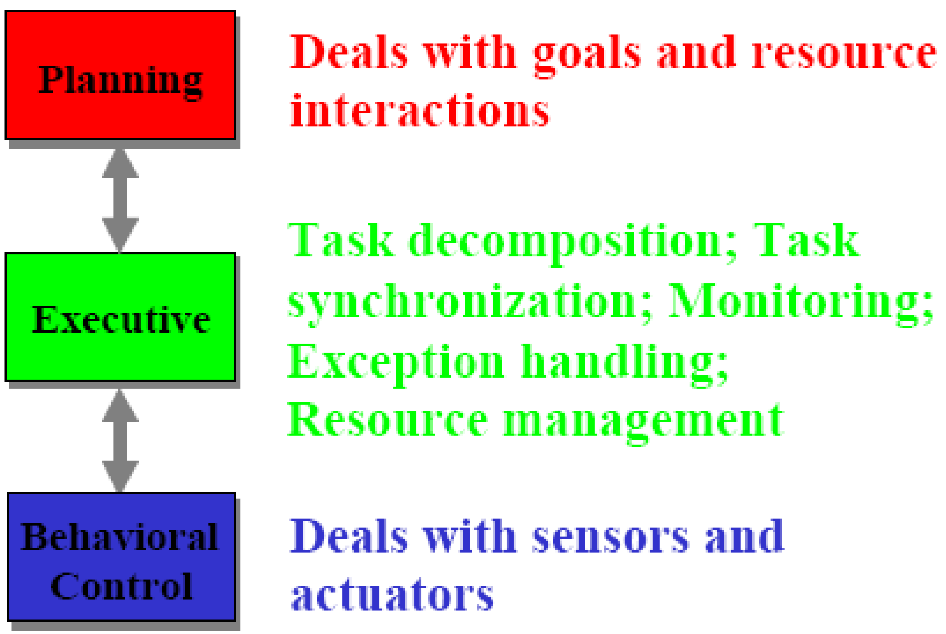
Task Planning

Architetture Robotiche

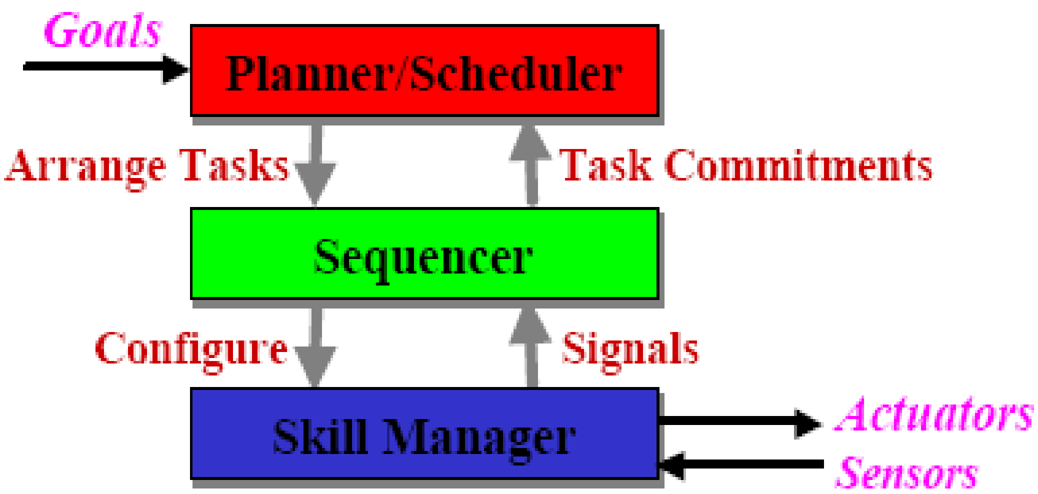
Architetture a 3 Livelli

- 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

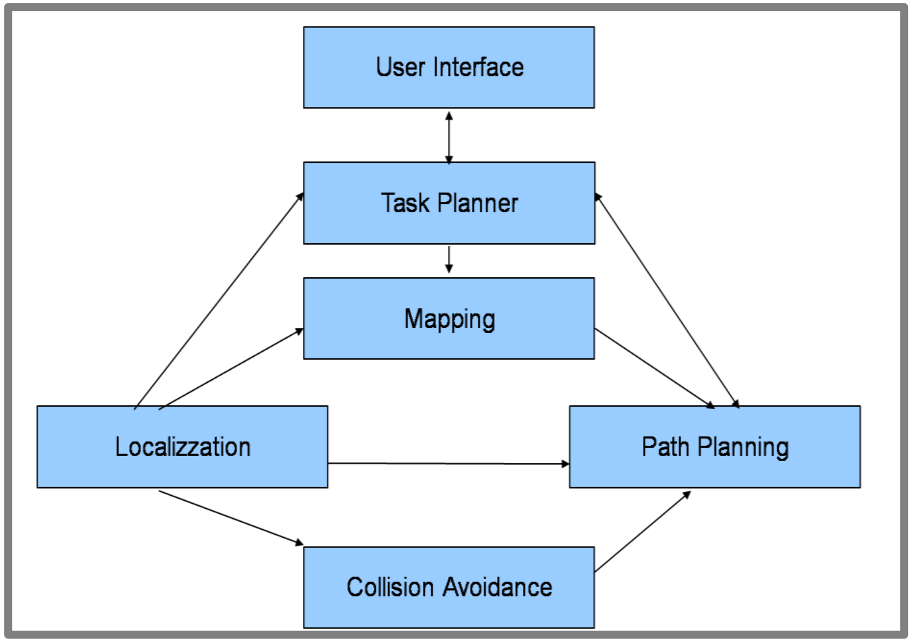


Esempio: RHINO Architettura

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

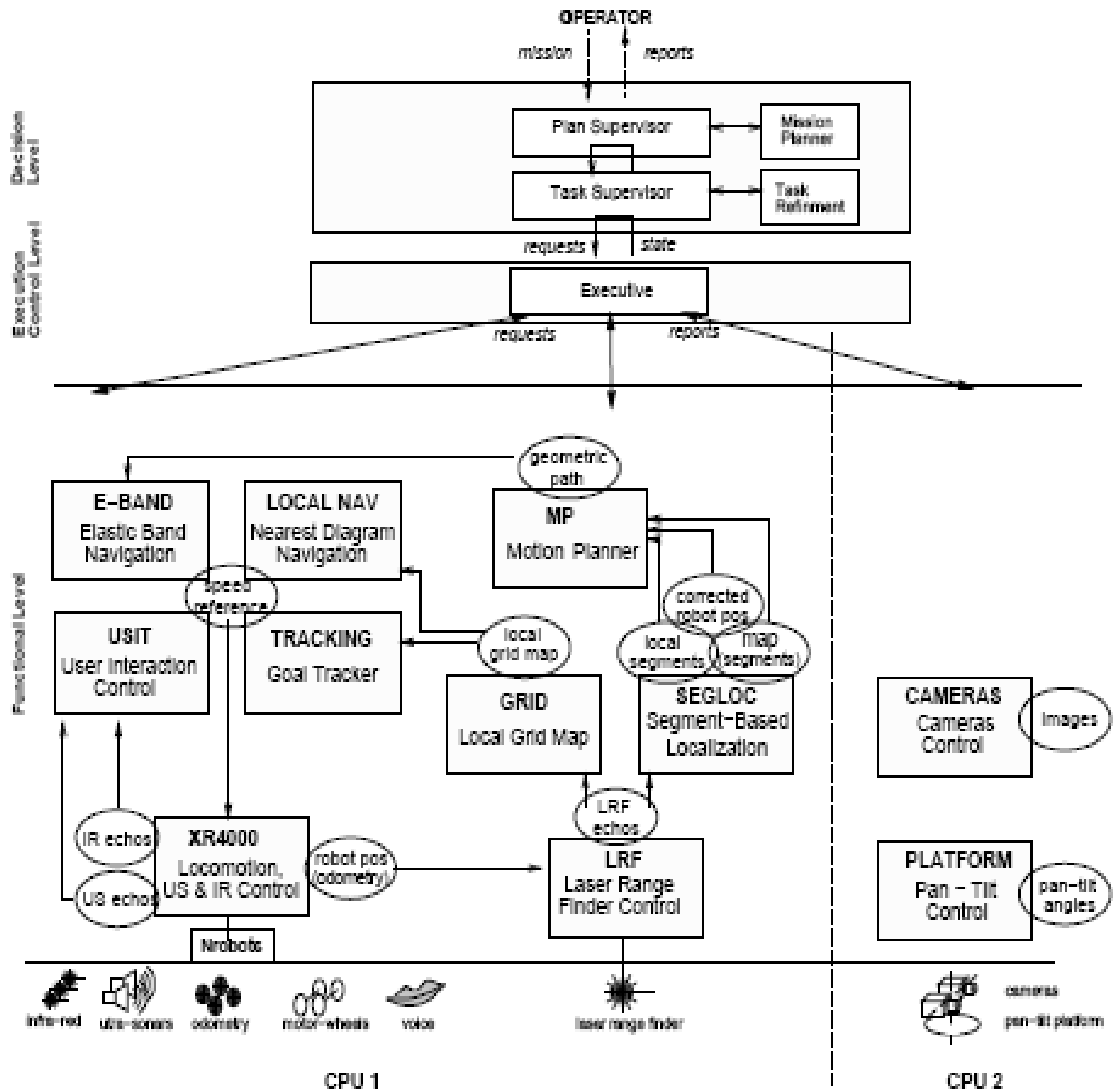
Architetture a 3 Livelli

- LAAS architecture:

Tre Livelli:

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

Controllo di Rover



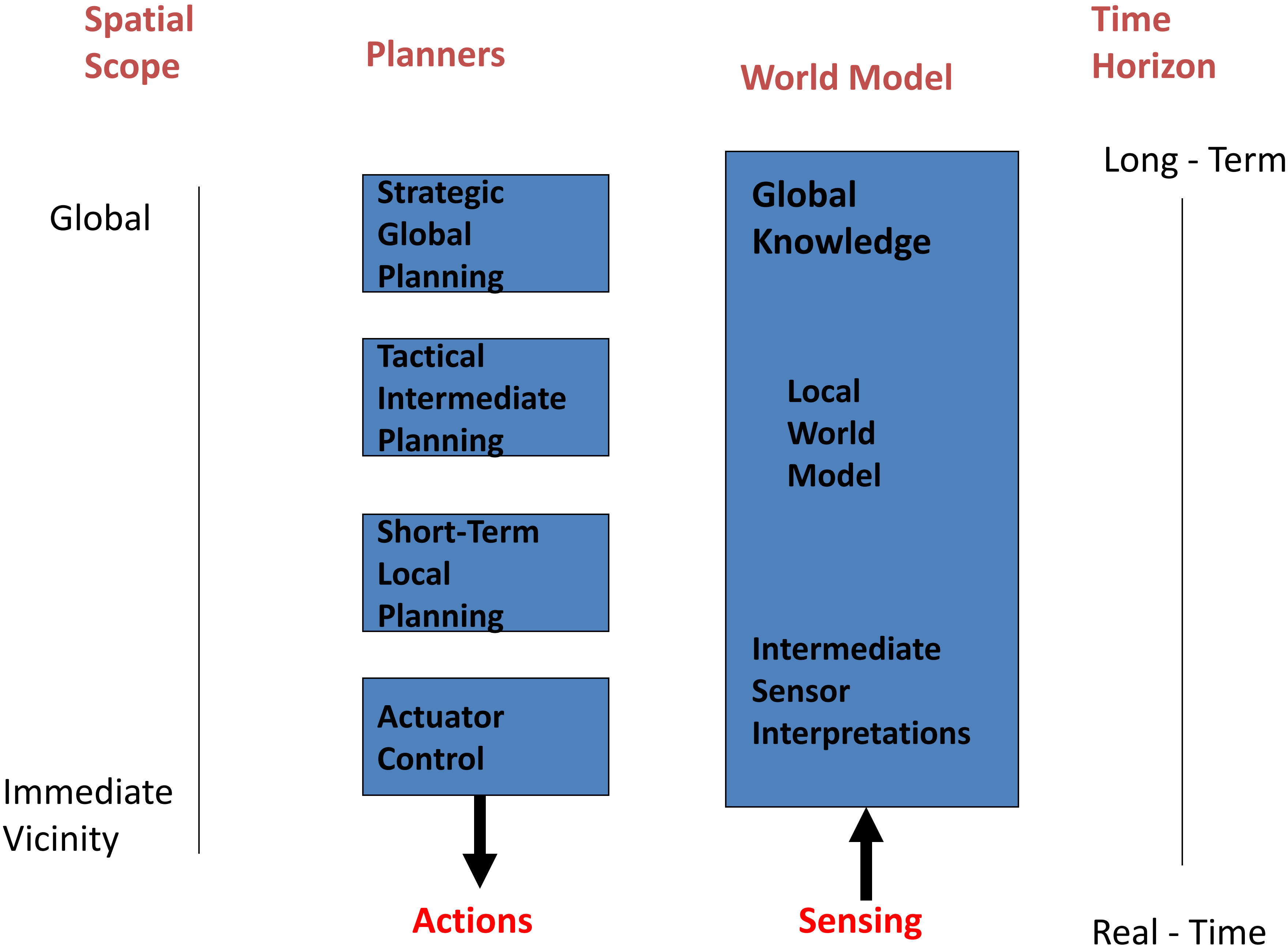
Architetture a 3 Livelli

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

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.

Plan-based vs. BB System

Plan-base control

- 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.

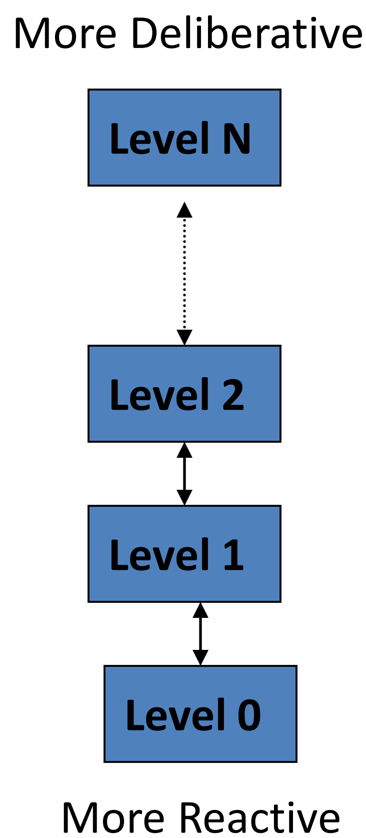
Hybrid Systems

Planning and reaction can be tied:

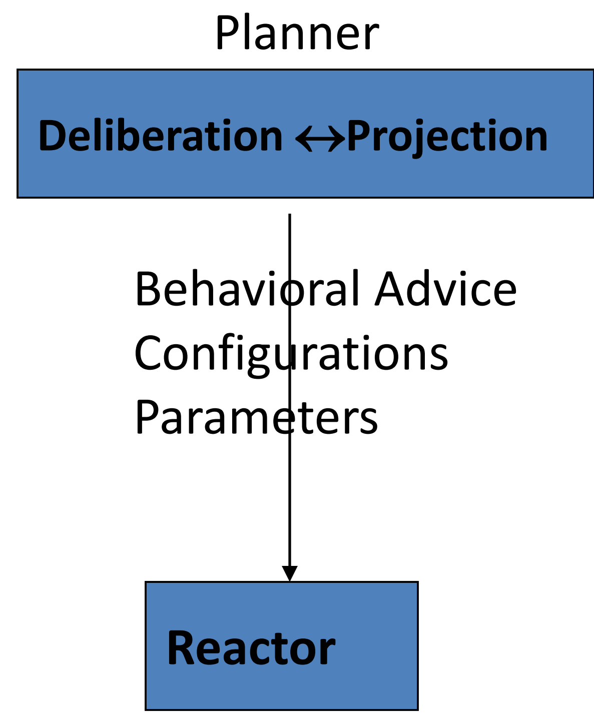
A: hierarchical integration - planning and reaction are involved with different activities, time scales

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

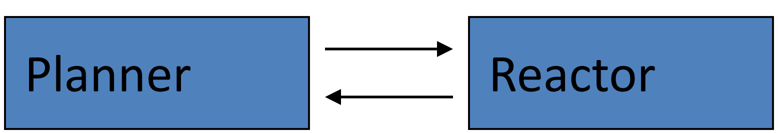
C: coupled - concurrent activities



A



B



C

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.**

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.

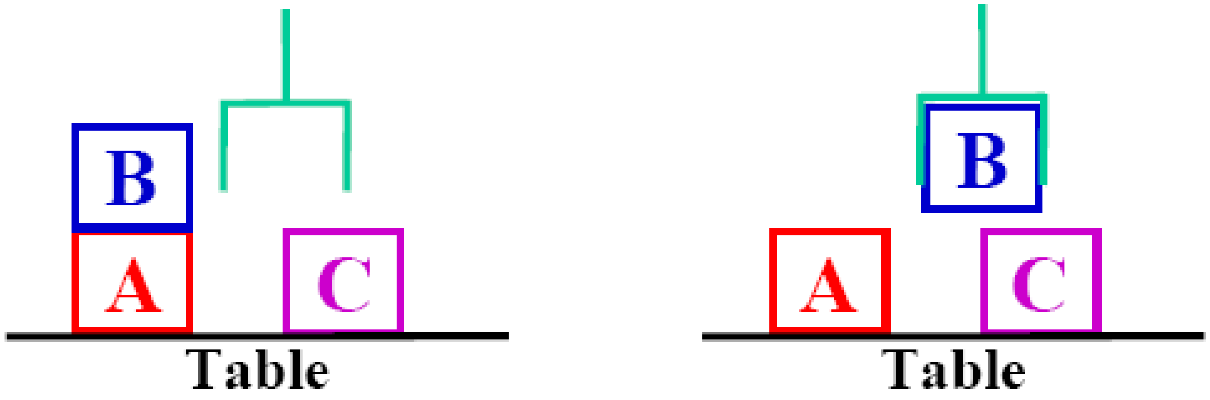
Action Model, State, Goals

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
- …

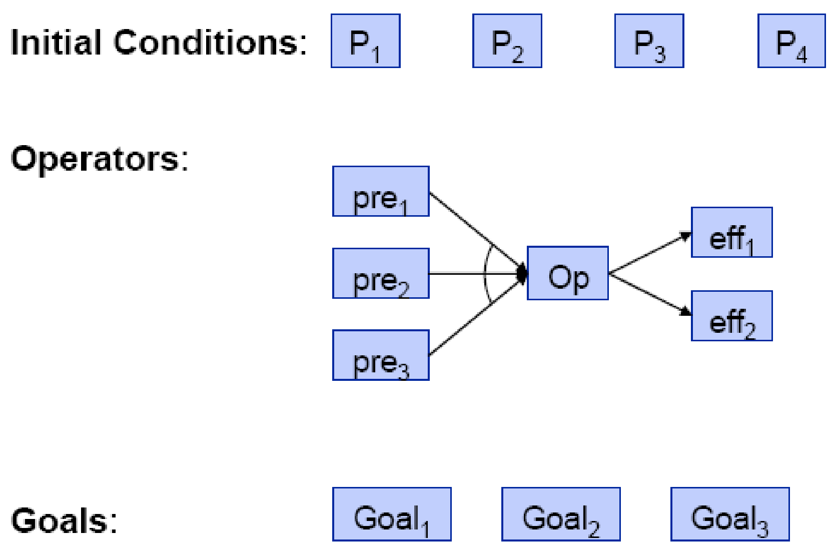
TakeImage (?target, ?instr):
- Pre: Status(?instr, Calibrated), Pointing(?target)
- Eff: Image(?target)

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

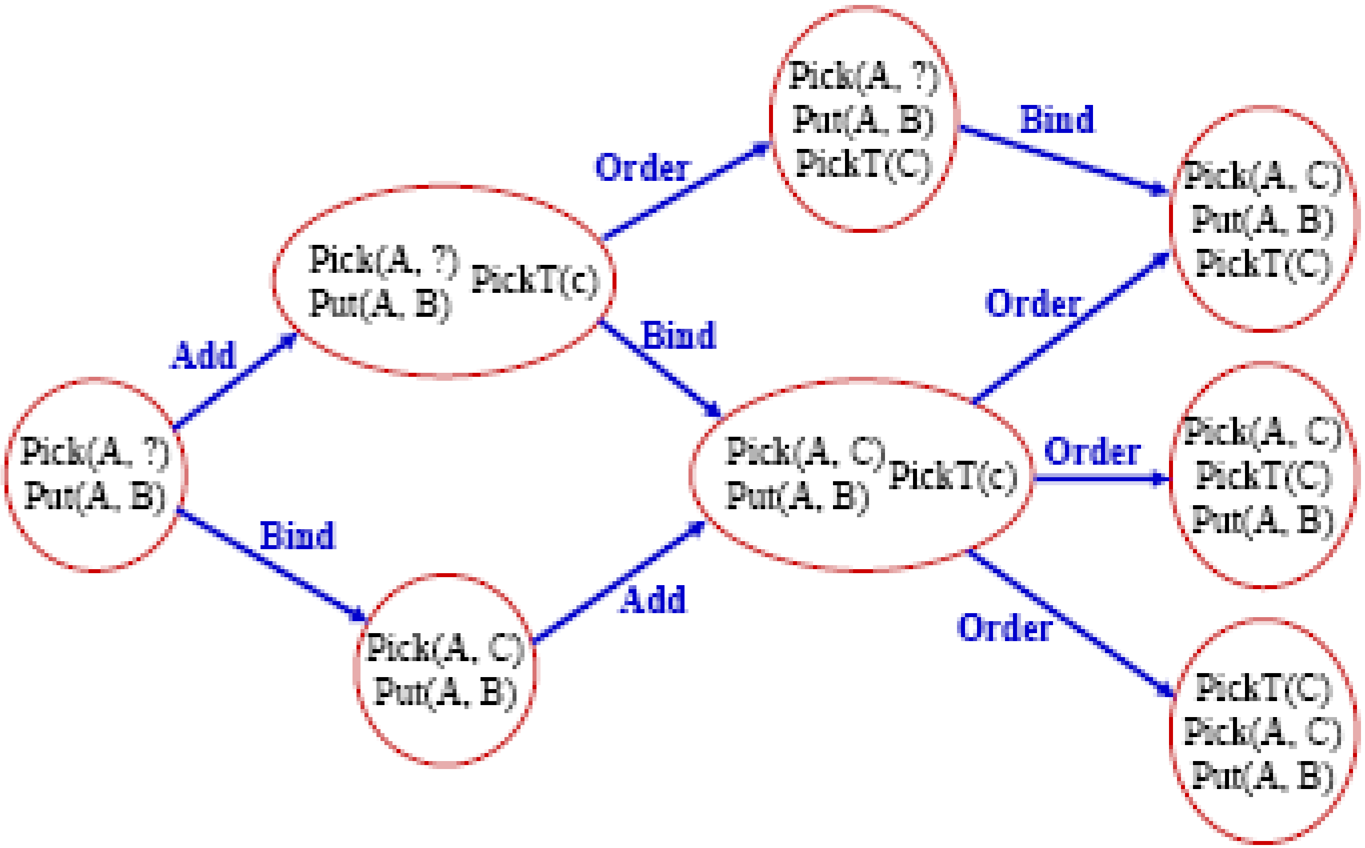
Operators on partial plans

- 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

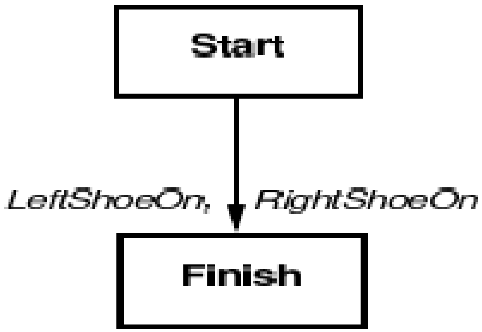
Open condition

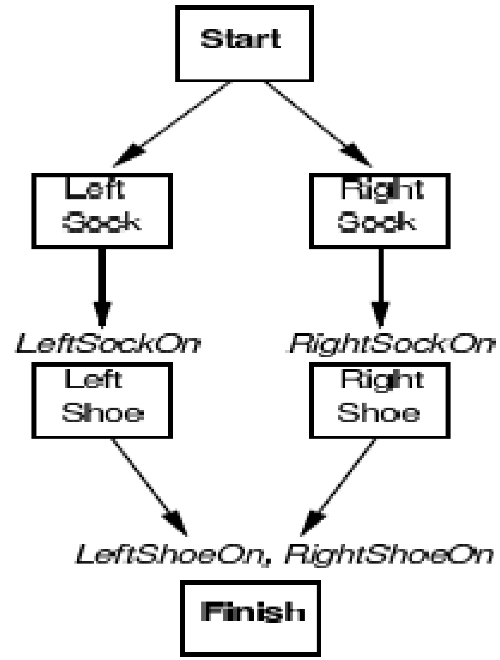
A precondition of an action not yet causally linked

