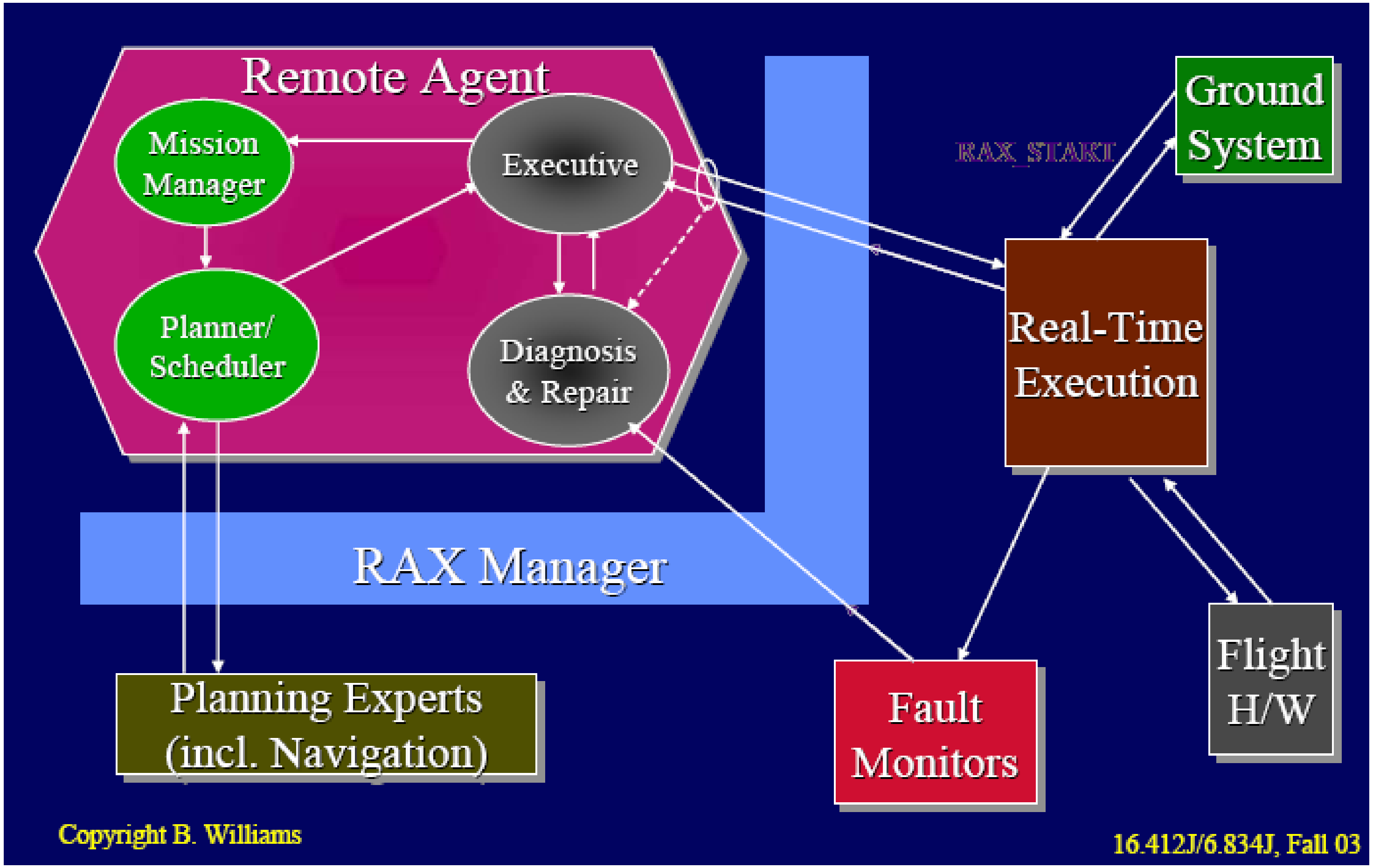
May 17-18th experiment

- Generate plan for course correction and thrust
- Diagnose camera as stuck on
  - Power constraints violated, abort current plan and replan
- Perform optical navigation
- Perform ion propulsion thrust

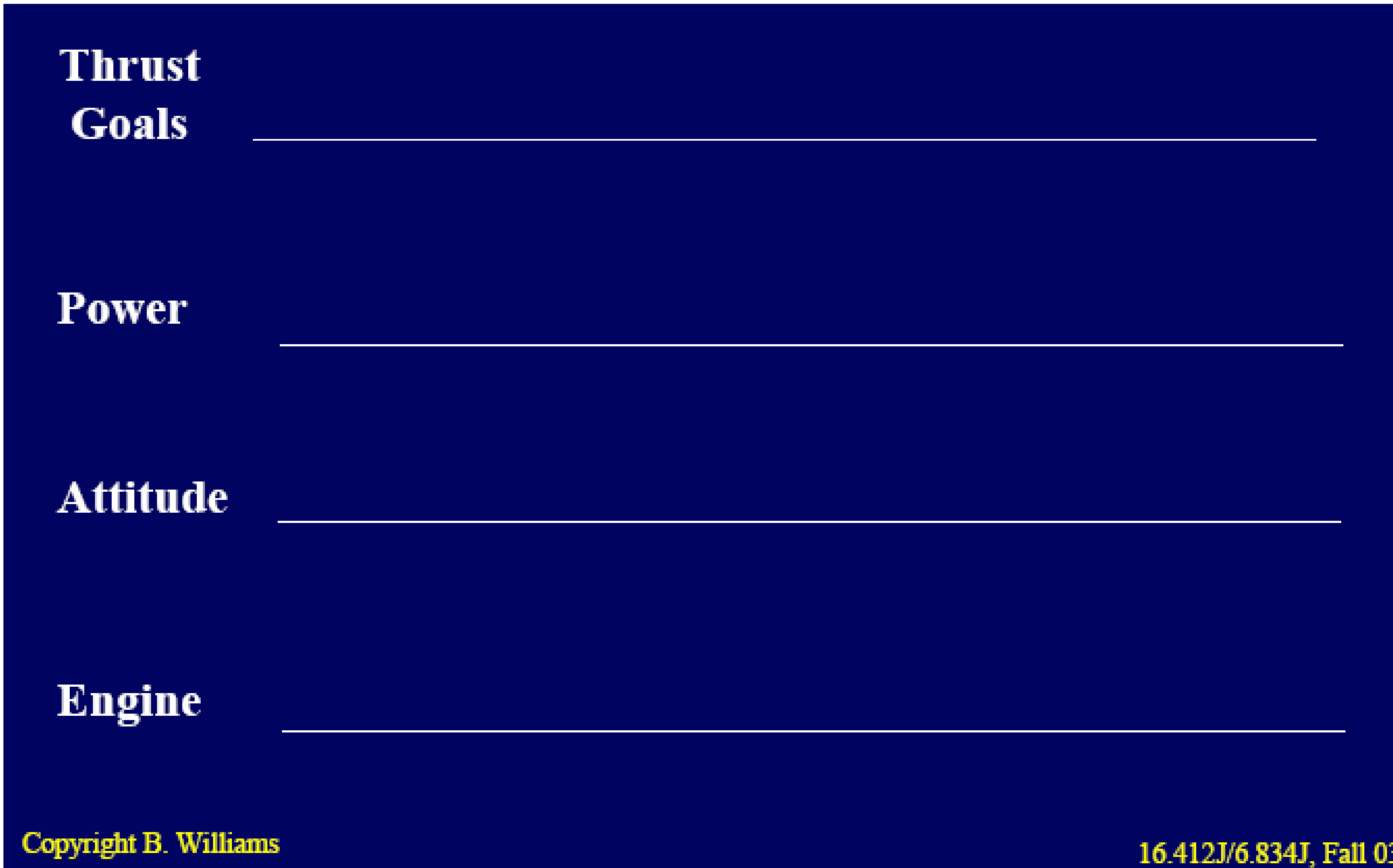
May 21th experiment.