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)

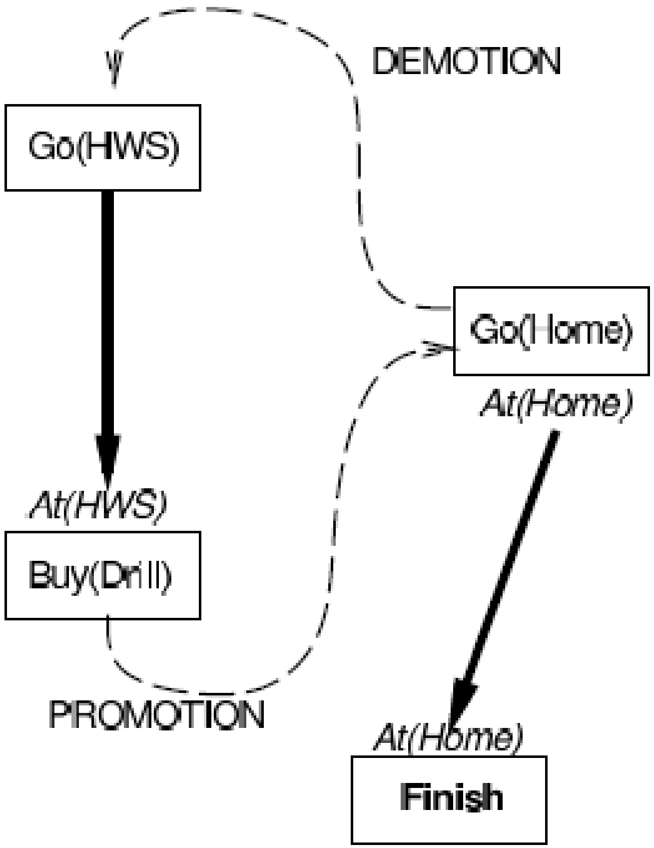
Clobberer / threat

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

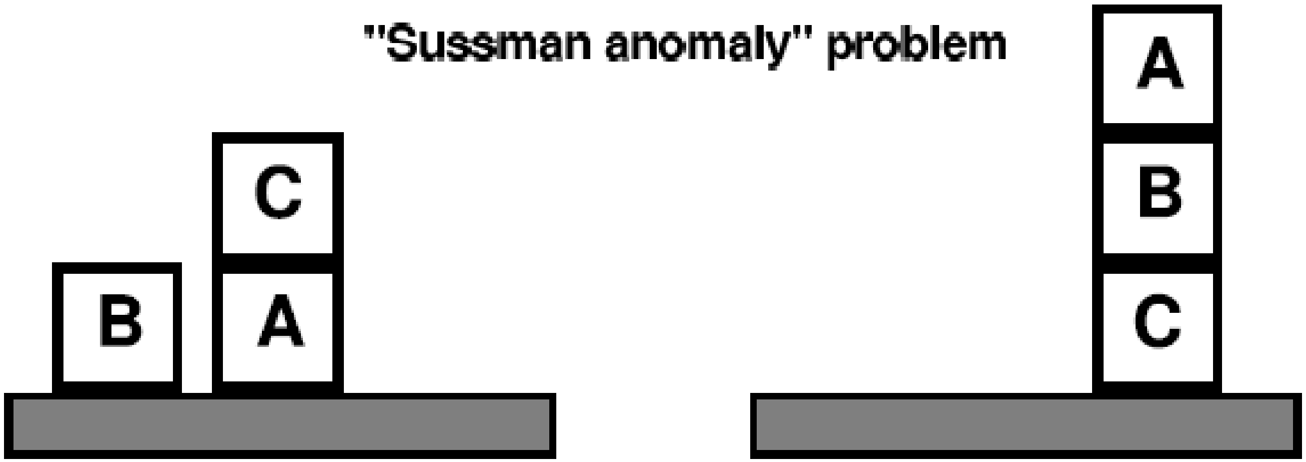
Put after $Buy(Drill)$

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



Start State

Goal State

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)

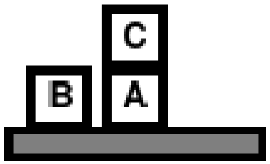
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Blocks World

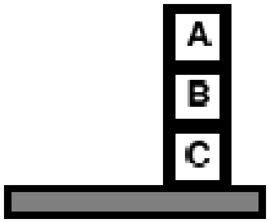
START

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

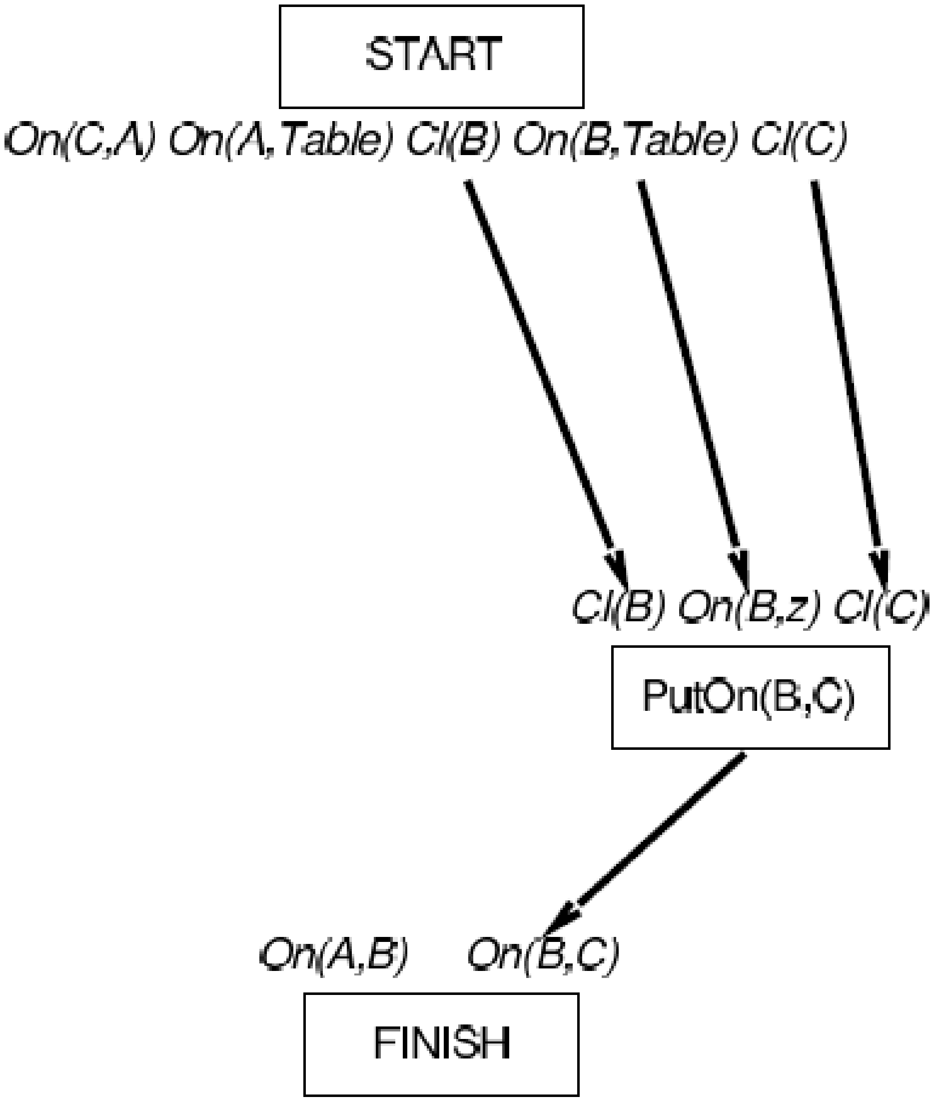


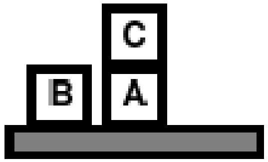
On(A,B) *On(B,C)*

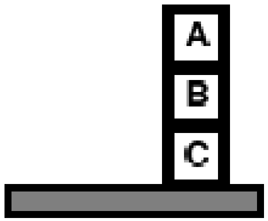
FINISH



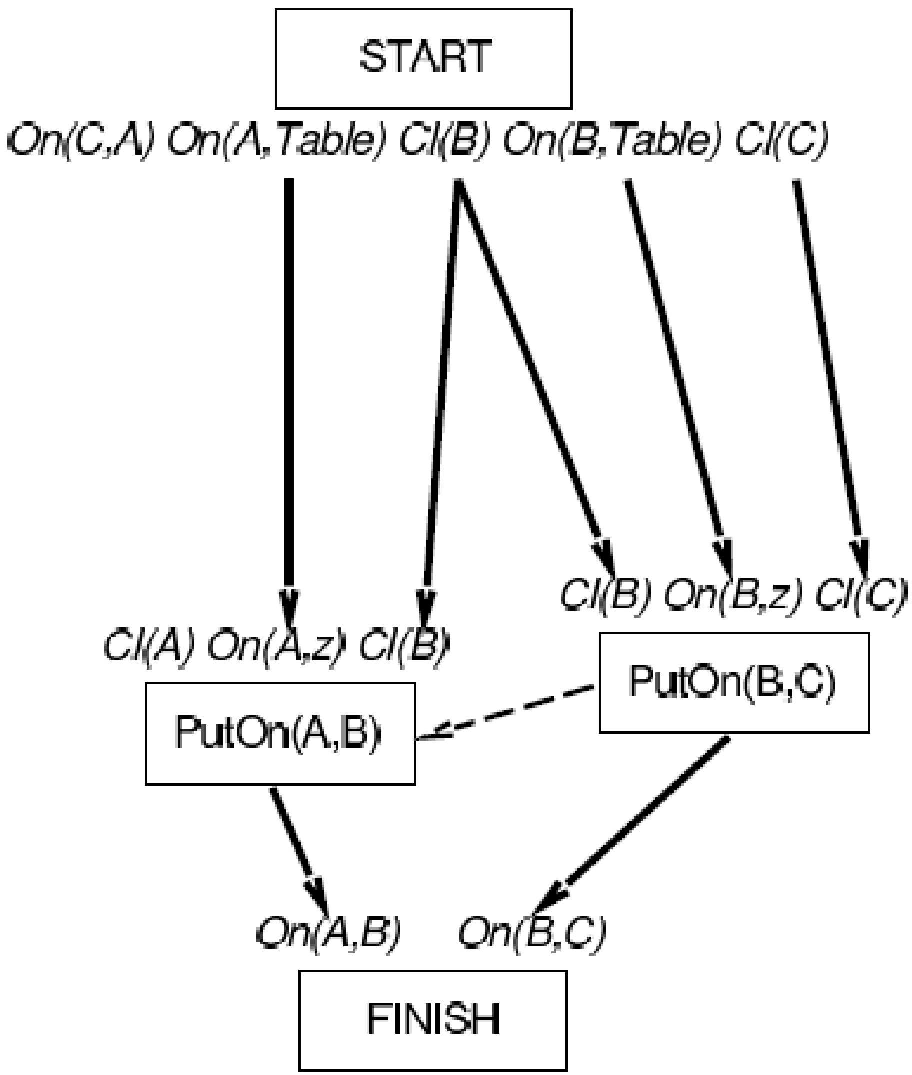
Blocks World

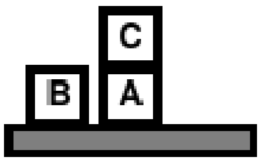




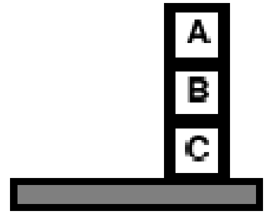


Blocks World

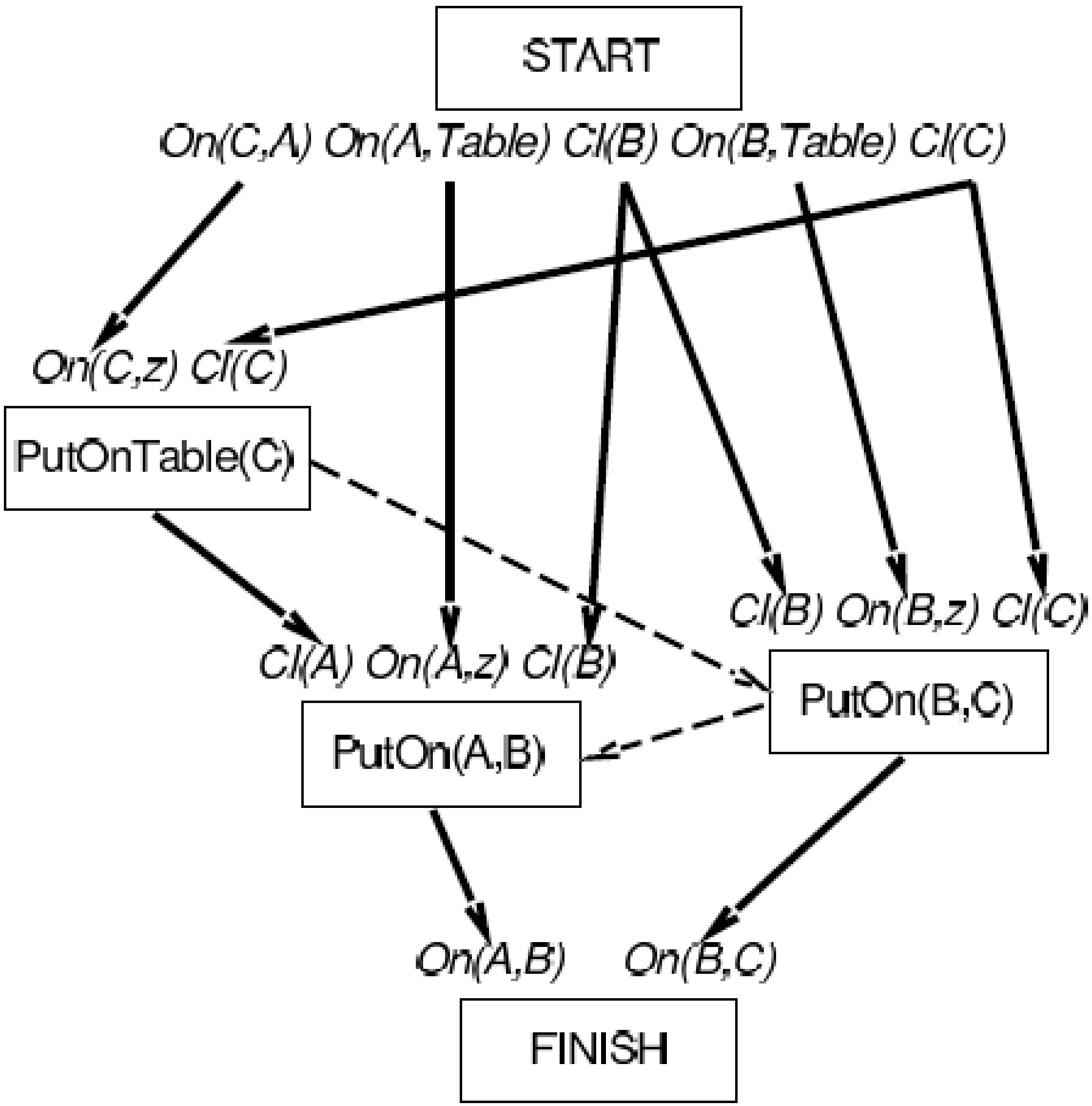


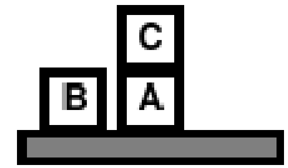


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



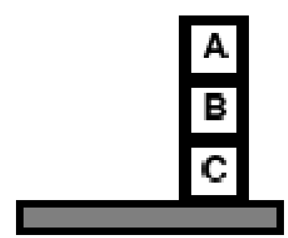
Blocks World



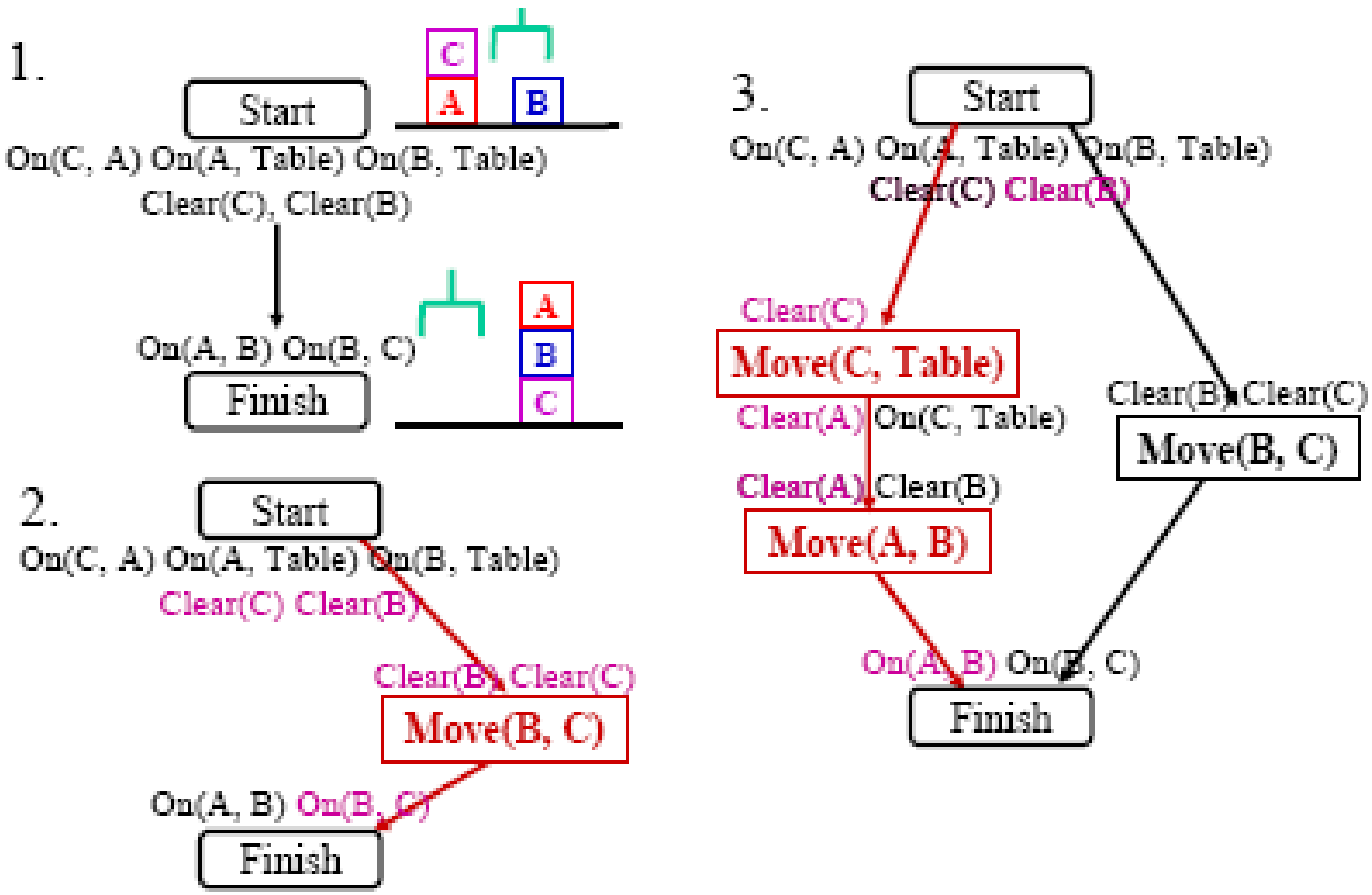


PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

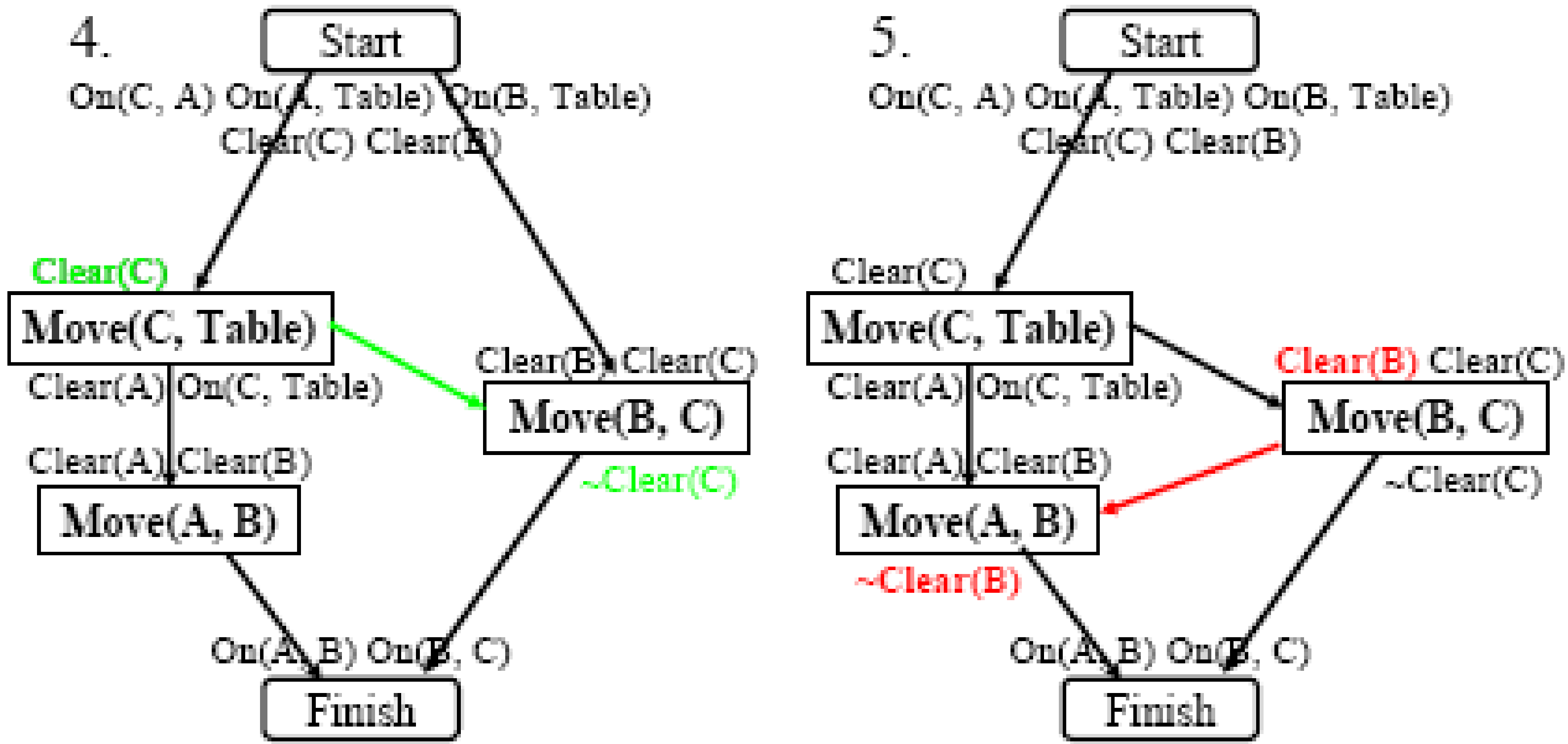
PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)



Blocks World



Blocks World



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

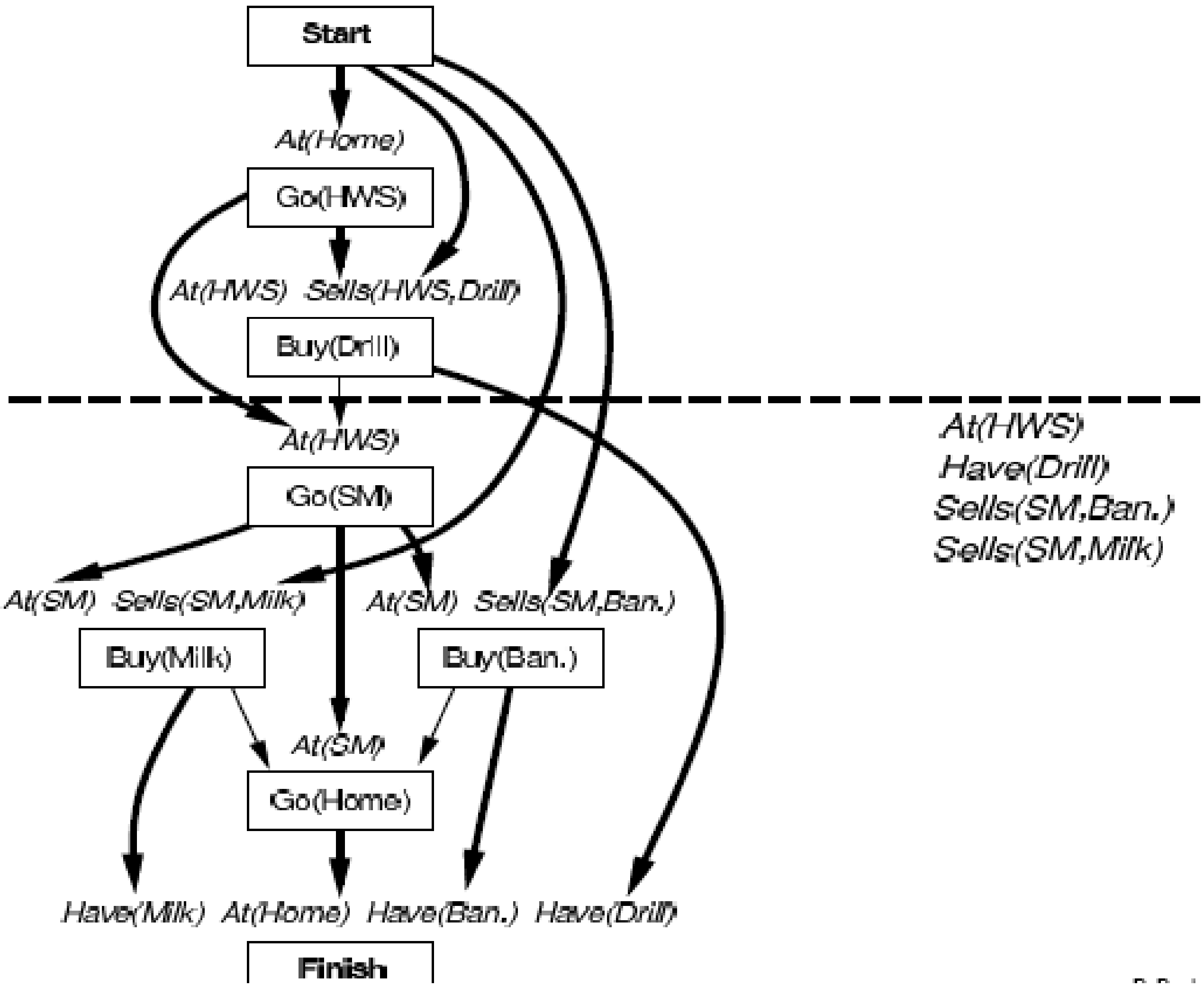
Failure: Preconditions of next action not met

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

Consequence of failure

Need to replan

Preconditions for the rest of the plan



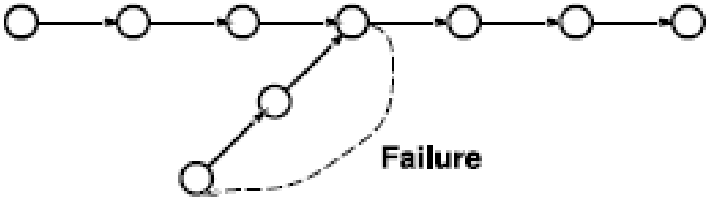
Replanning

Simplest

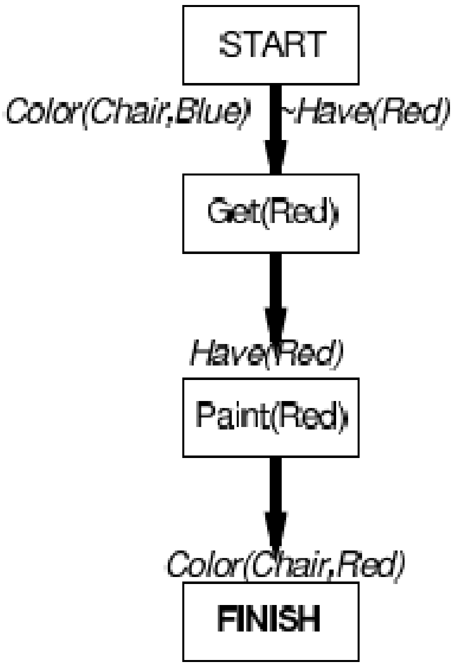
On failure, replan from scratch

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

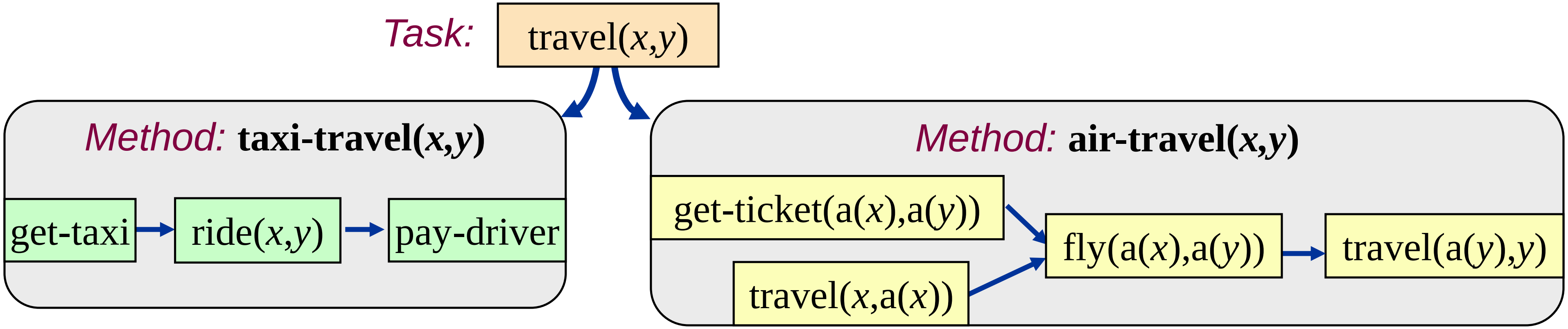
Motivation

- We may already have an idea how to go about solving problems in a planning domain
- Example: travel to a destination that's far away:
  - Domain-independent planner:
    - many combinations of vehicles and routes
  - Experienced human: small number of "recipes"

    e.g., flying:

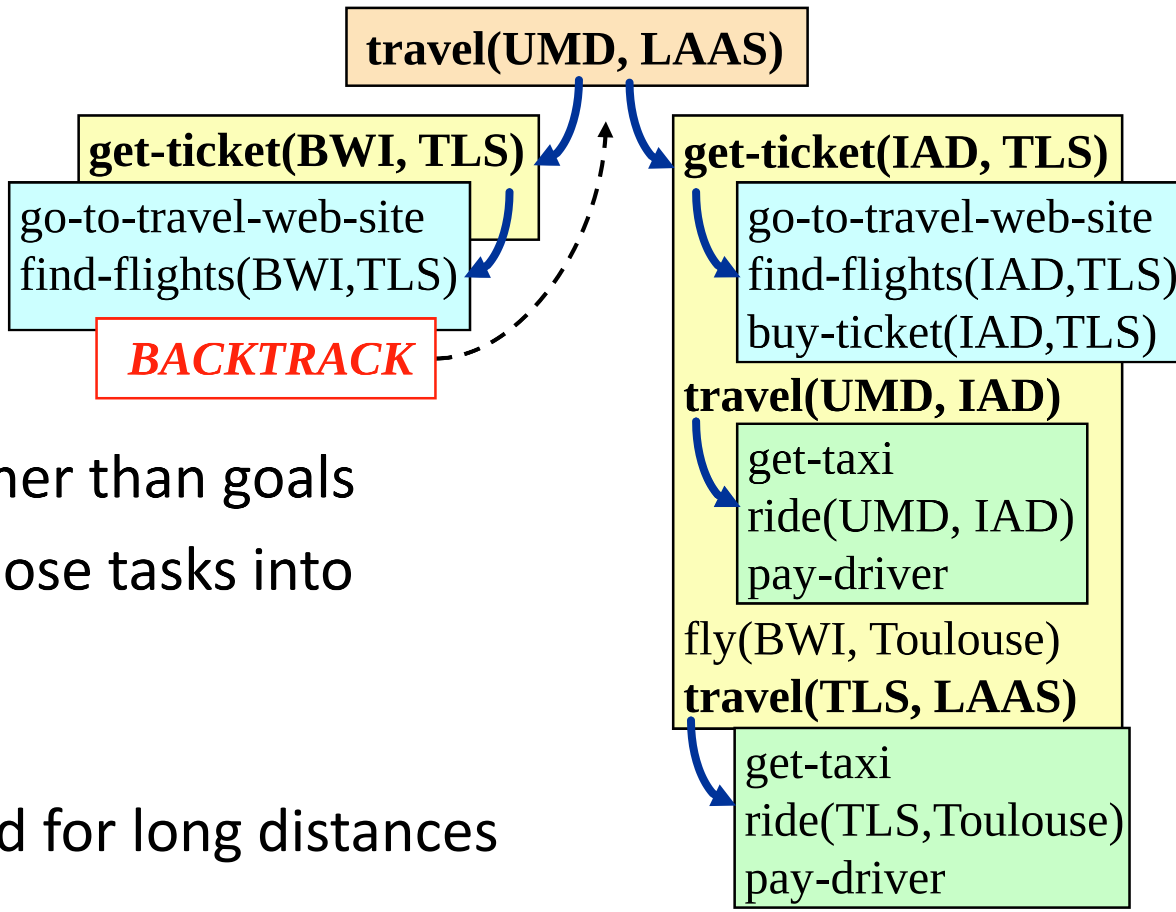
    1. buy ticket from local airport to remote airport
    2. travel to local airport
    3. fly to remote airport
    4. travel to final destination

- How to enable planning systems to make use of such recipes?



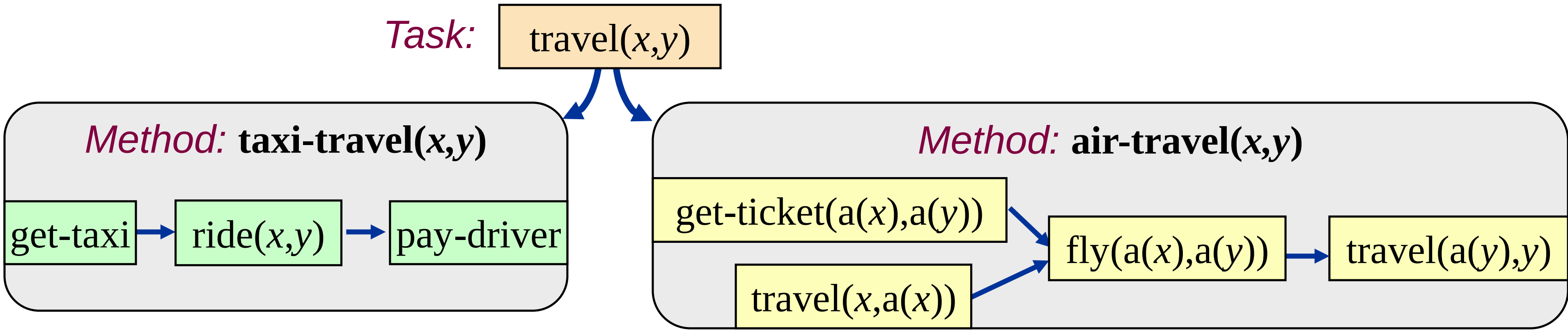
HTN Planning

- Problem reduction
  - *Tasks* (activities) rather than goals
  - *Methods* to decompose tasks into subtasks
  - Enforce constraints
    - E.g., taxi not good for long distances
  - Backtrack if necessary



HTN Planning

- HTN planners may be domain-specific
- Or they may be domain-configurable
  - Domain-independent planning engine
  - Domain description that defines not only the operators, but also the methods
  - Problem description
    - domain description, initial state, initial task network



Simple Task Network (STN) Planning

- A special case of HTN planning
- States and operators
  - The same as in classical planning
- *Task*: an expression of the form $t(u_1,\ldots,u_n)$
  - t is a *task symbol*, and each u_i is a term
  - Two kinds of task symbols (and tasks):
    - *primitive*: tasks that we know how to execute directly
      - task symbol is an operator name
    - *nonprimitive*: tasks that must be decomposed into subtasks
      - use *methods*

Methods

- Totally ordered method: a 4-tuple

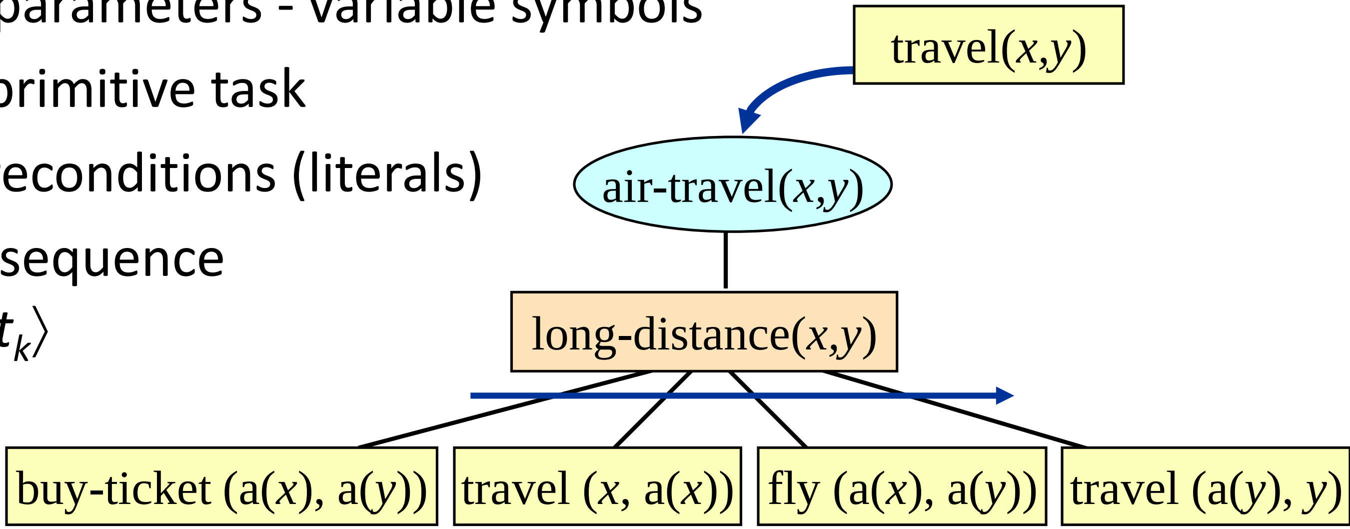
  m = (name(m), task(m), precond(m), subtasks(m))

  – name(m): an expression of the form $n(x_1,\ldots,x_n)$
    - $x_1,\ldots,x_n$ are parameters - variable symbols

  – task(m): a nonprimitive task

  – precond(m): preconditions (literals)

  – subtasks(m): a sequence of tasks $\langle t_1, \ldots, t_k \rangle$



air-travel(x,y)

task: travel(x,y)

precond: long-distance(x,y)

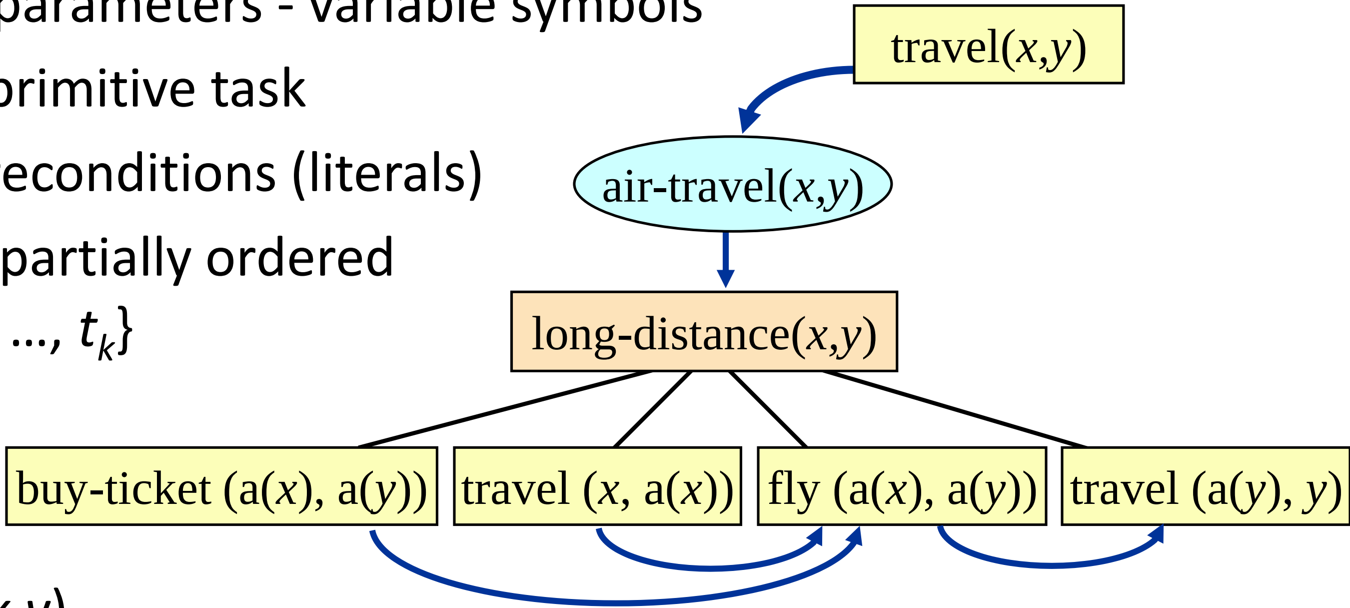
subtasks: $\langle$buy-ticket(a(x), a(y)), travel(x,a(x)), fly(a(x), a(y)), travel(a(y),y)$\rangle$

Methods (Continued)

- Partially ordered method: a 4-tuple

  m = (name(*m*), task(*m*), precond(*m*), subtasks(*m*))

  - name(*m*): an expression of the form $n(x_1,\ldots,x_n)$
    - $x_1,\ldots,x_n$ are parameters - variable symbols
  - task(*m*): a nonprimitive task
  - precond(*m*): preconditions (literals)
  - subtasks(*m*): a partially ordered set of tasks $\{t_1, \ldots, t_k\}$



air-travel(*x,y*)

task: travel(*x,y*)