- Diagnose faulty device and
  - Repair by issuing reset.
- Diagnose switch sensor failure.
  - Determine harmless, and continue plan.
- Diagnose thruster stuck closed and
  - Repair by switching to alternate method of thrusting.
- Back to back planning

Remote Agent

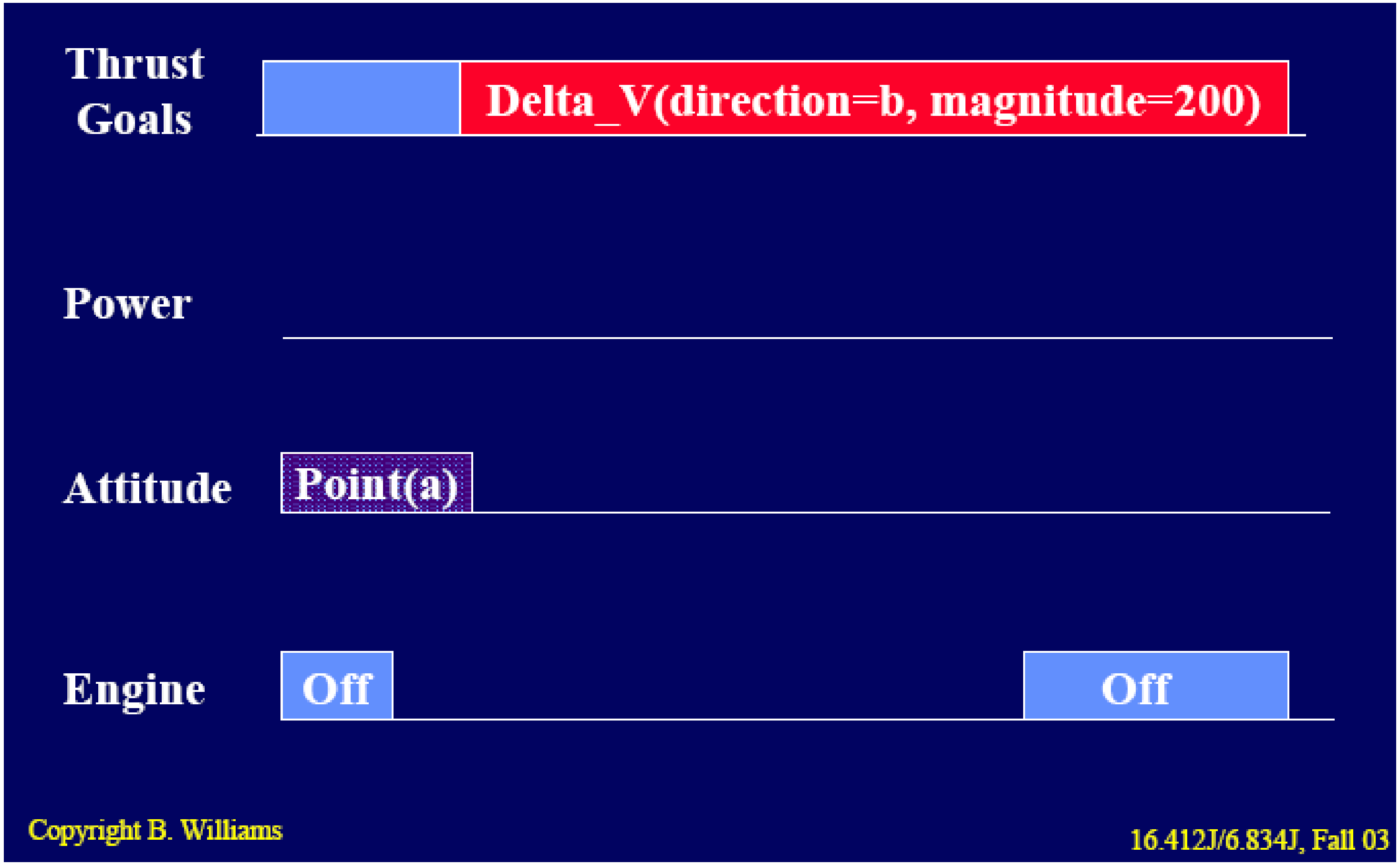


Remote Agent



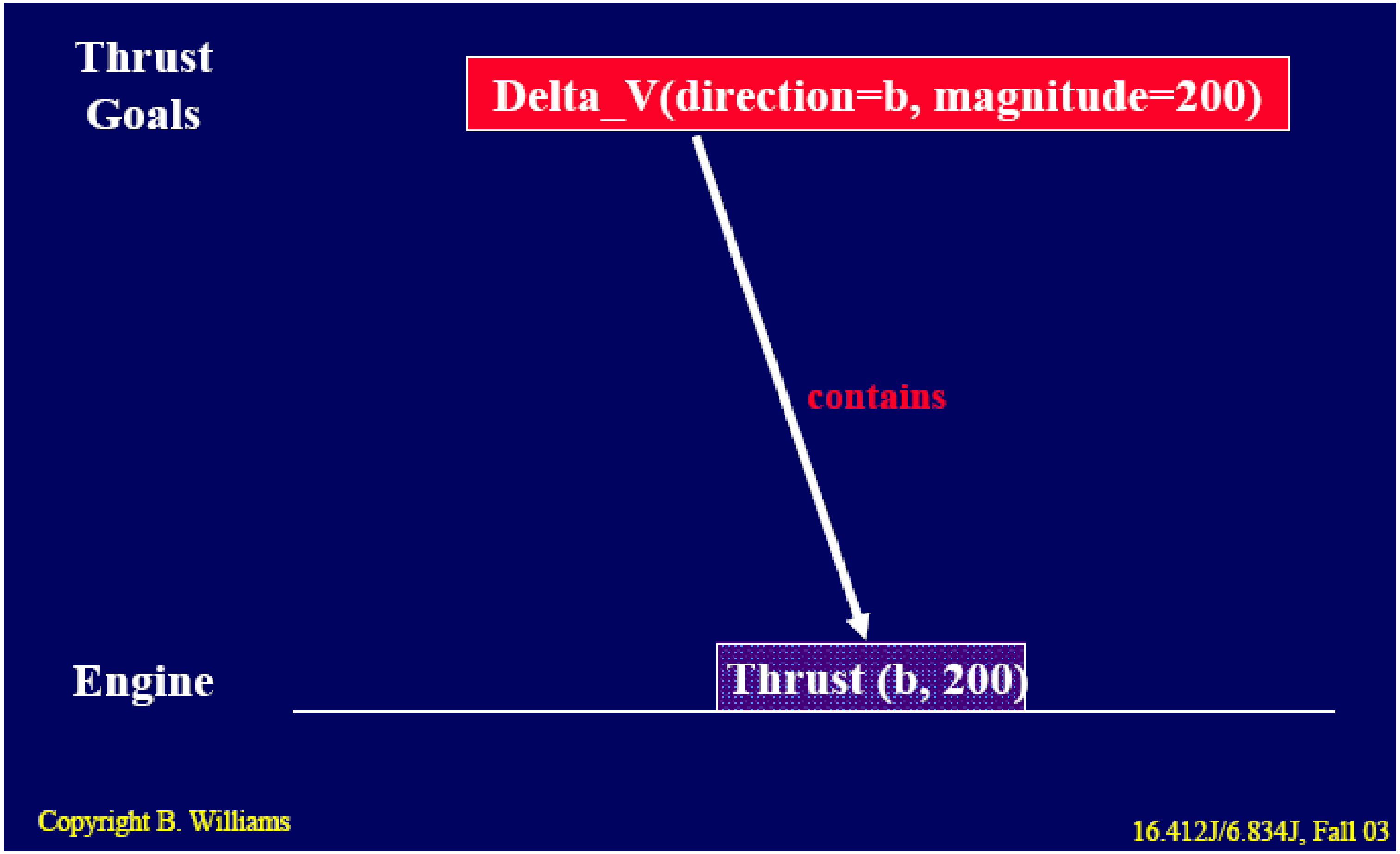
Remote Agent

- Mission Manager



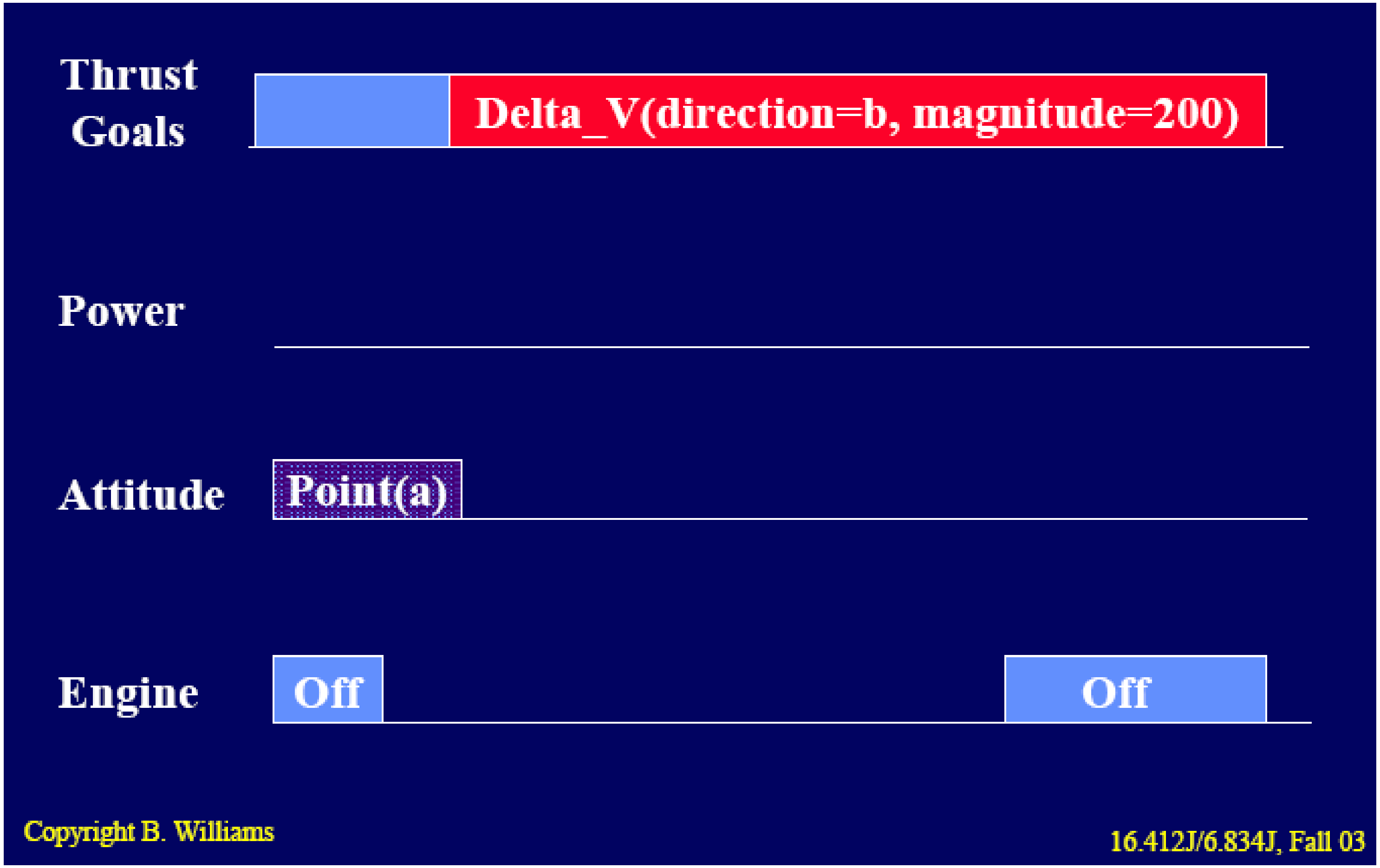
Remote Agent

- Constraints:



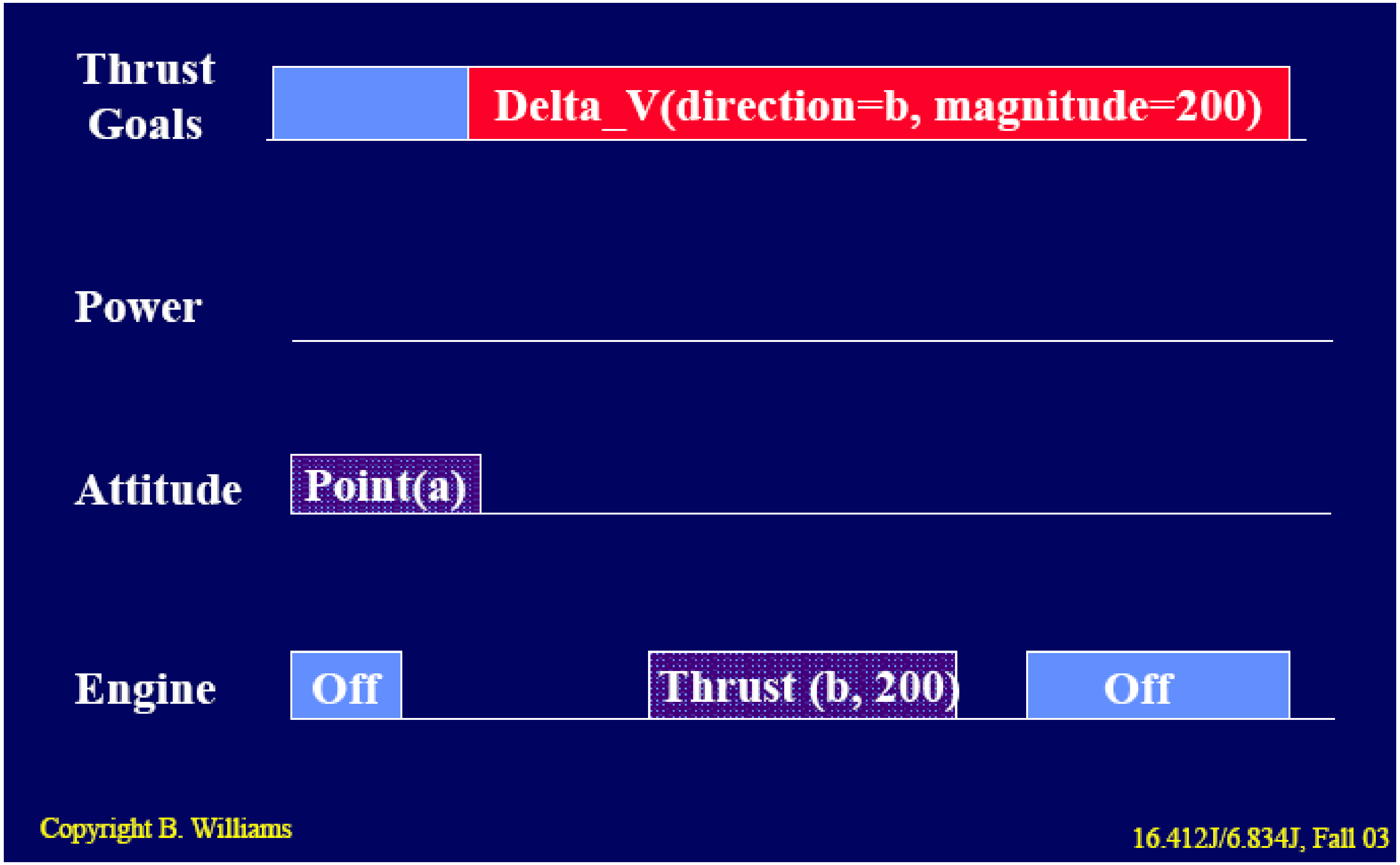
Remote Agent

- Planner starts



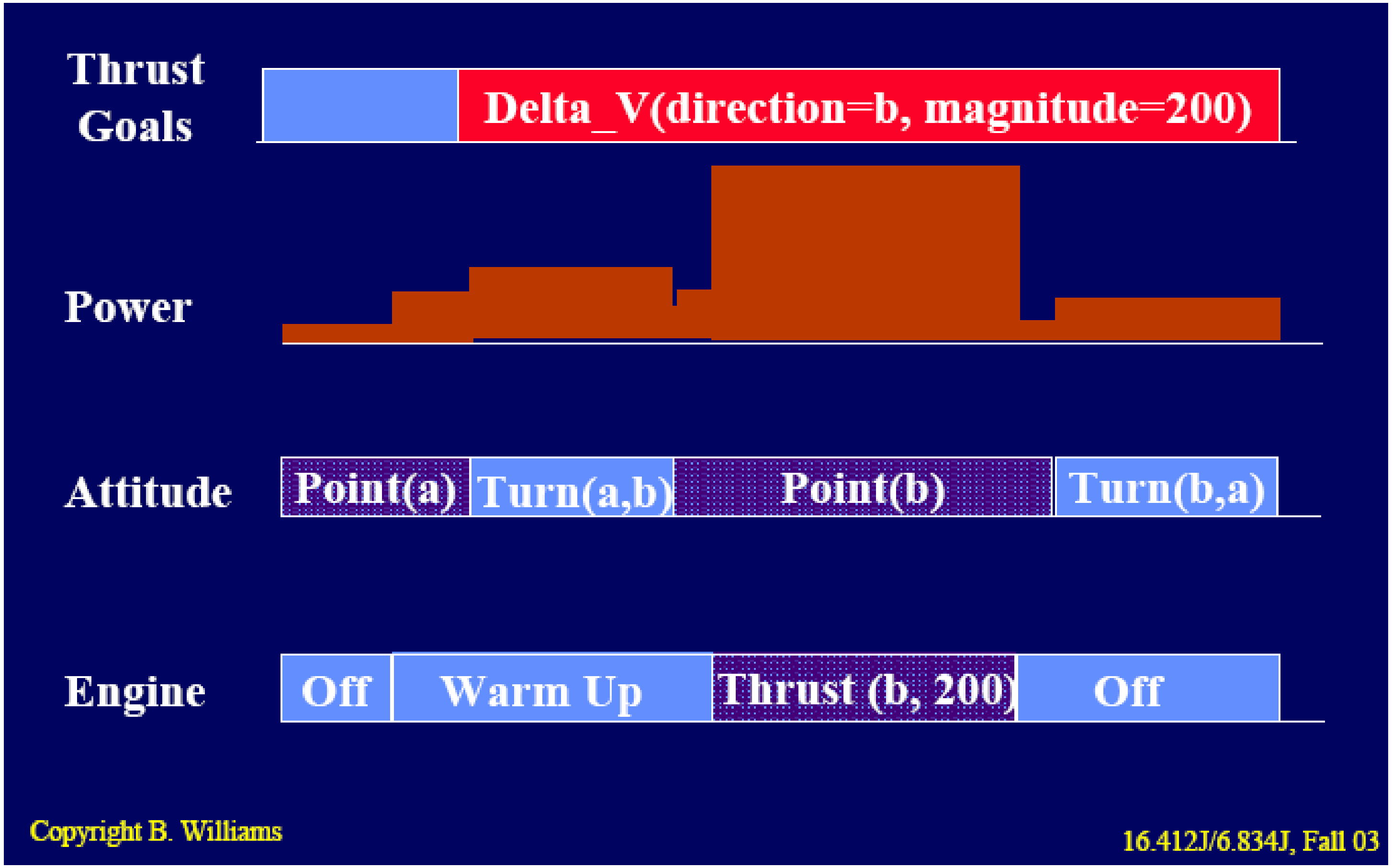
Remote Agent

- Planning