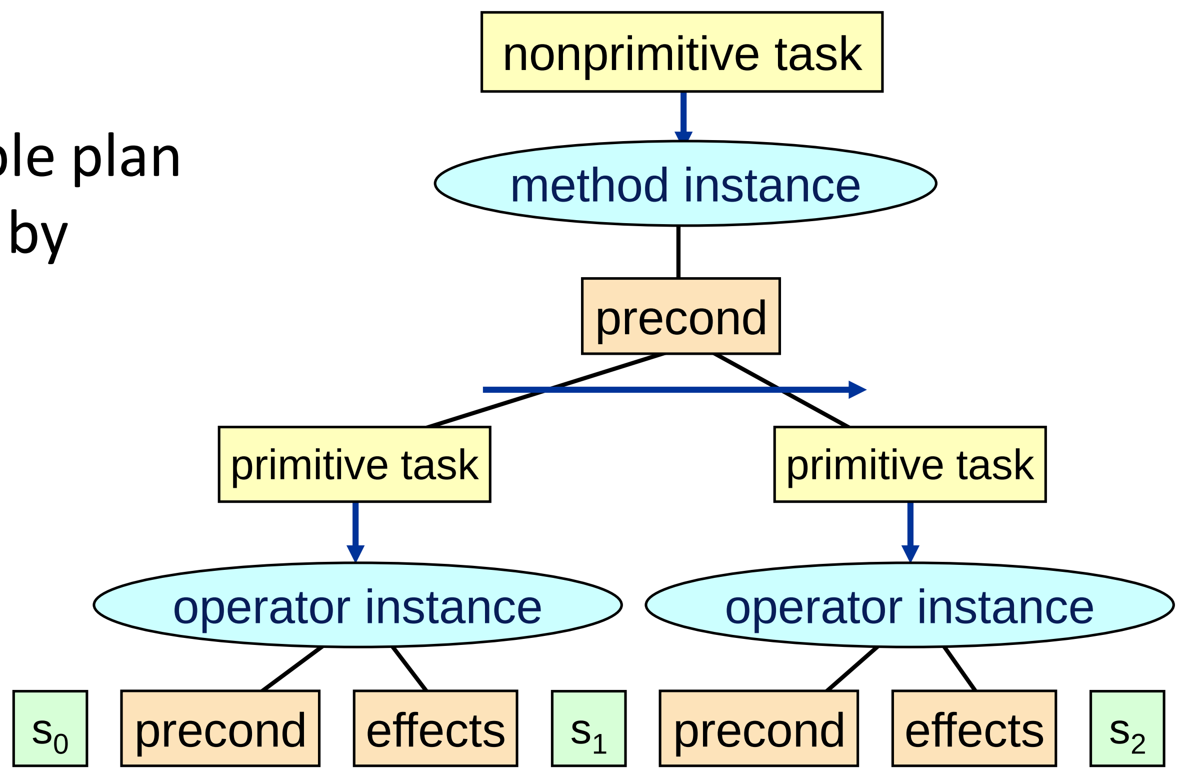
precond: long-distance(*x,y*)

network: u_1=buy-ticket(a(x),a(y)), u_2= travel(x,a(x)), u_3= fly(a(x), a(y))

u_4= travel(a(y),y), $\{(u_1,u_3), (u_2,u_3), (u_3,u_4)\}$

Domains, Problems, Solutions

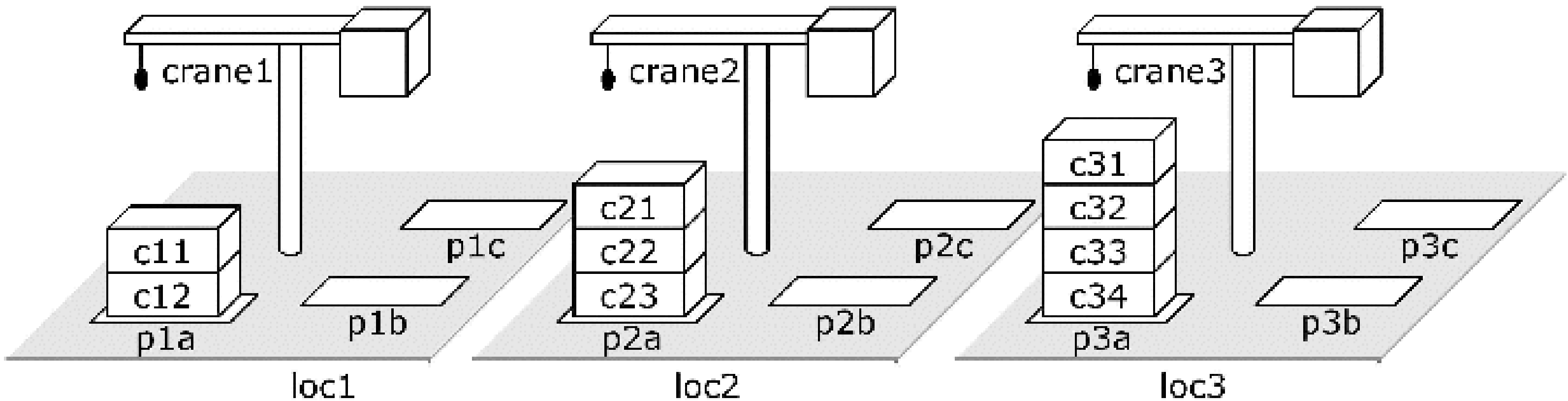
- STN planning domain: methods, operators
- STN planning problem: methods, operators, initial state, task list
- Total-order STN planning domain and planning problem:



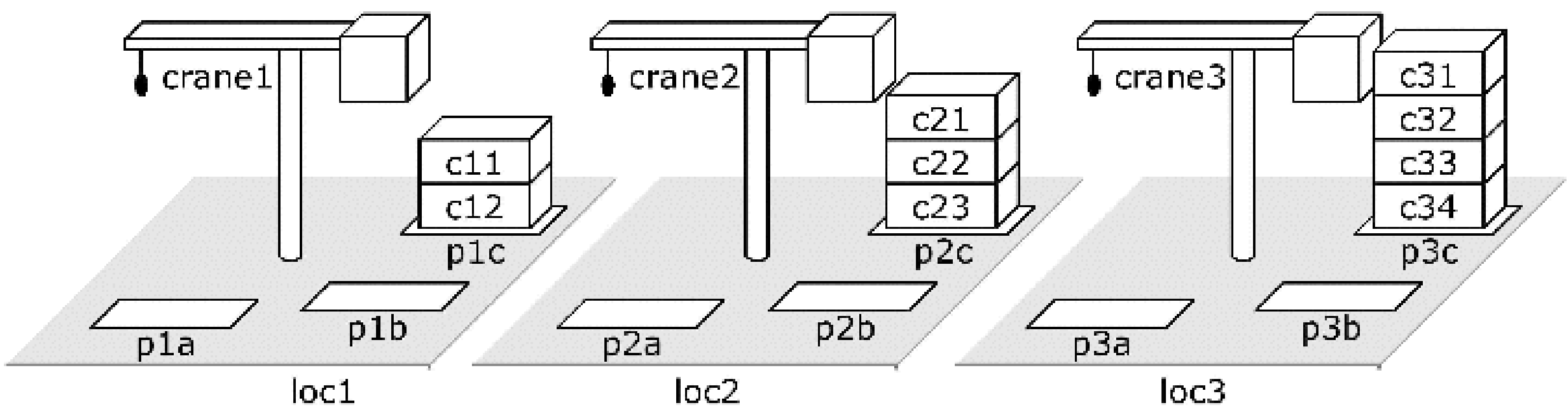
- Solution: any executable plan that can be generated by recursively applying
  - methods to nonprimitive tasks
  - operators to primitive tasks

Example

- Suppose we want to move three stacks of containers in a way that preserves the order of the containers



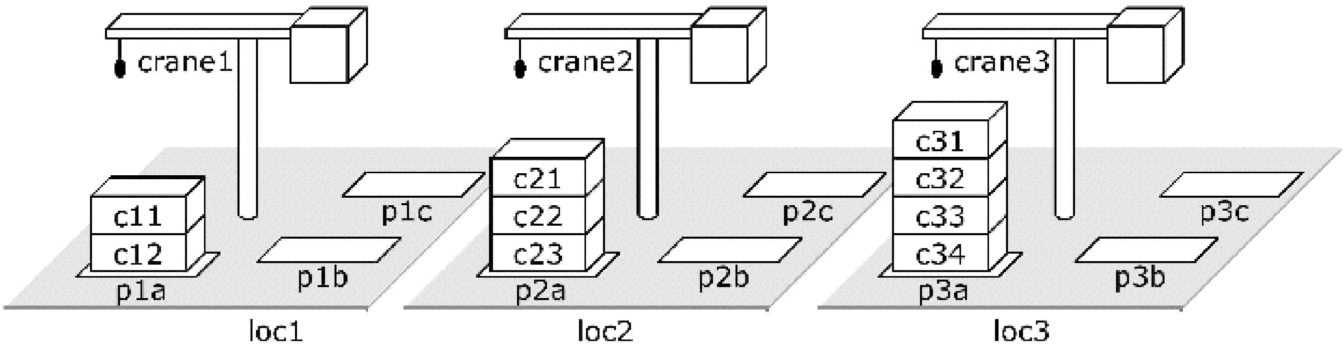
(a) initial state



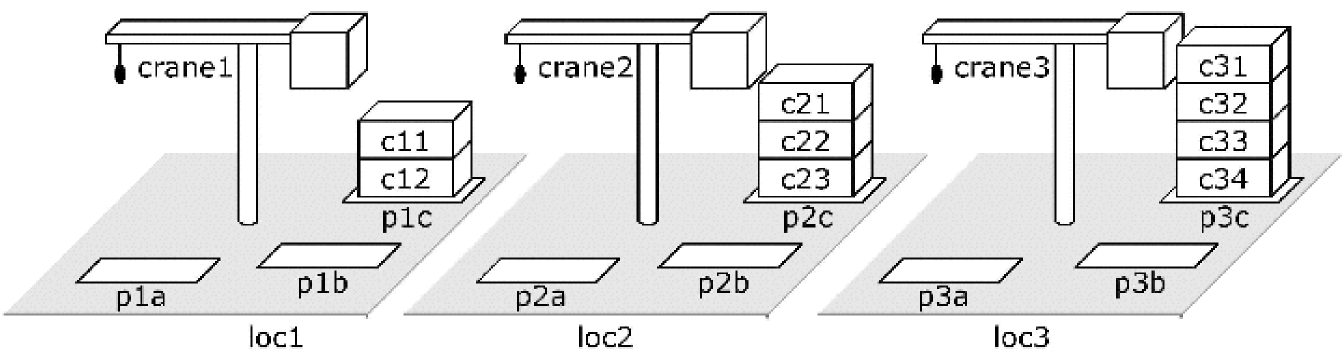
(b) goal

Example (continued)

- A way to move each stack:
  - first move the containers from *p* to an intermediate pile *r*
  - then move them from *r* to *q*



(a) initial state

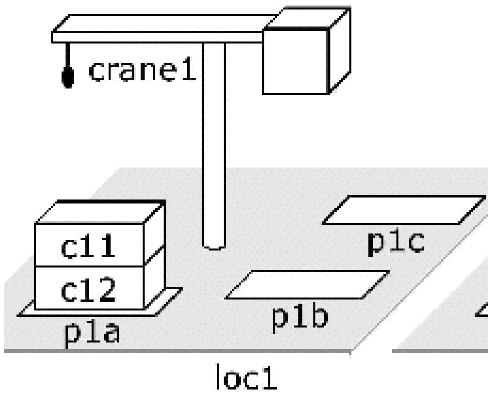


(b) goal

Total-Order Formulation

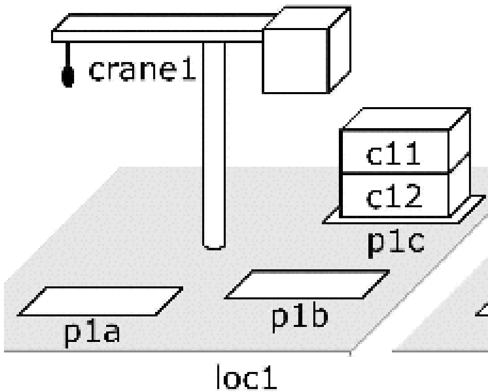
take-and-put$(c, k, l_1, l_2, p_1, p_2, x_1, x_2)$:
- task: move-topmost-container(p_1, p_2)
- precond: top(c, p_1), on(c, x_1), *; true if p_1 is not empty*
  attached(p_1, l_1), belong(k, l_1), *; bind l_1 and k*
  attached(p_2, l_2), top(x_2, p_2) *; bind l_2 and x_2*
- subtasks: ⟨take(k, l_1, c, x_1, p_1), put(k, l_2, c, x_2, p_2)⟩

recursive-move(p, q, c, x):
- task: move-stack(p, q)
- precond: top(c, p), on(c, x) *; true if p is not empty*
- subtasks: ⟨move-topmost-container(p, q), move-stack(p, q)⟩
  ;; the second subtask recursively moves the rest of the stack



do-nothing(p, q)
- task: move-stack(p, q)
- precond: top$(pallet, p)$ *; true if p is empty*
- subtasks: ⟨⟩ *; no subtasks, because we are done*

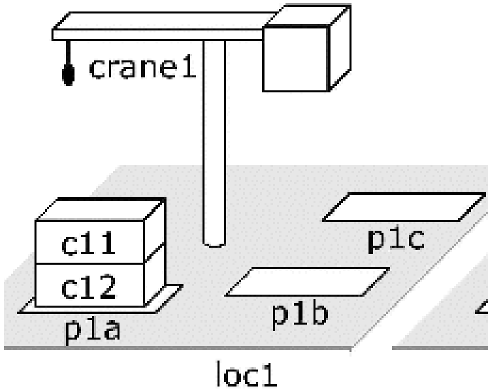
move-each-twice()
- task: move-all-stacks()
- precond: *; no preconditions*
- subtasks: *; move each stack twice:*
  ⟨move-stack(p1a,p1b), move-stack(p1b,p1c),
  move-stack(p2a,p2b), move-stack(p2b,p2c),
  move-stack(p3a,p3b), move-stack(p3b,p3c)⟩



Partial-Order Formulation

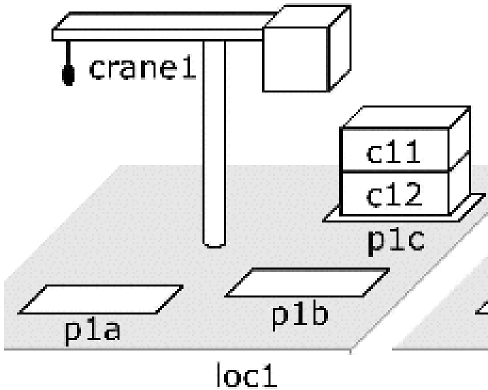
take-and-put($c, k, l_1, l_2, p_1, p_2, x_1, x_2$):
task: move-topmost-container(p_1, p_2)
precond: top(c, p_1), on(c, x_1), *; true if p_1 is not empty*
attached(p_1, l_1), belong(k, l_1), *; bind l_1 and k*
attached(p_2, l_2), top(x_2, p_2) *; bind l_2 and x_2*
subtasks: ⟨take(k, l_1, c, x_1, p_1), put(k, l_2, c, x_2, p_2)⟩

recursive-move(p, q, c, x):
task: move-stack(p, q)
precond: top(c, p), on(c, x) *; true if p is not empty*
subtasks: ⟨move-topmost-container(p, q), move-stack(p, q)⟩
;; the second subtask recursively moves the rest of the stack



do-nothing(p, q)
task: move-stack(p, q)
precond: top($pallet, p$) *; true if p is empty*
subtasks: ⟨⟩ *; no subtasks, because we are done*

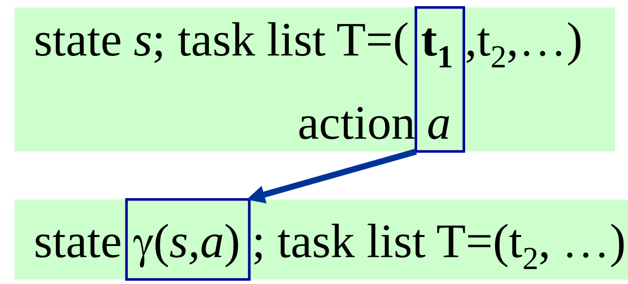
move-each-twice()
task: move-all-stacks()
precond: *; no preconditions*
network: *; move each stack twice:*
u_1 =move-stack(p1a,p1b), u_2 =move-stack(p1b,p1c),
u_3 =move-stack(p2a,p2b), u_4 =move-stack(p2b,p2c),
u_5 =move-stack(p3a,p3b), u_6 =move-stack(p3b,p3c),
$\{(u_1, u_2), (u_3, u_4), (u_5, u_6)\}$

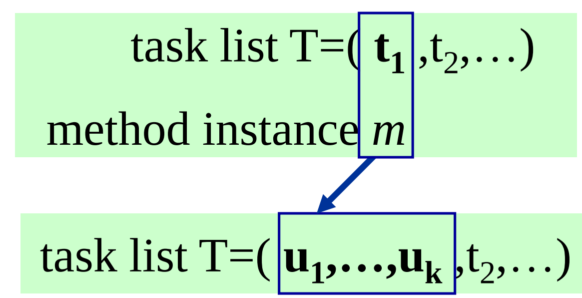


Solving Total-Order STN Planning Problems

Total-order Forward Dec

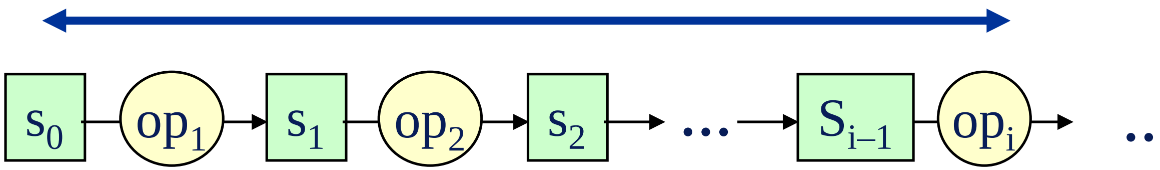
TFD$(s, \langle t_1, \ldots, t_k \rangle, O, M)$
  if $k = 0$ then return $\langle\rangle$ (i.e., the empty plan)
  if t_1 is primitive then
    active $\leftarrow \{(a, \sigma) \mid a$ is a ground instance of an operator in O,
      σ is a substitution such that a is relevant for $\sigma(t_1)$,
      and a is applicable to $s\}$
    if *active* $= \emptyset$ then return failure
    nondeterministically choose any $(a, \sigma) \in$ *active*
    $\pi \leftarrow$ TFD$(\gamma(s, a), \sigma(\langle t_2, \ldots, t_k \rangle), O, M)$
    if π = failure then return failure
    else return $a.\pi$
  else if t_1 is nonprimitive then
    active $\leftarrow \{m \mid m$ is a ground instance of a method in M,
      σ is a substitution such that m is relevant for $\sigma(t_1)$,
      and m is applicable to $s\}$
    if *active* $= \emptyset$ then return failure
    nondeterministically choose any $(m, \sigma) \in$ *active*
    $w \leftarrow$ subtasks$(m).\sigma(\langle t_2, \ldots, t_k \rangle)$
    return TFD(s, w, O, M)



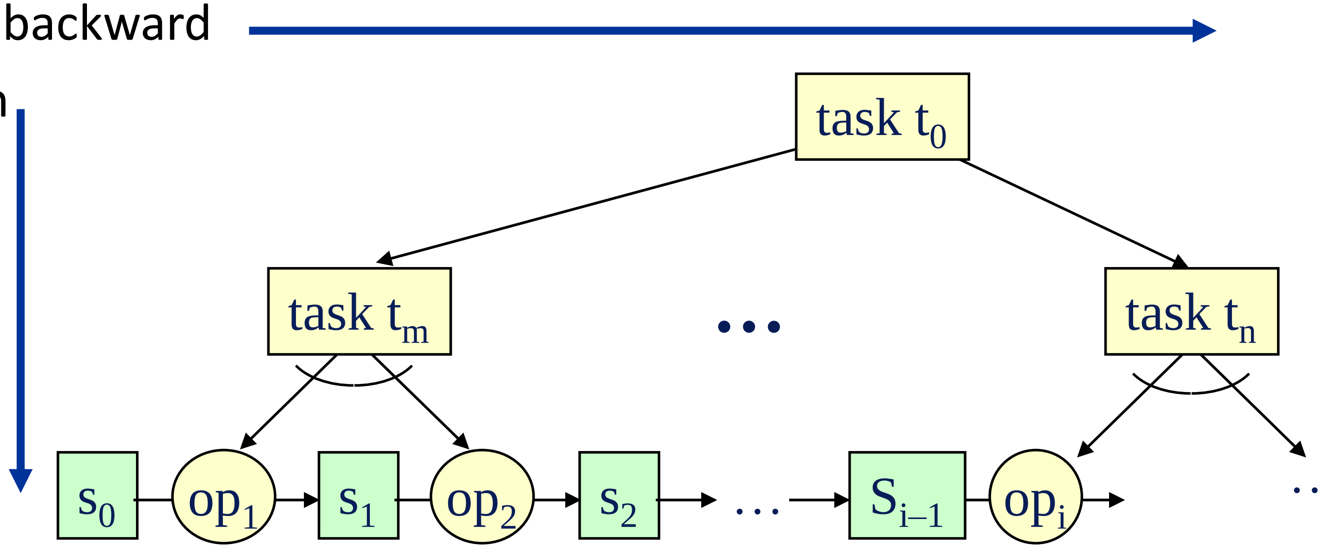


Comparison to Forward and Backward Search

- In state-space planning, must choose whether to search forward or backward

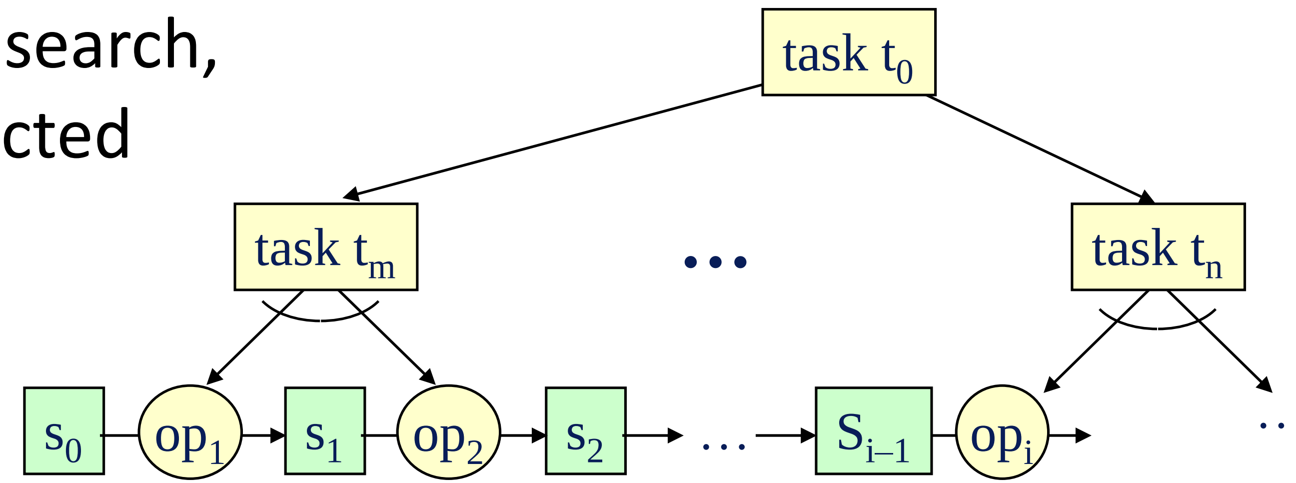


- In HTN planning, there are *two* choices to make about direction:
  - forward or backward
  - up or down
- TFD goes *down* and *forward*



Comparison to Forward and Backward Search

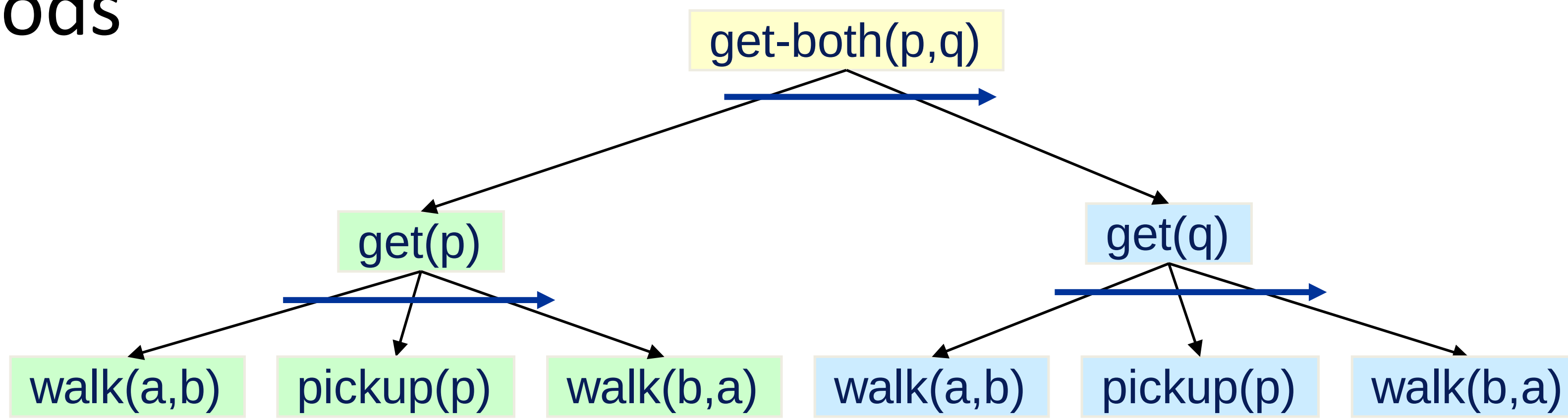
- Like a backward search, TFD is goal-directed
  - Goals correspond to tasks



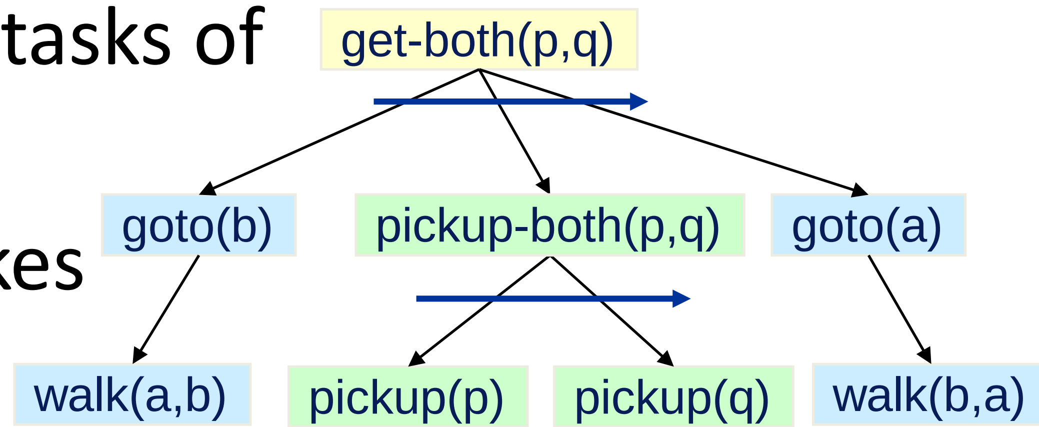
- Like a forward search, it generates actions in the same order in which they'll be executed
- Whenever we want to plan the next task
  - we've already planned everything that comes before it
  - we know the current state of the world

Limitation of Ordered-Task Planning

- TFD requires totally ordered methods



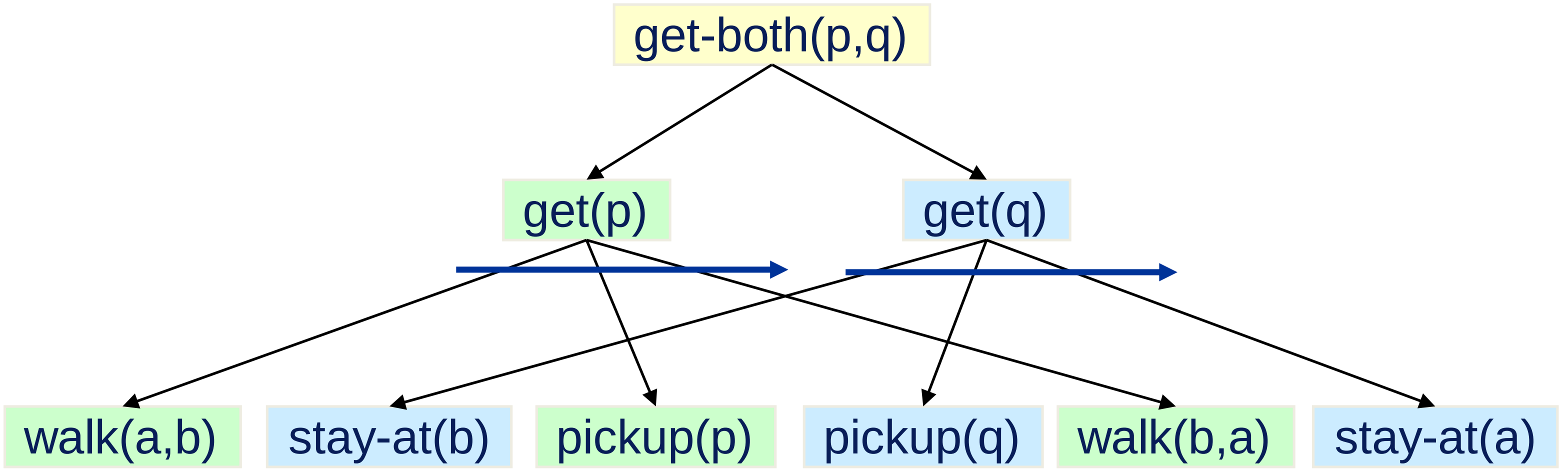
- Can’t interleave subtasks of different tasks



- Sometimes this makes things awkward
  - Need to write methods that reason globally instead of locally

Partially Ordered Methods

- With partially ordered methods, the subtasks can be interleaved

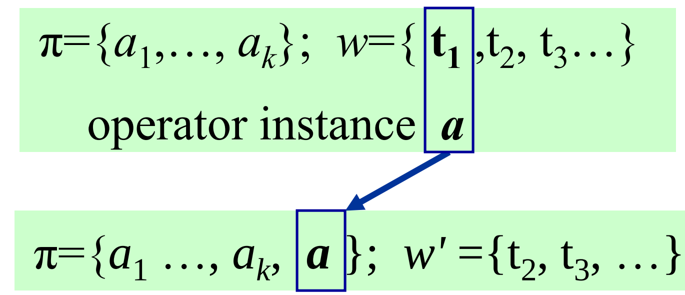


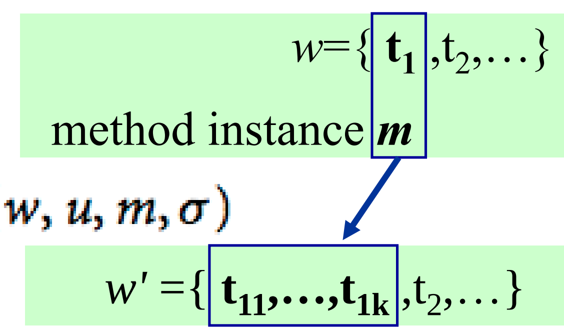
- Fits many planning domains better
- Requires a more complicated planning algorithm

Algorithm for Partial-Order STNs

Partial-order Forward Dec

PFD(s, w, O, M)
  if $w = \emptyset$ then return the empty plan
  nondeterministically choose any $u \in w$ that has no predecessors in w
  if t_u is a primitive task then
    active $\leftarrow \{(a, \sigma) \mid a$ is a ground instance of an operator in O,
      σ is a substitution such that name$(a) = \sigma(t_u)$,
      and a is applicable to $s\}$
    if *active* $= \emptyset$ then return failure
    nondeterministically choose any $(a, \sigma) \in$ *active*
    $\pi \leftarrow$ PFD$(\gamma(s, a), \sigma(w - \{u\}), O, M)$
    if π = failure then return failure
    else return $a.\pi$
  else
    active $\leftarrow \{(m, \sigma) \mid m$ is a ground instance of a method in M,
      σ is a substitution such that name$(m) = \sigma(t_u)$,
      and m is applicable to $s\}$
    if *active* $= \emptyset$ then return failure
    nondeterministically choose any $(m, \sigma) \in$ *active*
    nondeterministically choose any task network $w' \in \delta(w, u, m, \sigma)$
    return(PFD(s, w', O, M))





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

linked

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

Observation-2

Observation-3

Observation-4

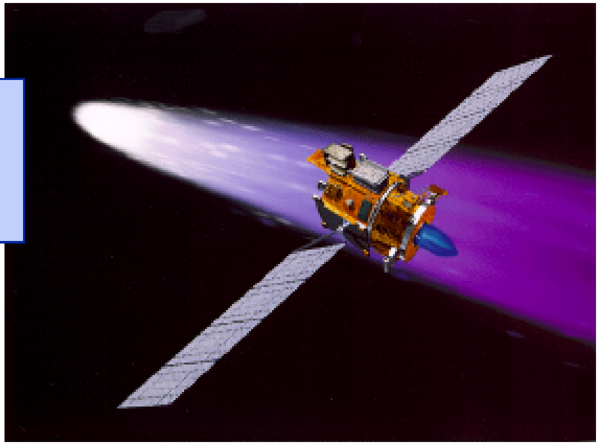
…

angle between targets

⇒ turn duration

calibration
- target1
- target2
- …

consumables:
- fuel
- power
- data storage
- cryogen



Objective:

maximize science return

Based on slides by Dave Smith, NASA Ames

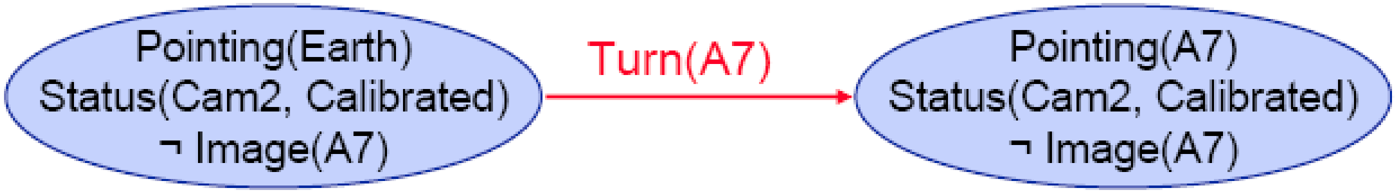
Extensions

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

Model

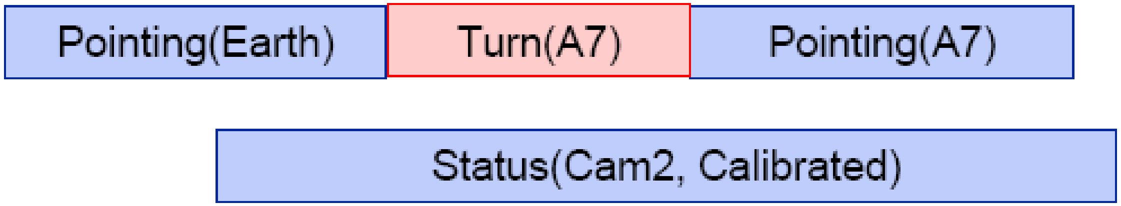
State-centric (Mc Carthy):

for each time describe propositions that are true

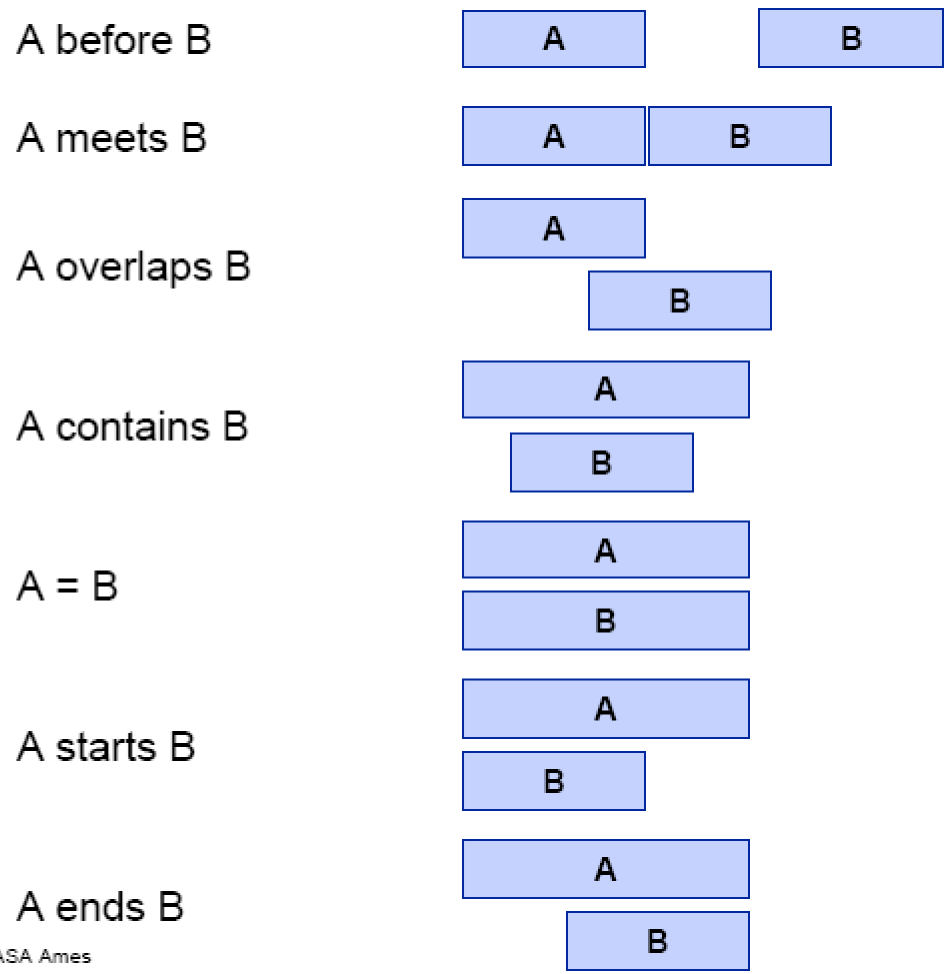


History-based (Hayes):

for each proposition describe times it is true



Temporal Interval Relations



Based on slides by Dave Smith, NASA Ames

Temporal Operators

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

↓

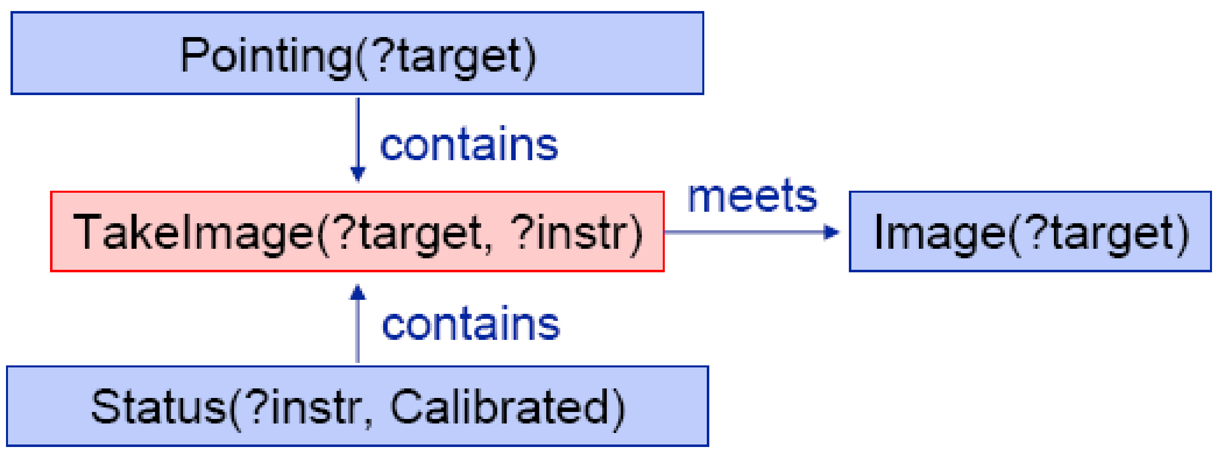
TakeImage (?target, ?instr)	
contained-by	Status(?instr, Calibrated)
contained-by	Pointing(?target)
meets	Image(?target)

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)

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

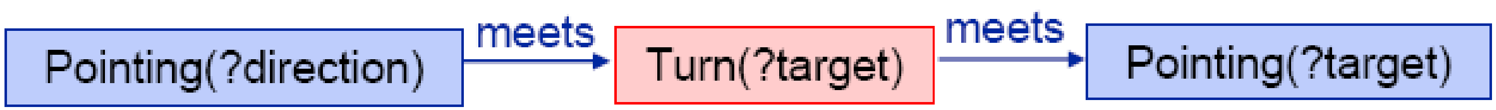
$$\Rightarrow \exists P\ \{\text{Status}(?instr, \text{Calibrated})_P \land \text{Contains}(P, A)\}$$

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

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

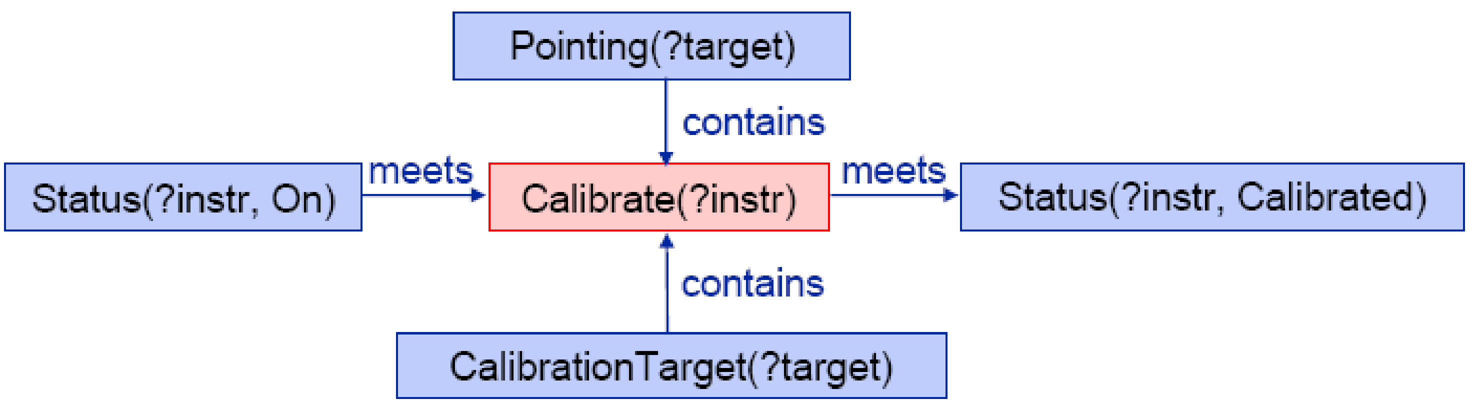
Temporal Operators

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



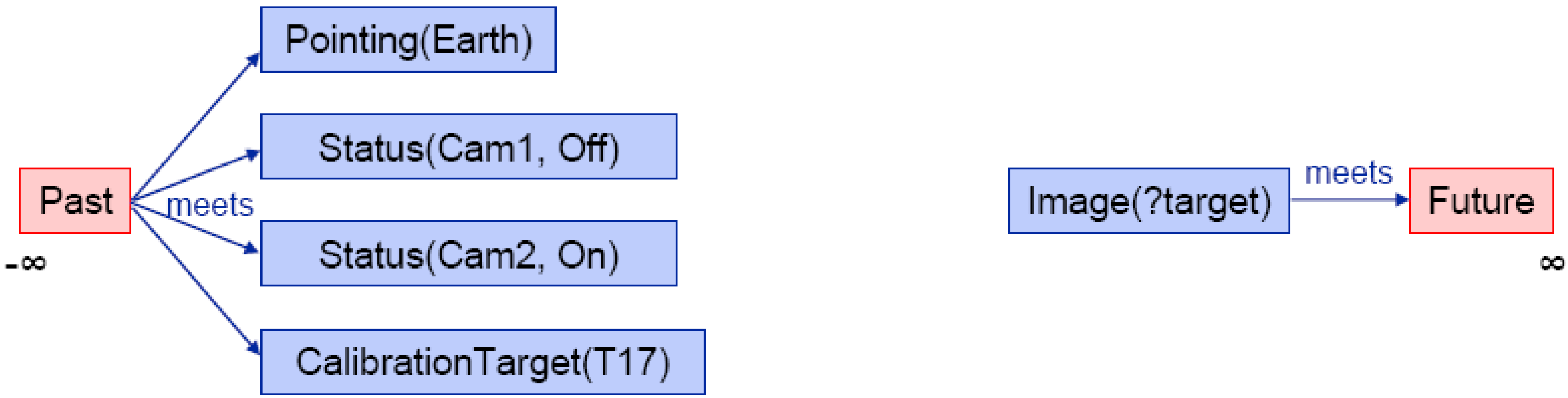
Temporal Operators

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



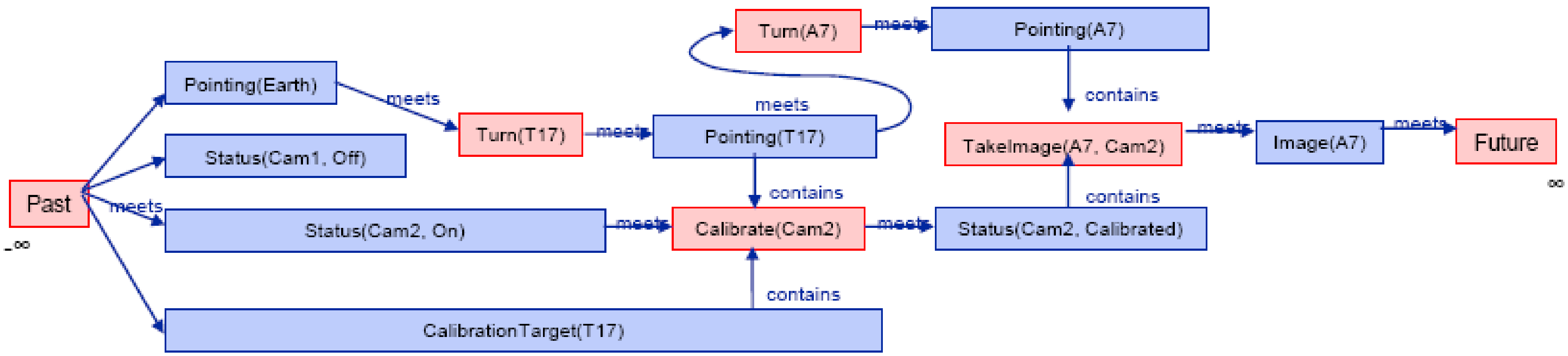
Based on slides by Dave Smith, NASA Ames