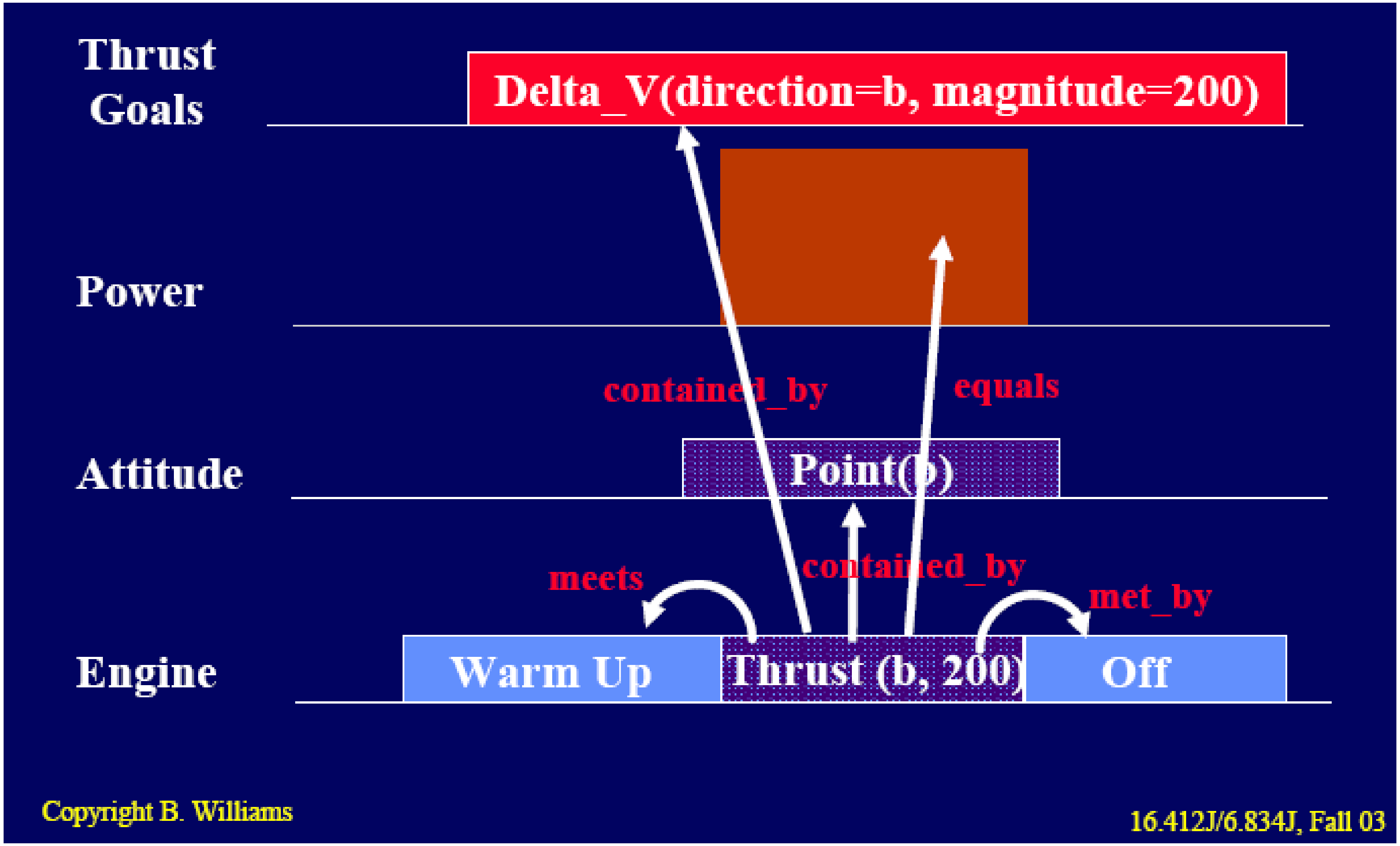
Remote Agent

- Final Plan



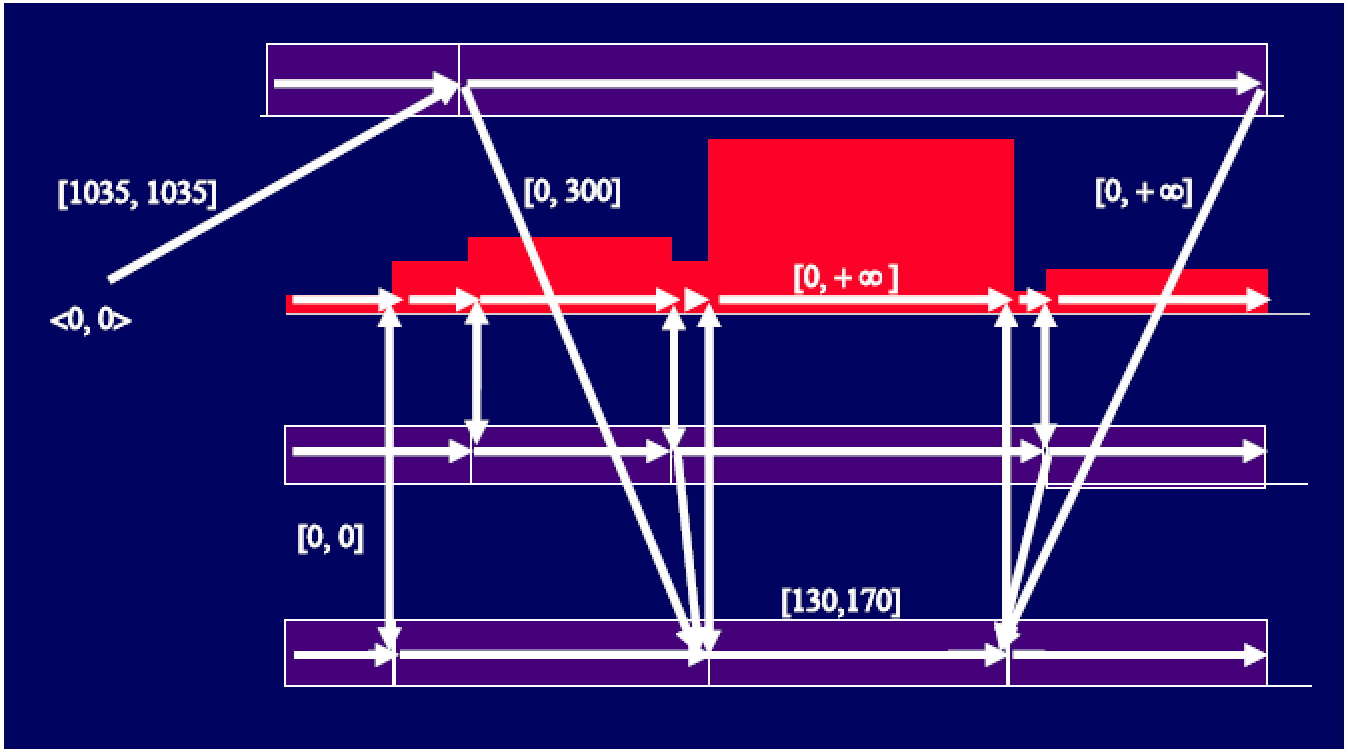
Remote Agent

- Constraints



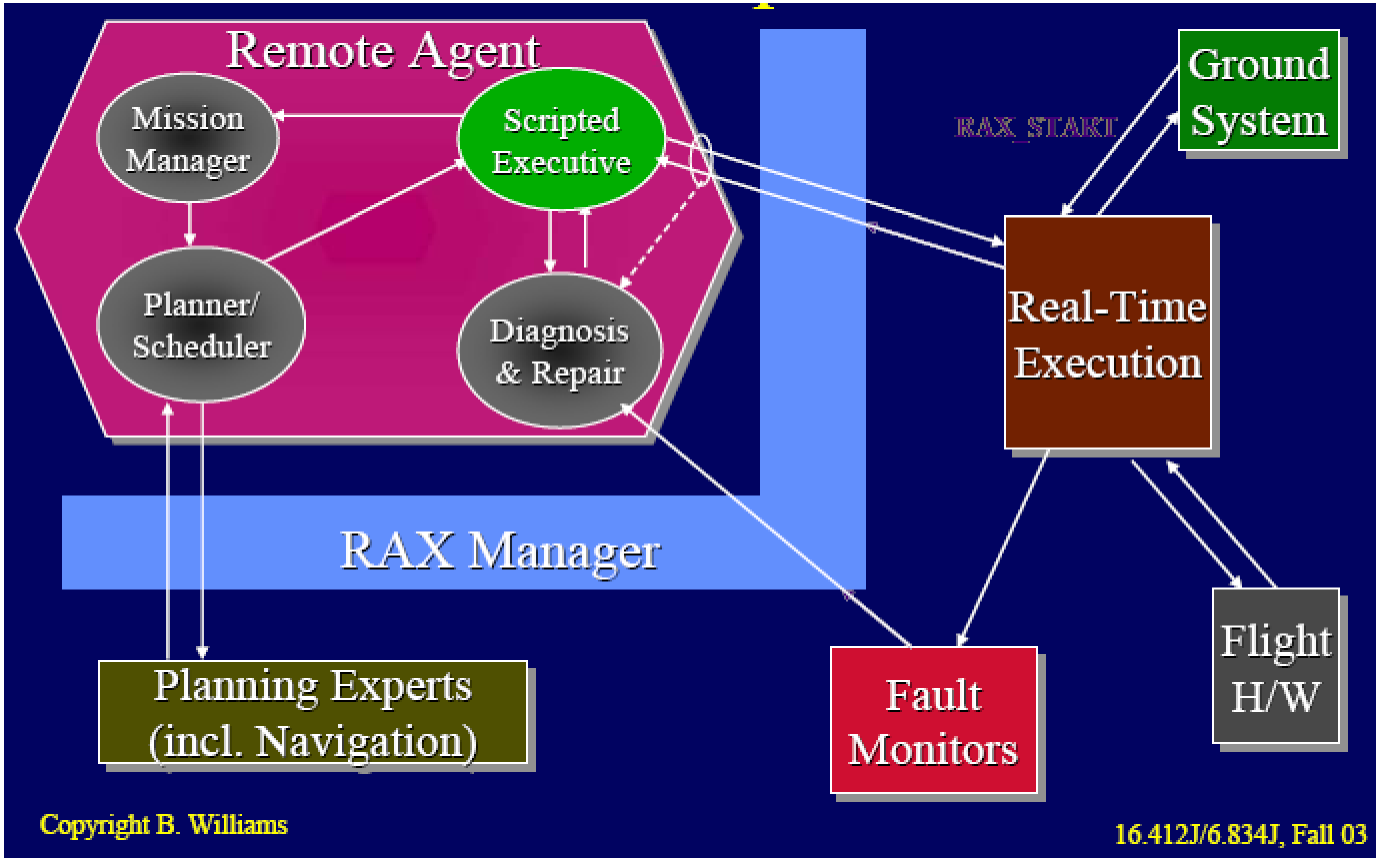