Temporal Planning Problem



Based on slides by Dave Smith, NASA Ames

Consistent Complete Plan



Based on slides by Dave Smith, NASA Ames

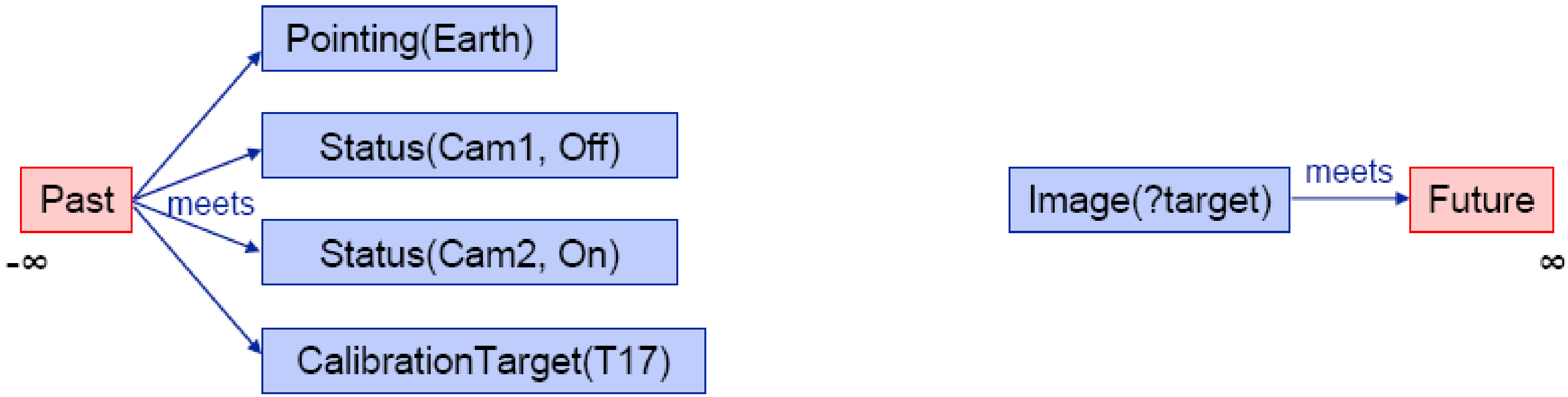
CBI-Planning

Choose:

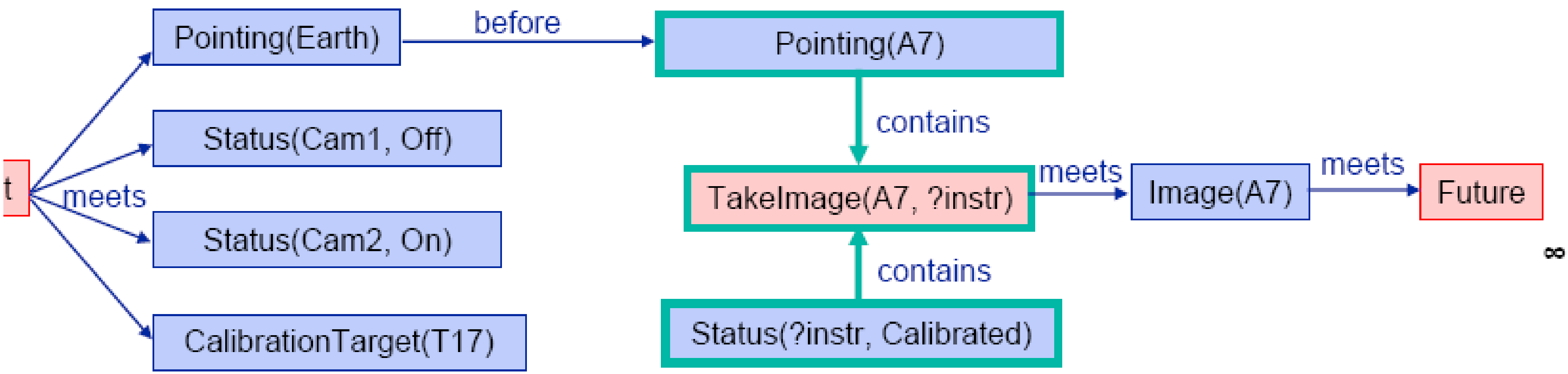
- introduce an action & instantiate constraints
- coalesce propositions

Propagate constraints

Initial Plan

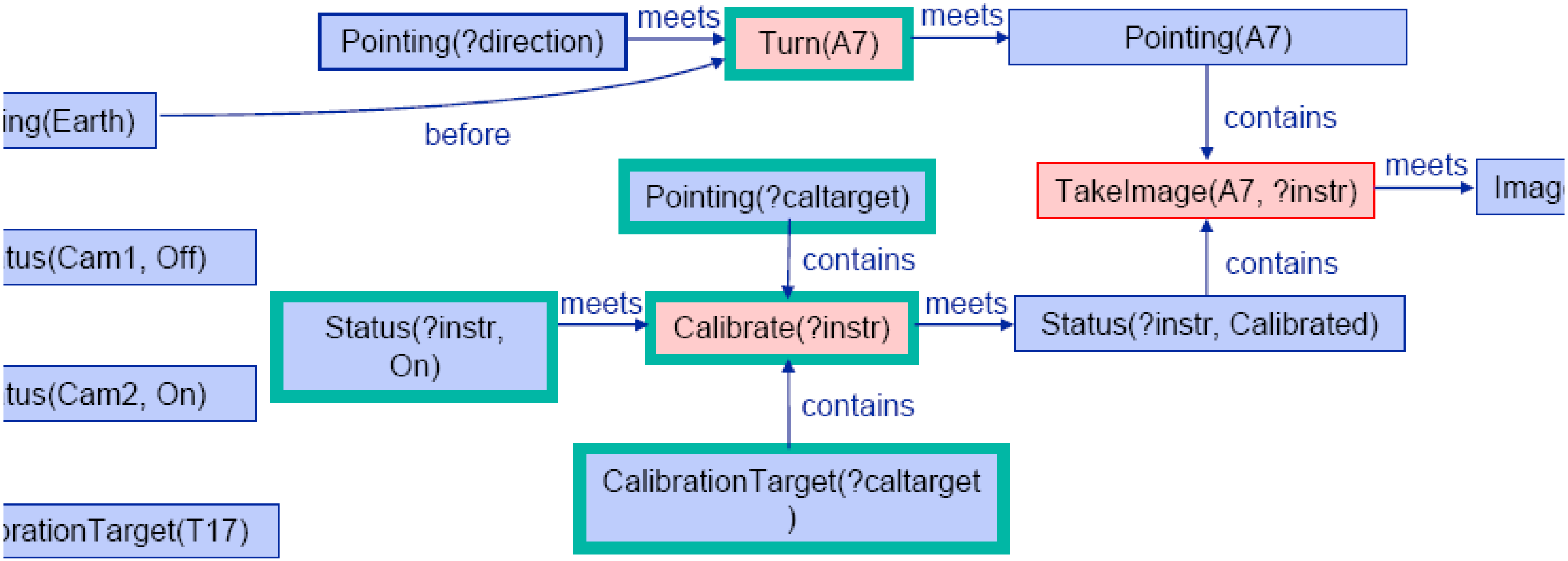


Expansion



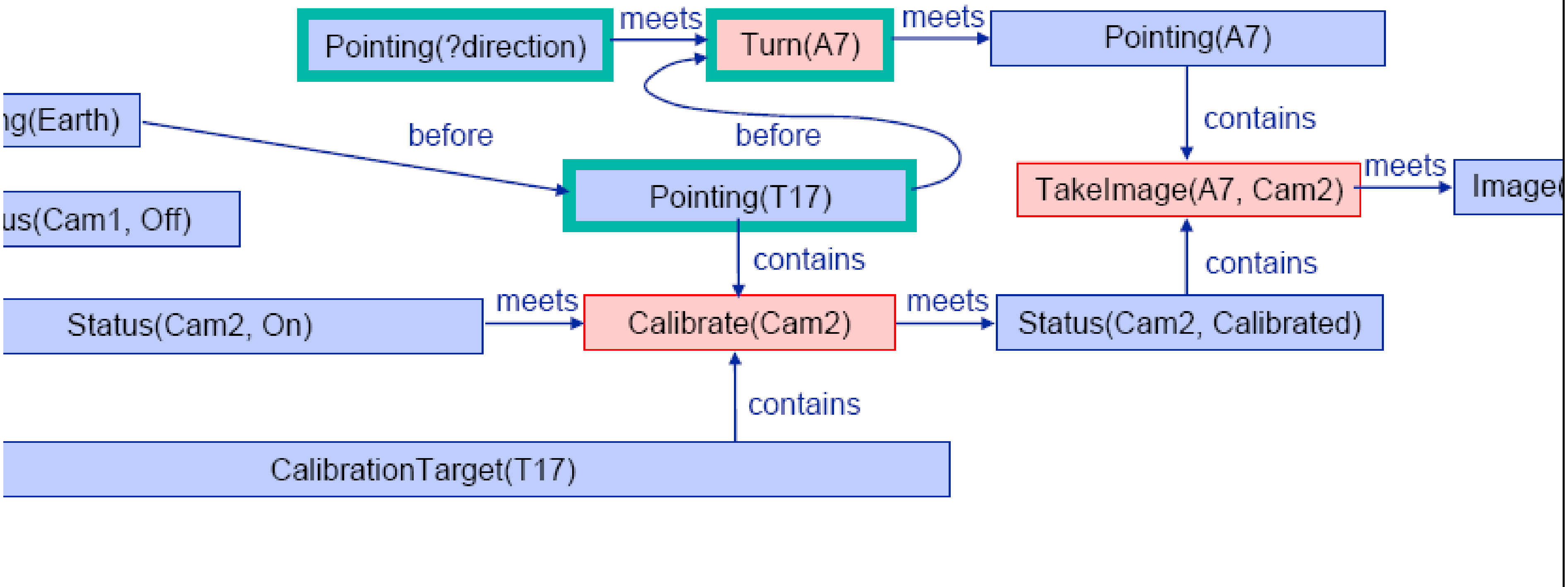
Pointing(Earth)
before
Pointing(A7)
Status(Cam1, Off)
contains
t
meets
TakeImage(A7, ?instr)
meets
Image(A7)
meets
Future
Status(Cam2, On)
∞
contains
CalibrationTarget(T17)
Status(?instr, Calibrated)

Expansion



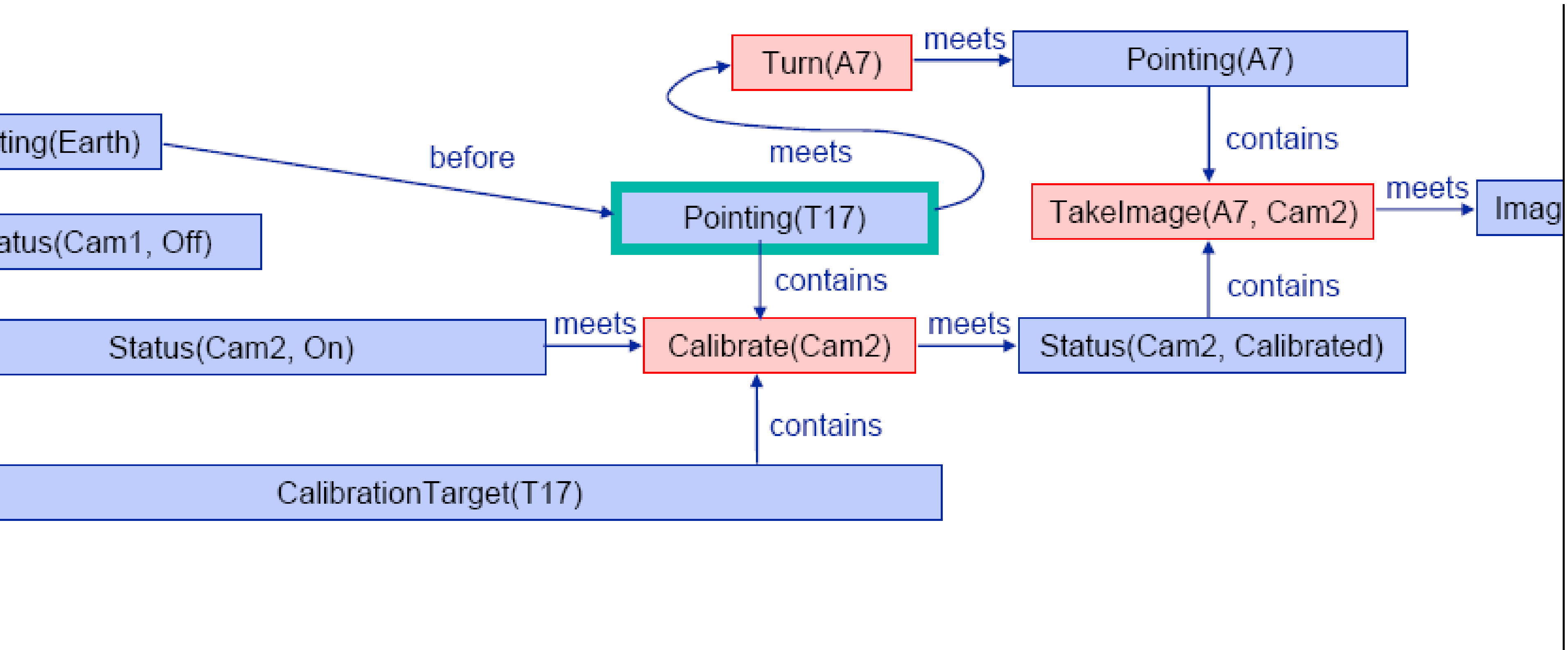
Based on slides by Dave Smith, NASA Ames

Coalescing



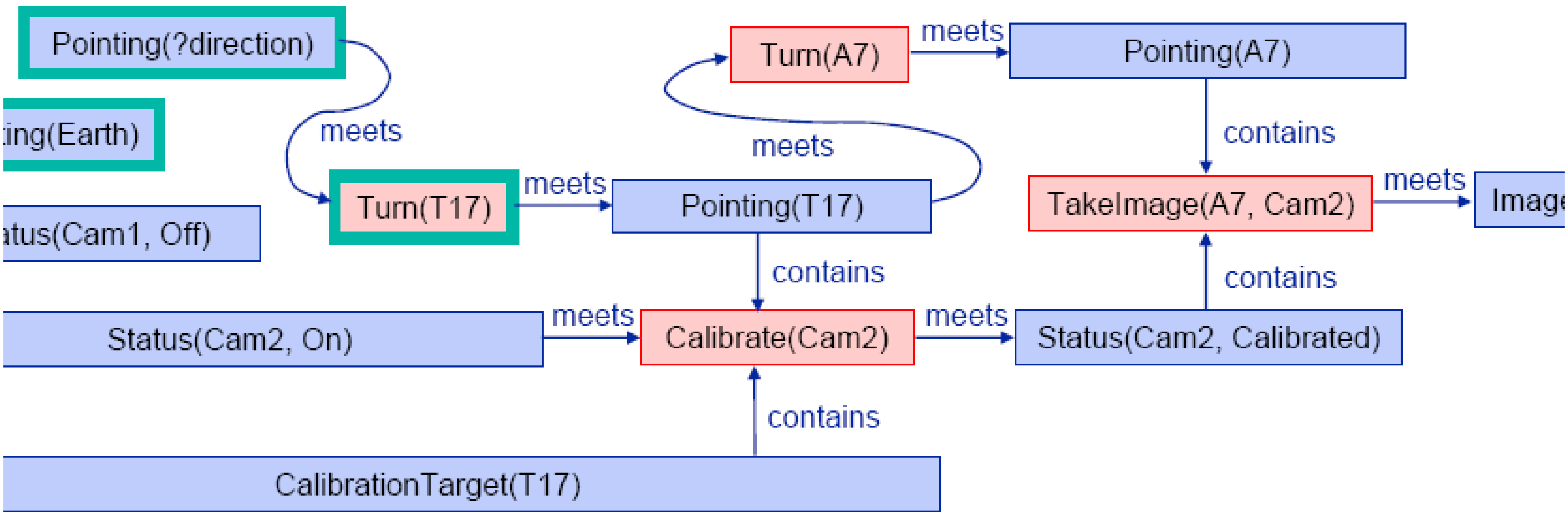
Pointing(?direction)
meets
Turn(A7)
meets
Pointing(A7)
ng(Earth)
before
before
contains
us(Cam1, Off)
Pointing(T17)
TakeImage(A7, Cam2)
meets
Image(
contains
contains
Status(Cam2, On)
meets
Calibrate(Cam2)
meets
Status(Cam2, Calibrated)
contains
CalibrationTarget(T17)

Coalescing



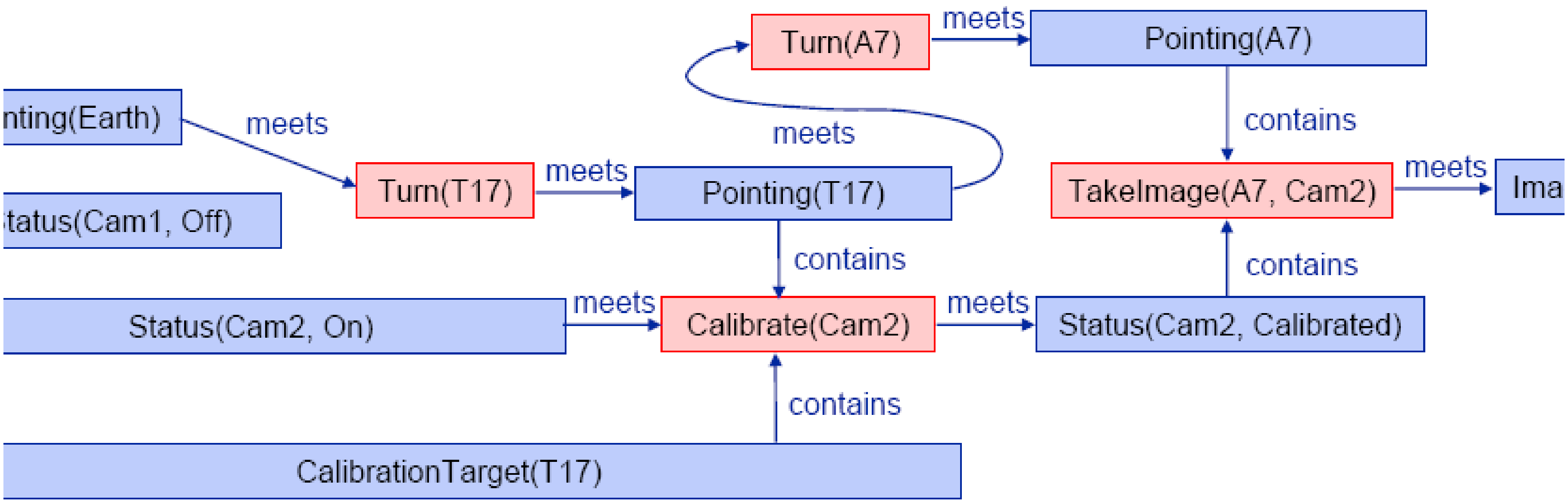
Turn(A7)
meets
Pointing(A7)
ting(Earth)
before
meets
contains
atus(Cam1, Off)
Pointing(T17)
TakeImage(A7, Cam2)
meets
Imag
contains
contains
Status(Cam2, On)
meets
Calibrate(Cam2)
meets
Status(Cam2, Calibrated)
contains
CalibrationTarget(T17)

Expansion



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

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

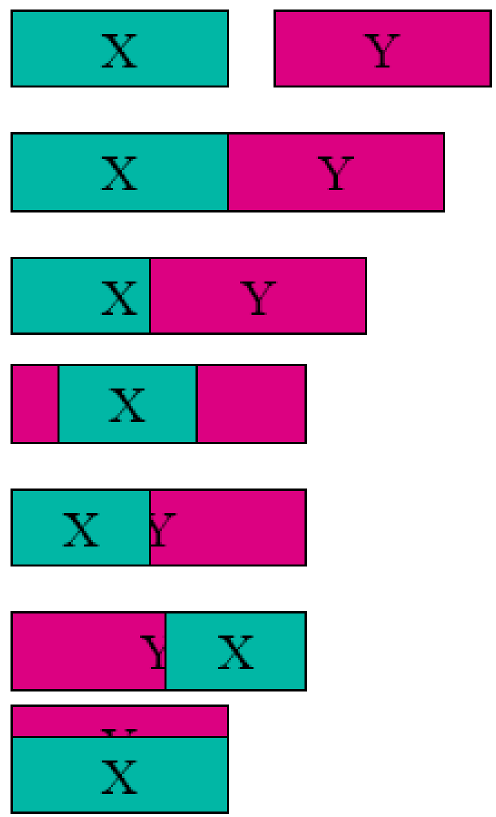
Zeno (Penberthy)	intervals, no CSP
Trains (Allen)	
Descartes (Joslin)	extreme least commitment
IxTeT (Ghallab)	functional rep.
HSTS (Muscettola)	functional rep., activities
EUROPA (Jonsson)	functional rep., activities

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

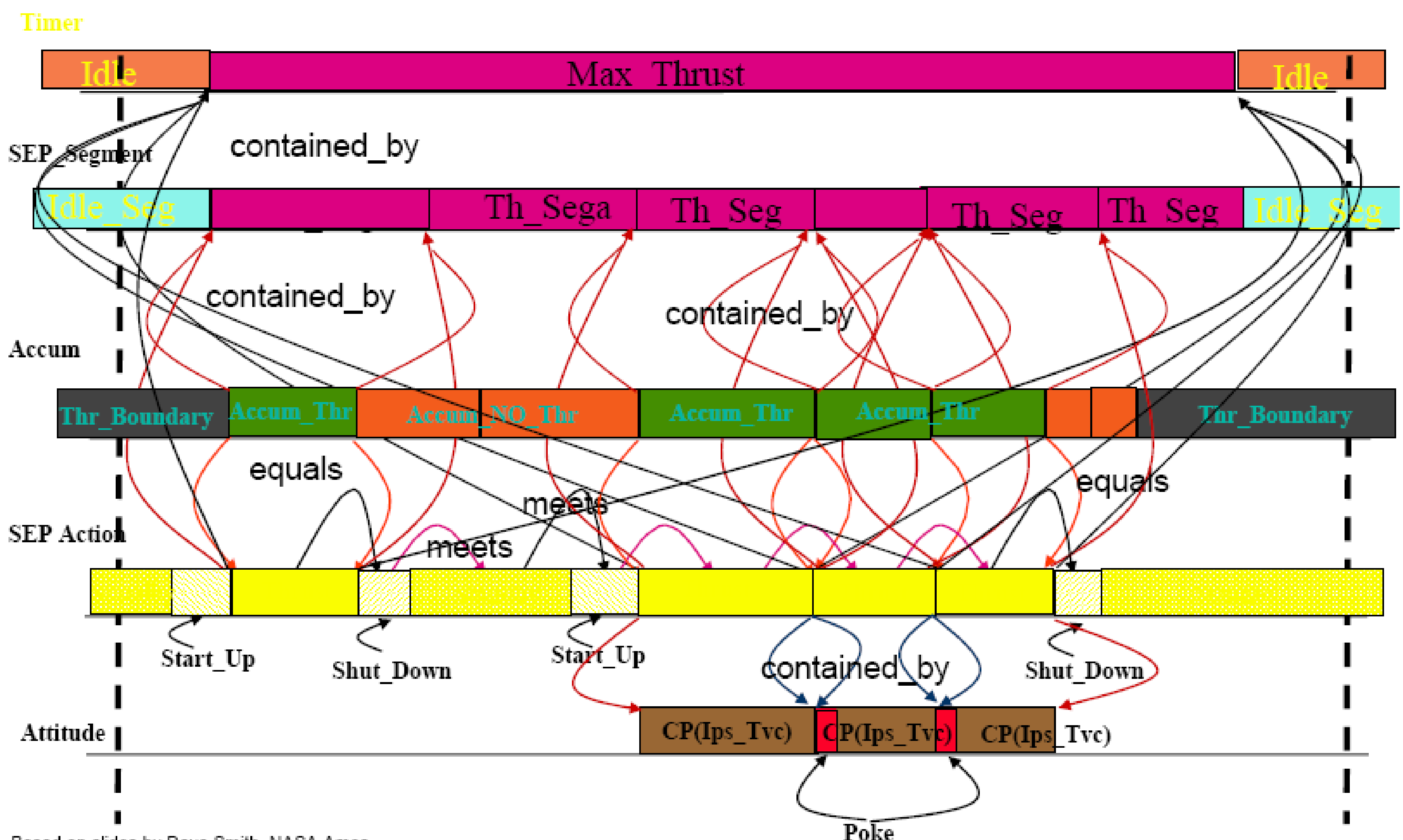
Temporal Constraints

- 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:

$$X^+ - Y^- < [-inf, 0]$$

Metric Constraints

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

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

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

Temporal Constraint Networks

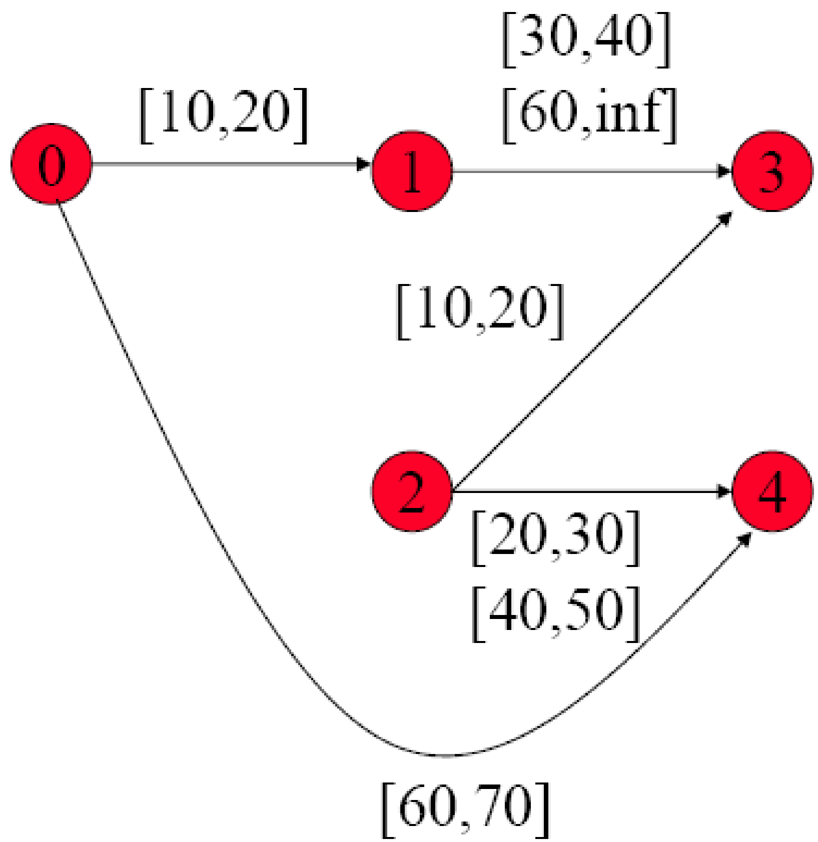
- A set of time points X_i at which events occur.
- Unary constraints

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

- 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

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

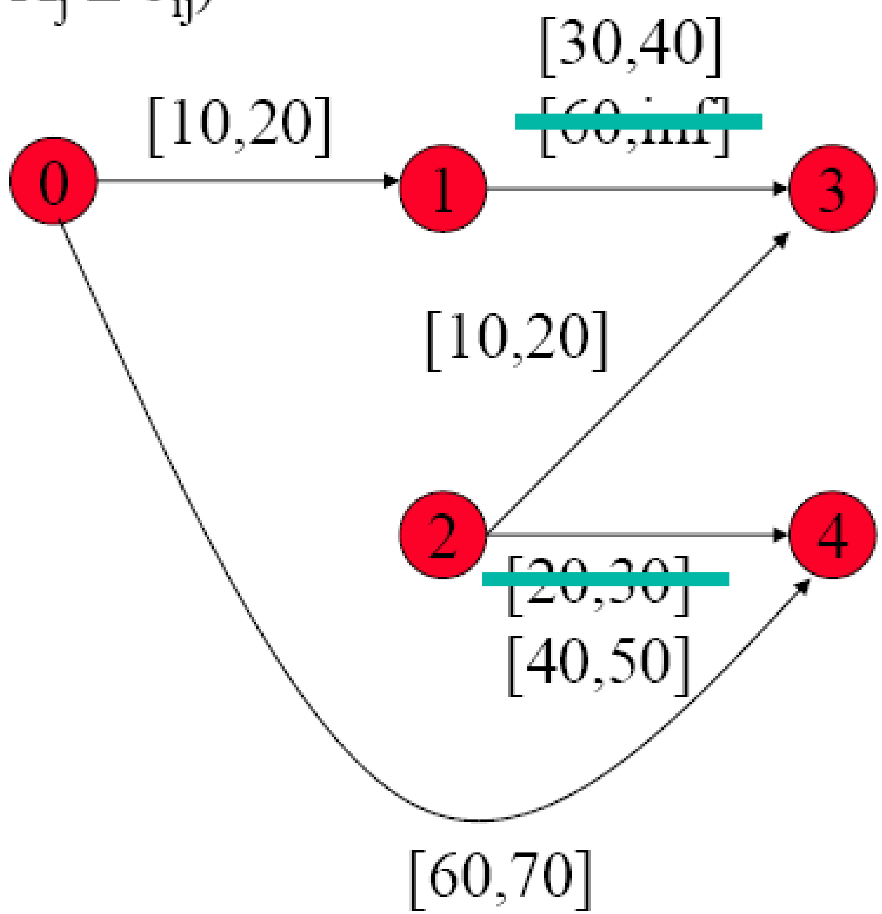
  $(a_0 \leq X_j - X_i \leq b_0)$ ~~or $(a_1 \leq X_j - X_i \leq b_1)$ or . . .~~

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

Can't represent:
- Disjoint activities

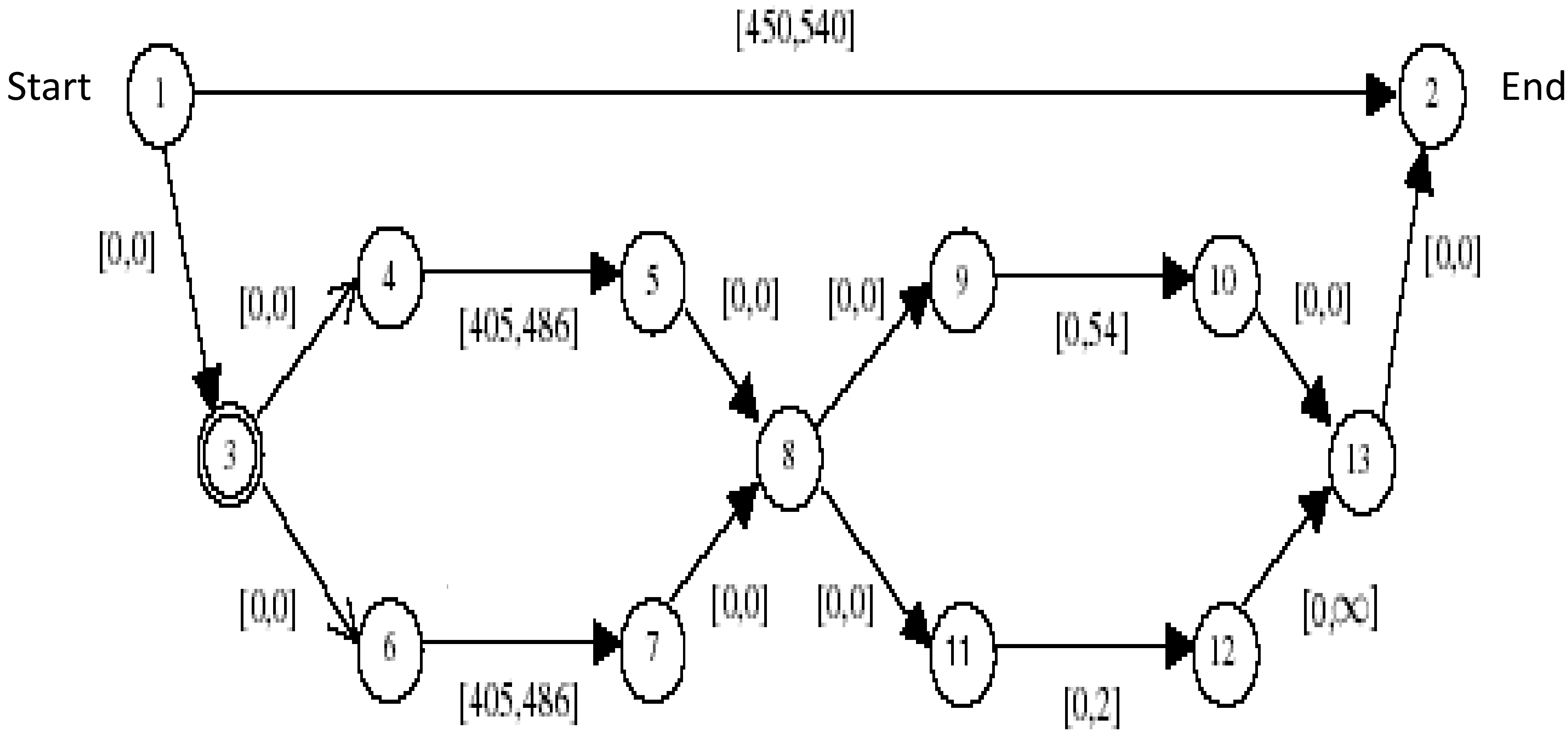
Simple Temporal Networks

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



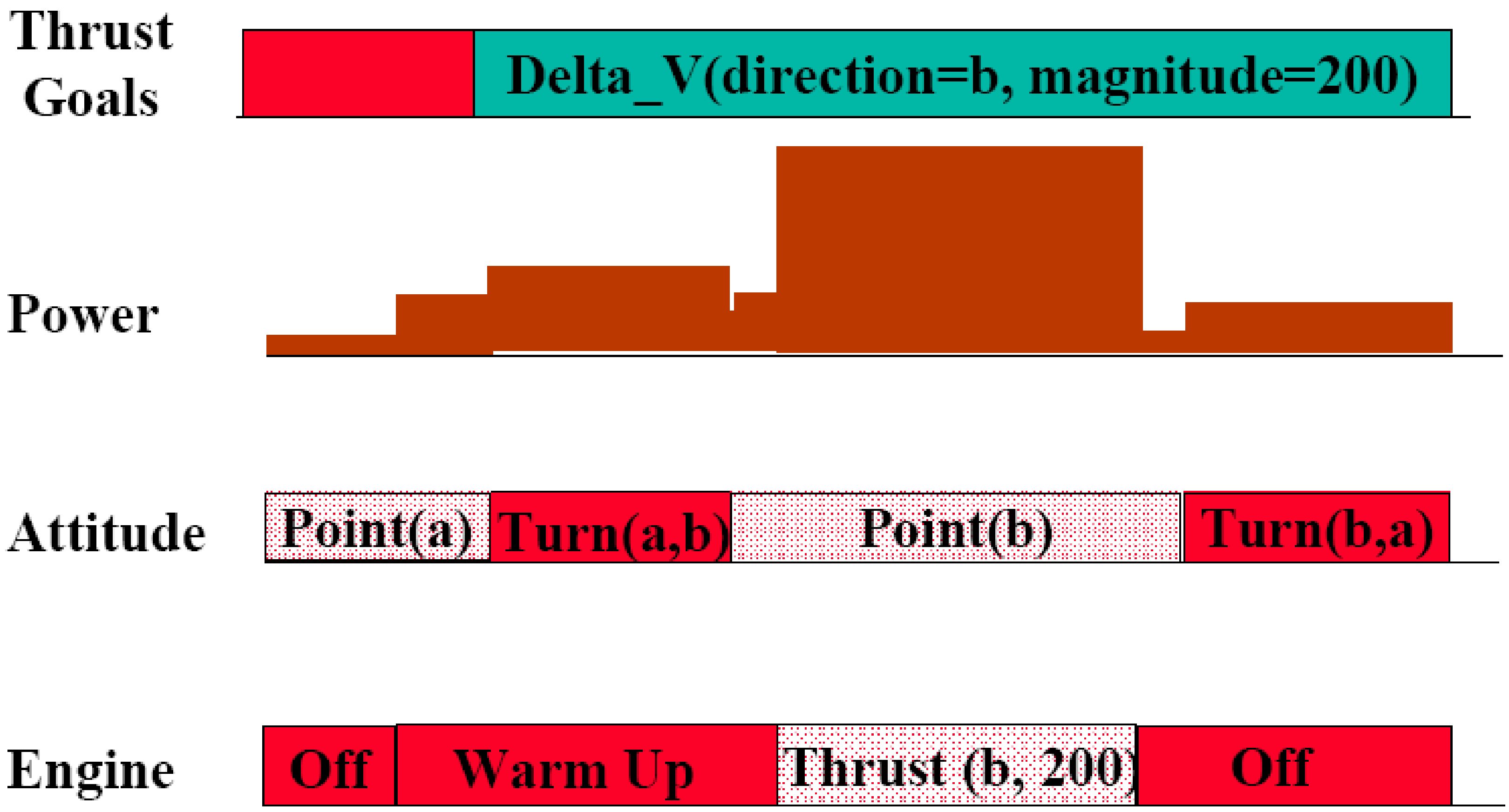
Based on slides by Dave Smith, NASA Ames

STN example



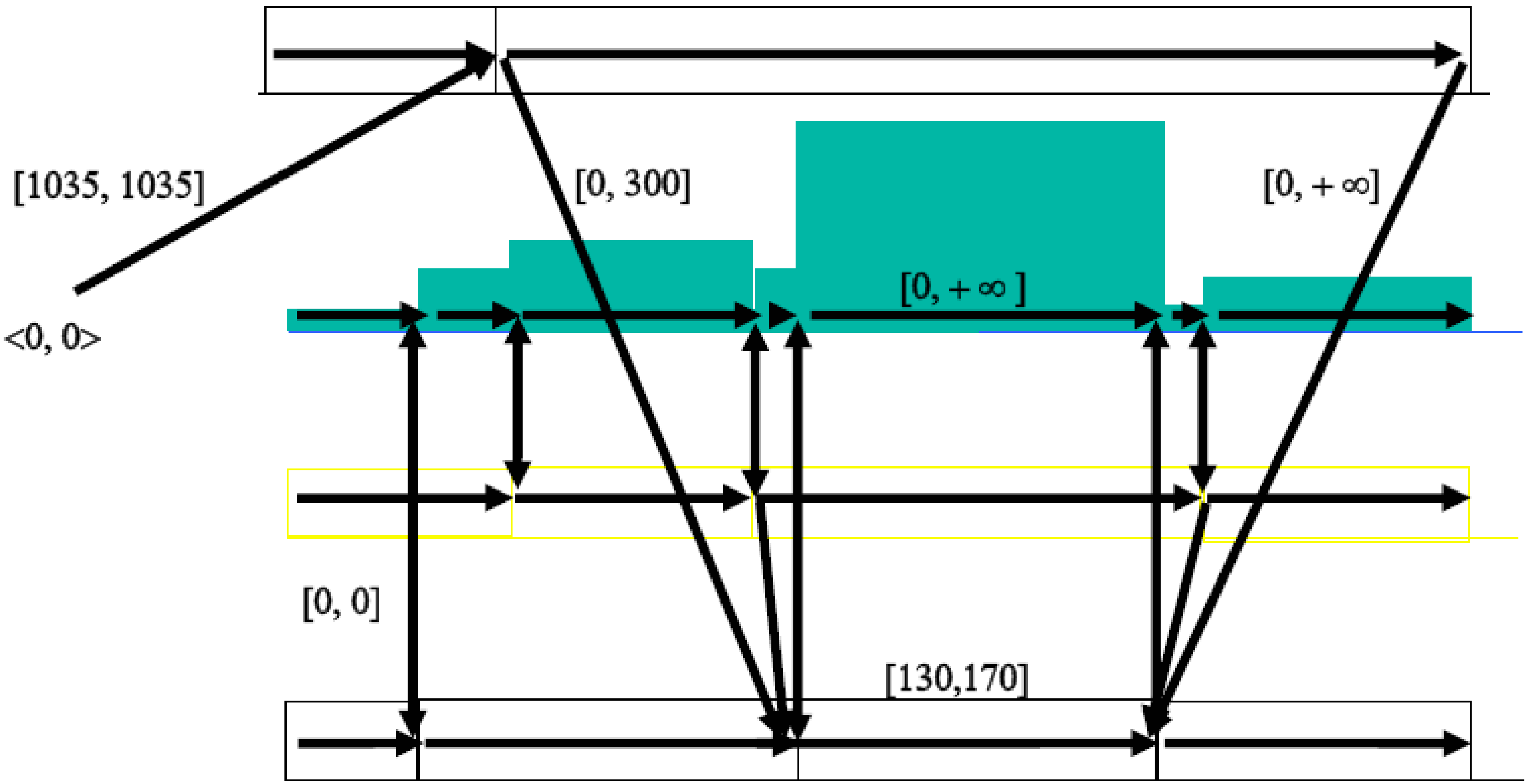
Start
End
[450,540]
1
2
3
4
5
6
7
8
9
10
11
12
13
[0,0]
[0,0]
[405,486]
[0,0]
[0,0]
[0,54]
[0,0]
[0,0]
[0,0]
[405,486]
[0,0]
[0,0]
[0,2]
[0,∞]