Remote Agent

- Flexible Temporal Plan through least commitment



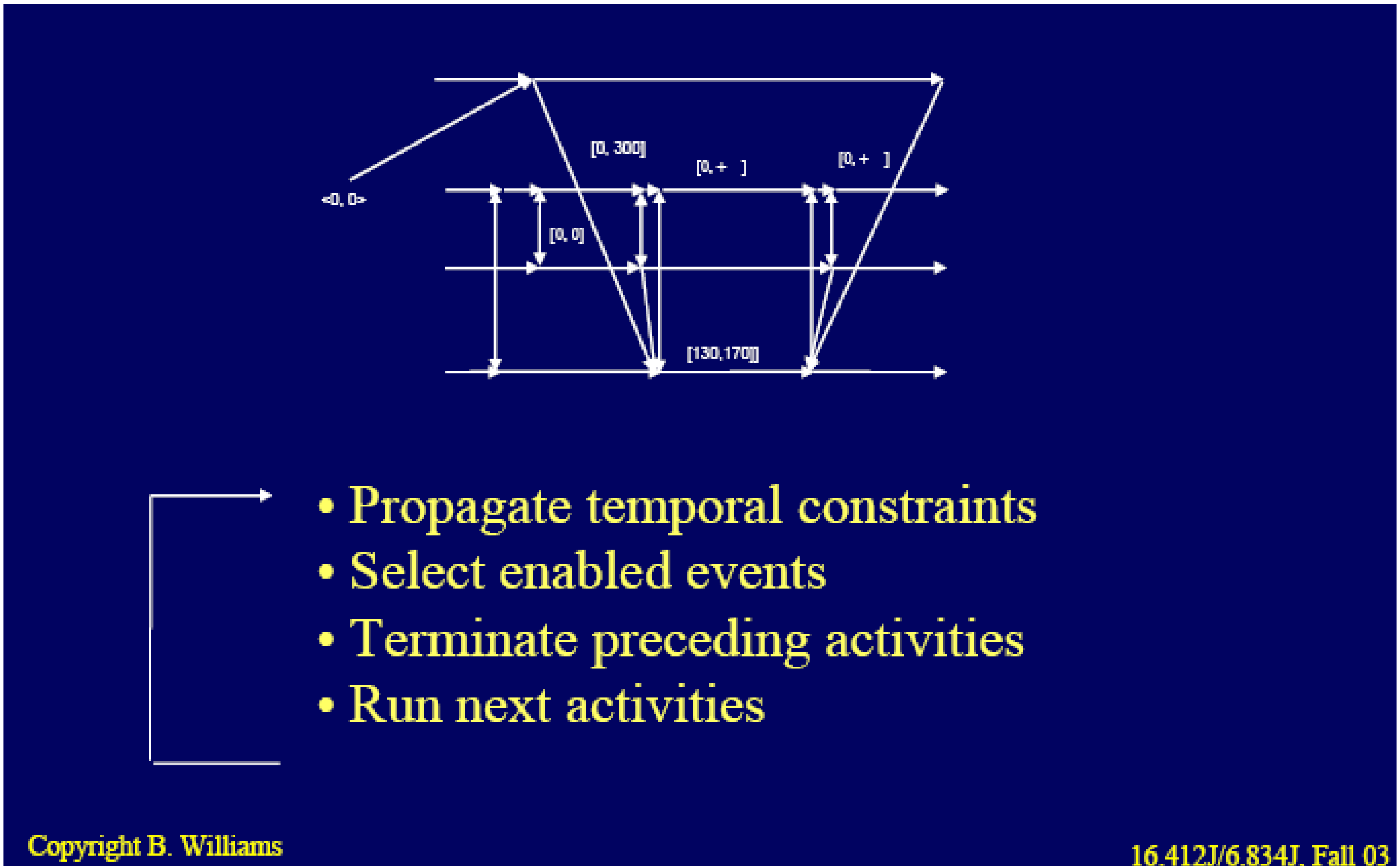
Remote Agent

- Executive system dispatch tasks



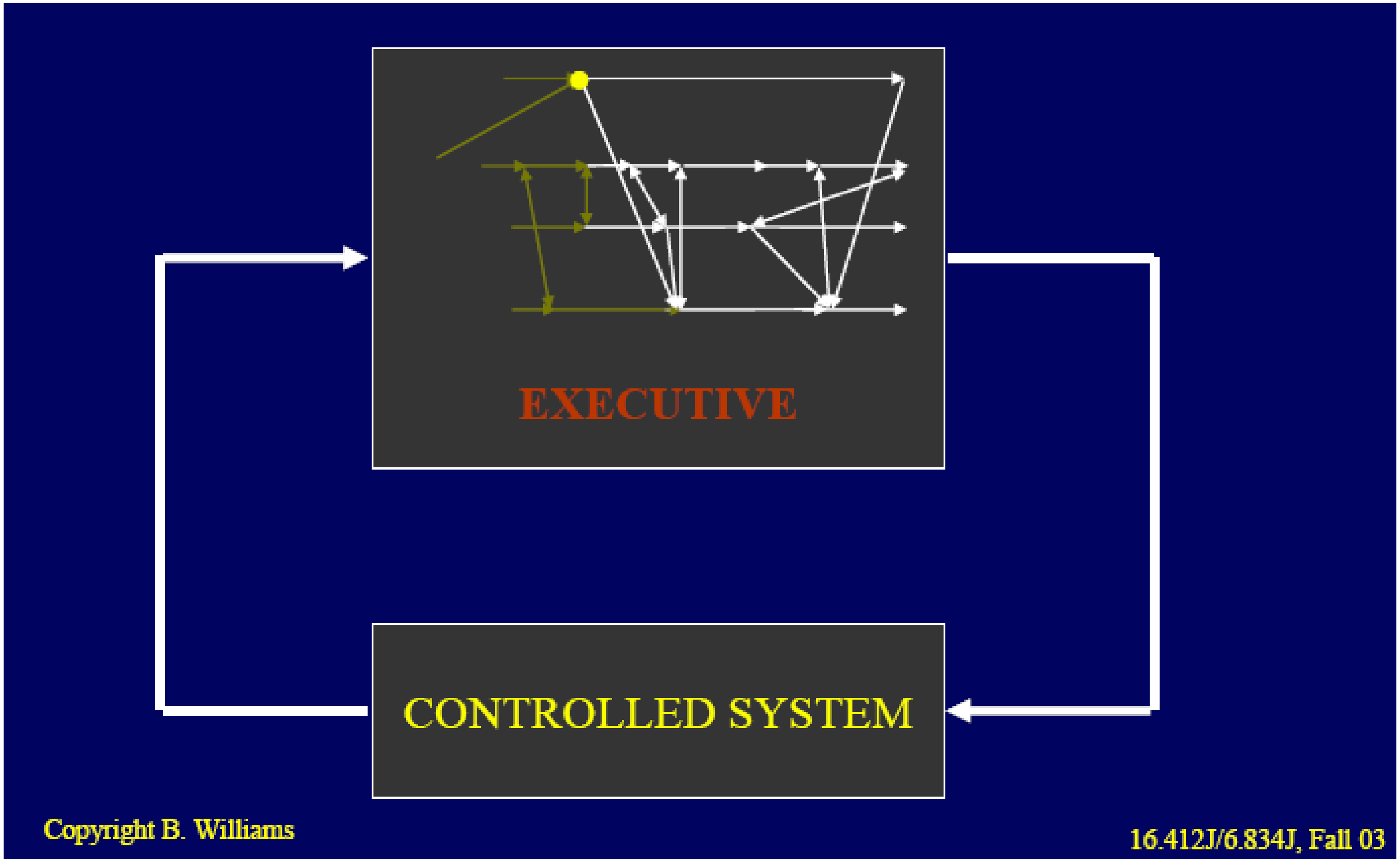
Remote Agent

- Executing Flexible Plans



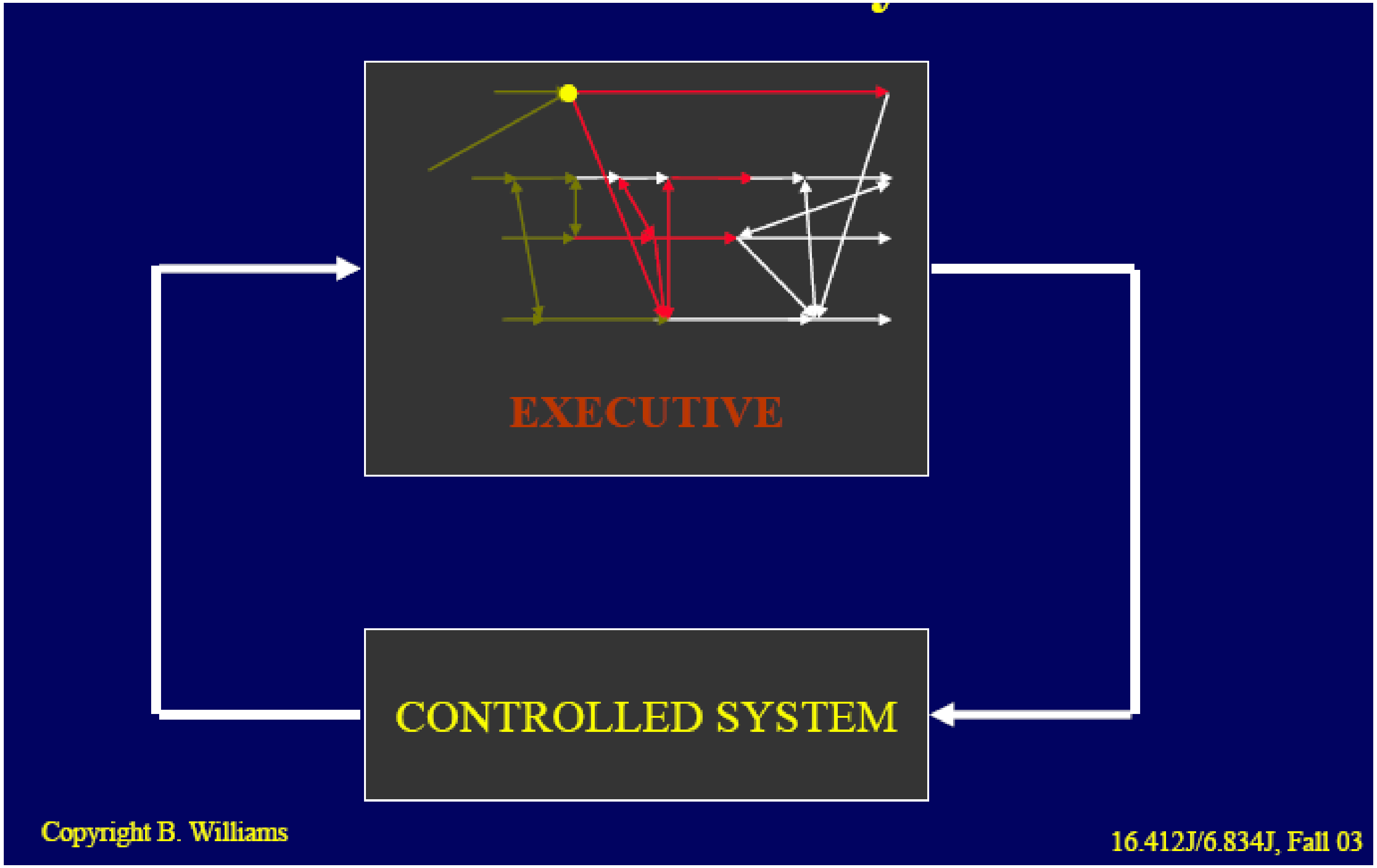
Remote Agent

- Constraint propagation can be costly



Remote Agent

- Constraint Propagation can be costly



Remote Agent

- Solution: compile temporal constraints to an efficient network