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

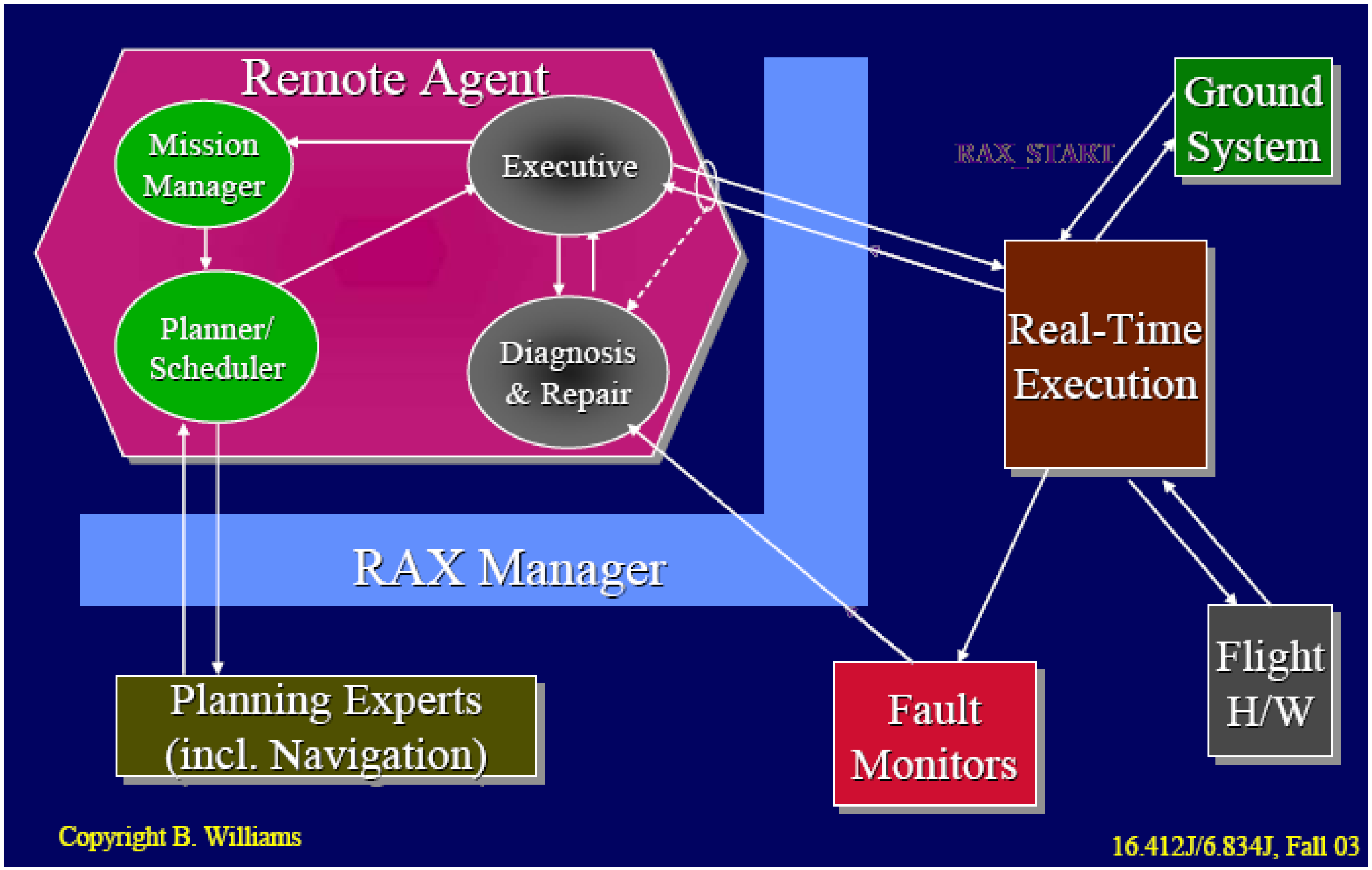
May 17-18th experiment

- 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

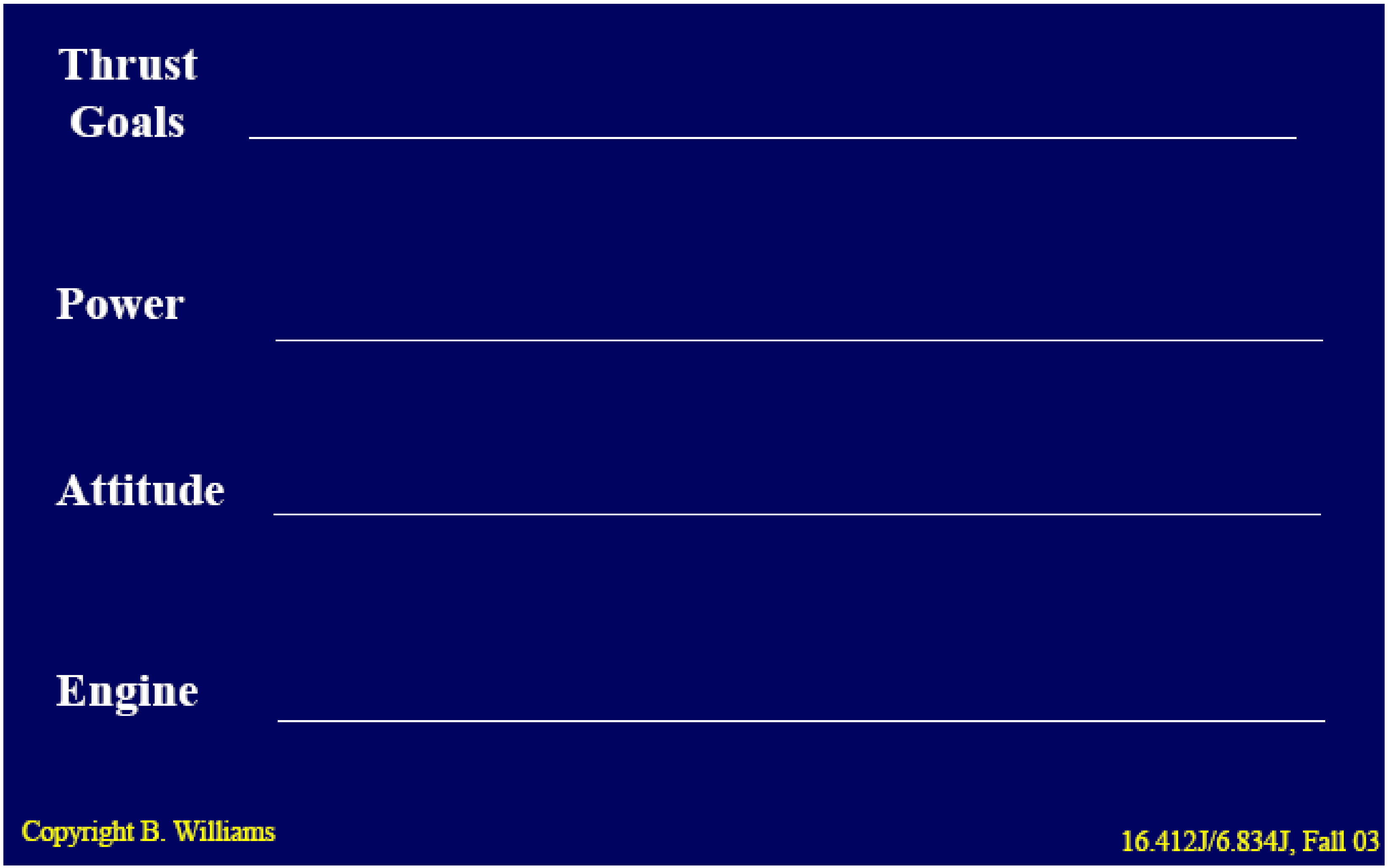
May 21th experiment.

- 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

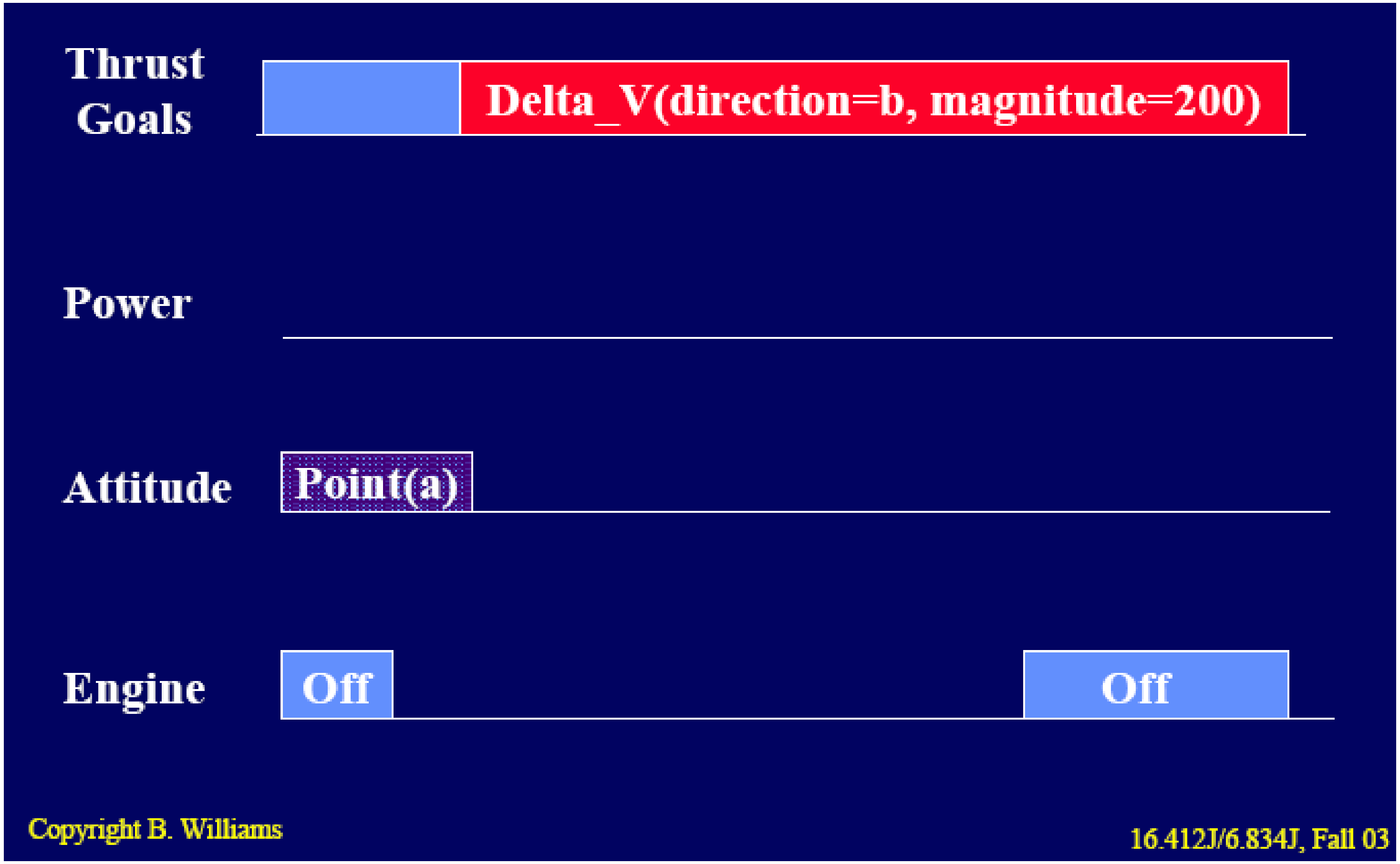


Remote Agent



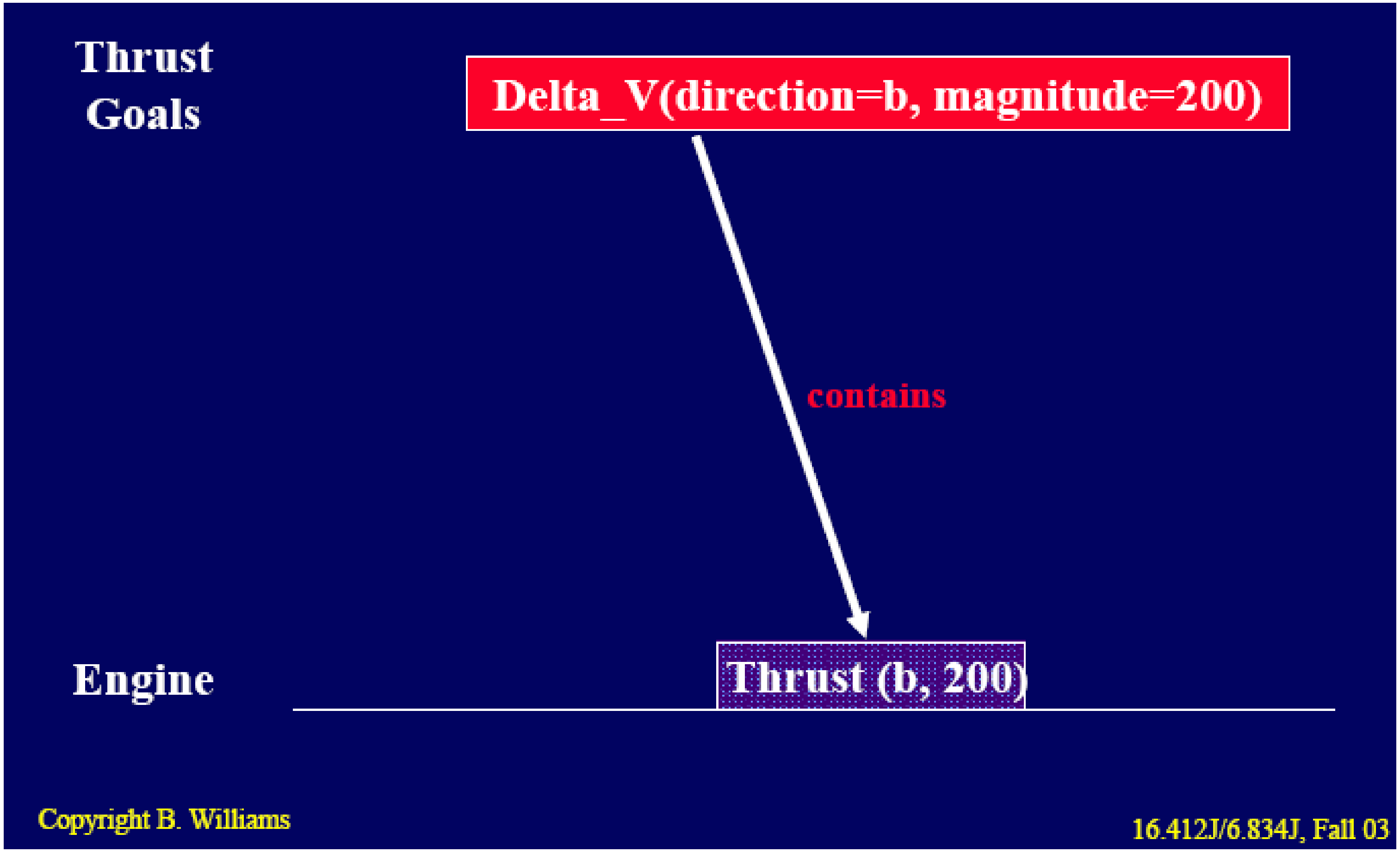
Remote Agent

- Mission Manager



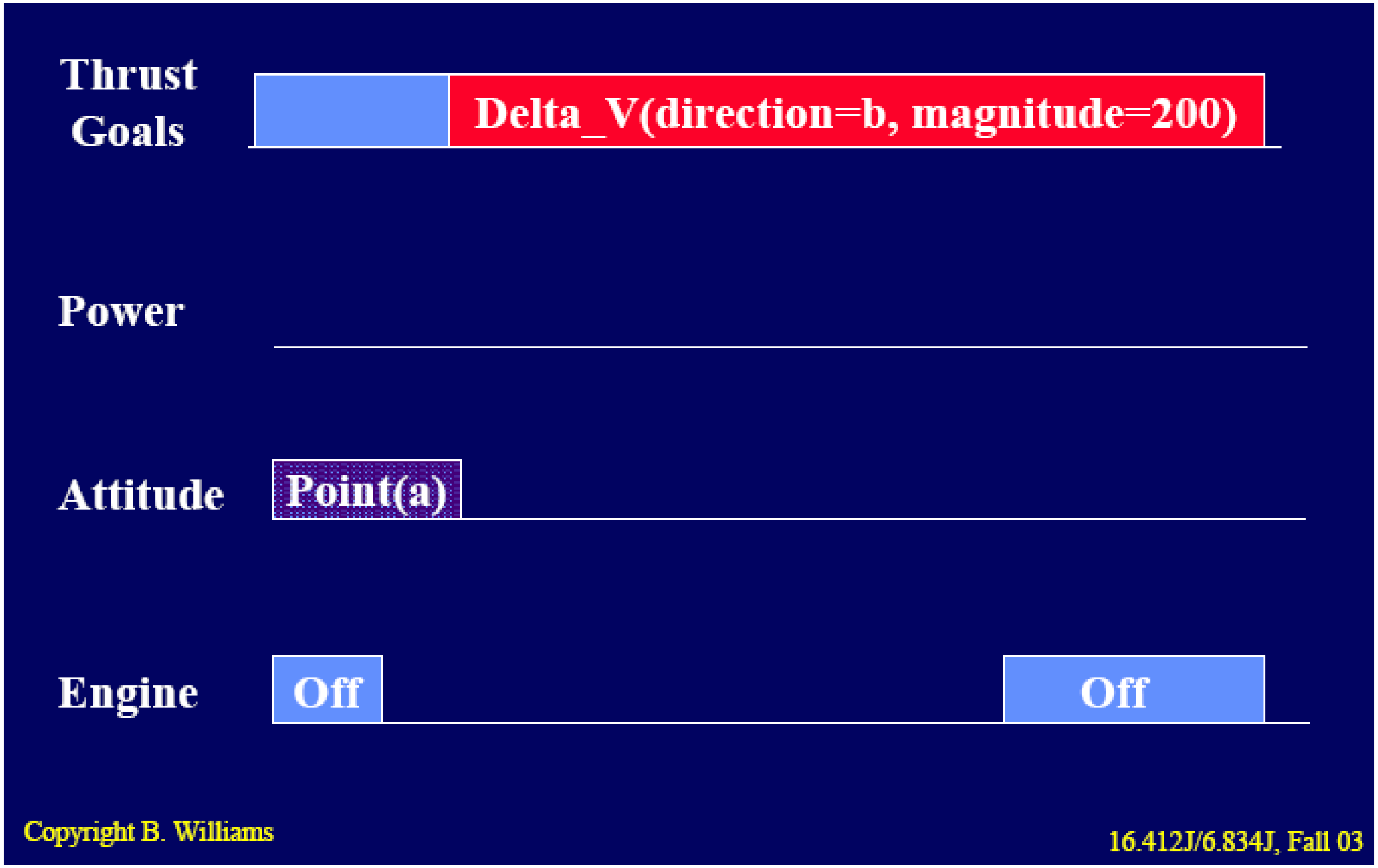
Remote Agent

- Constraints:



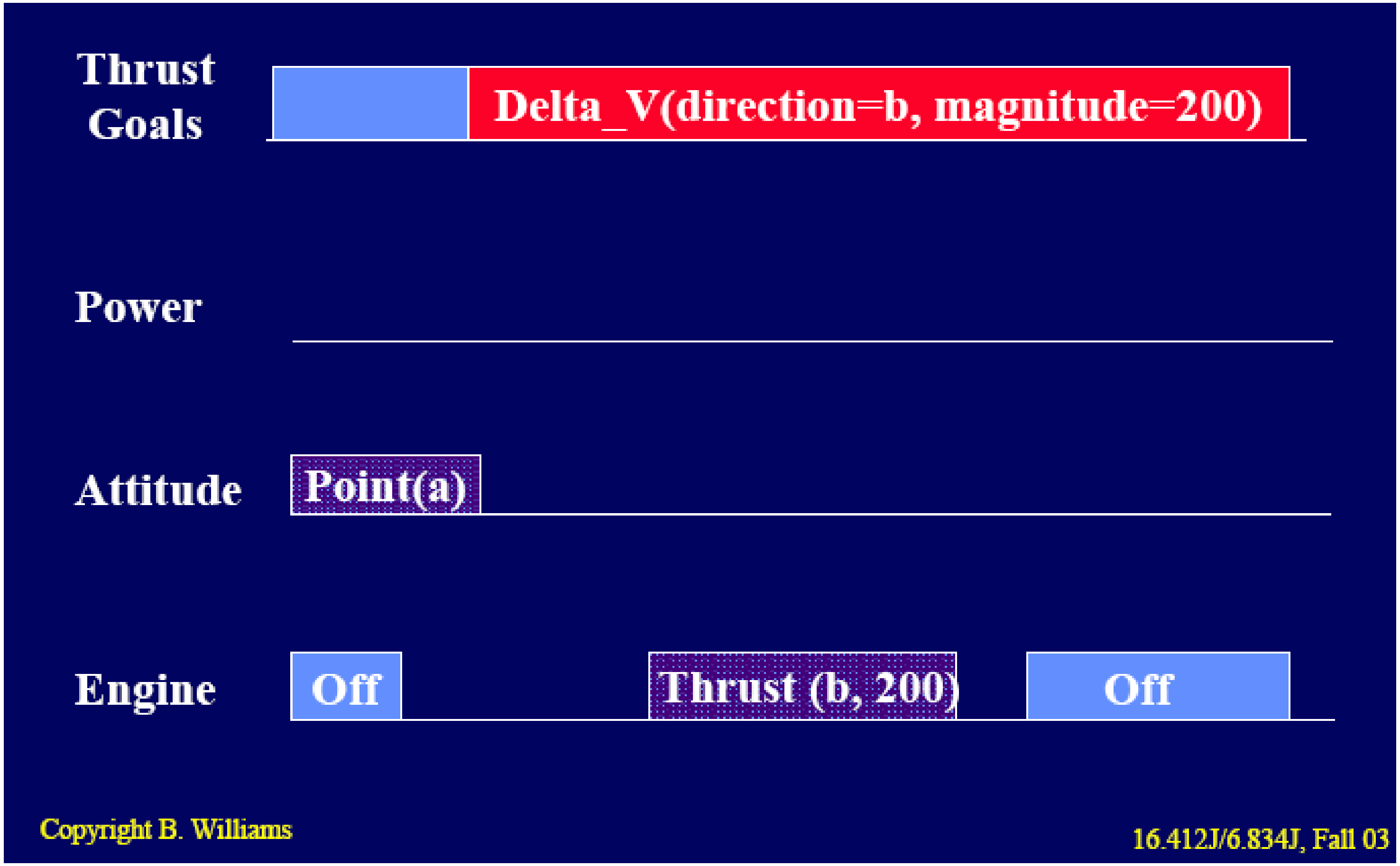
Remote Agent

- Planner starts



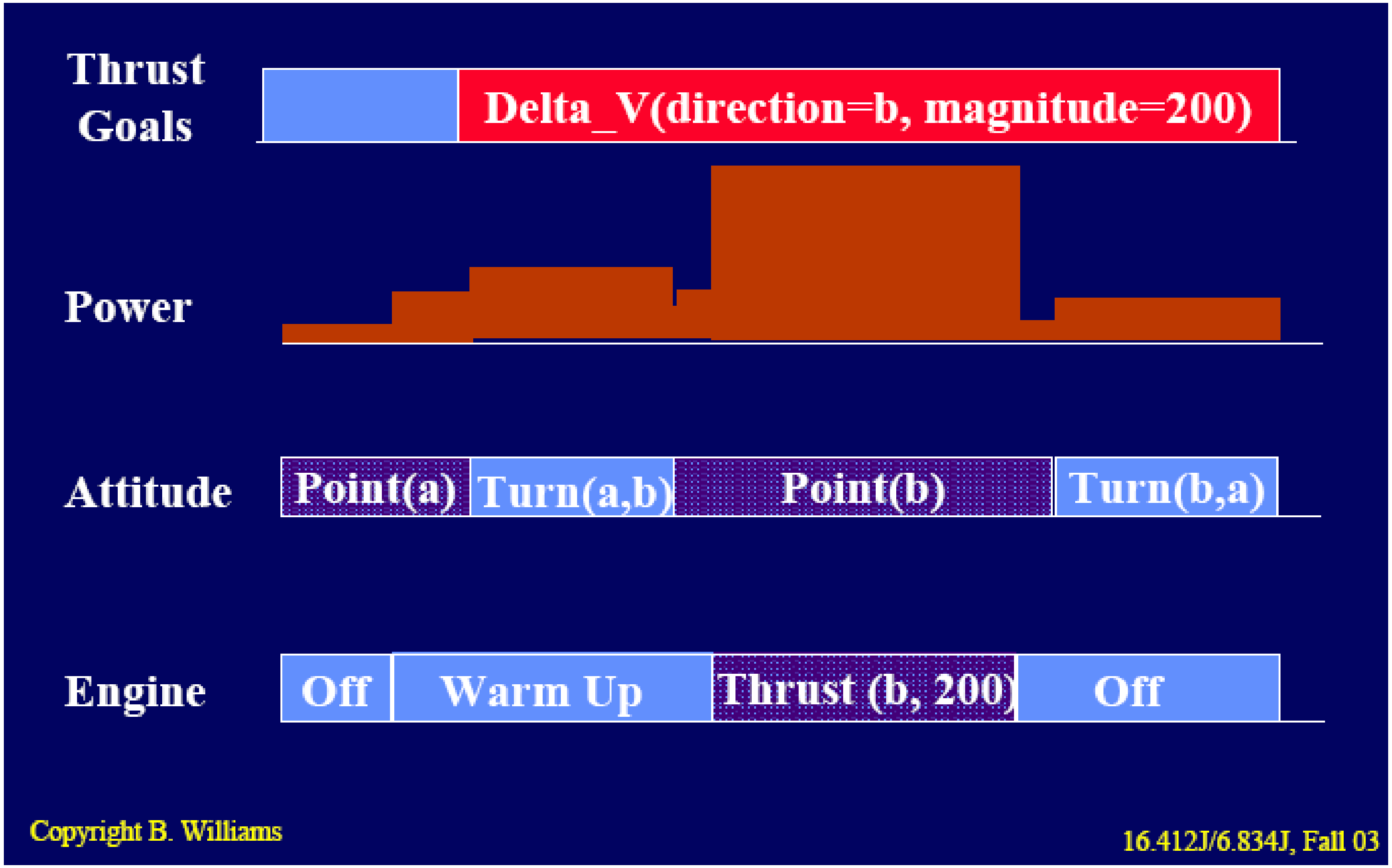
Remote Agent

- Planning



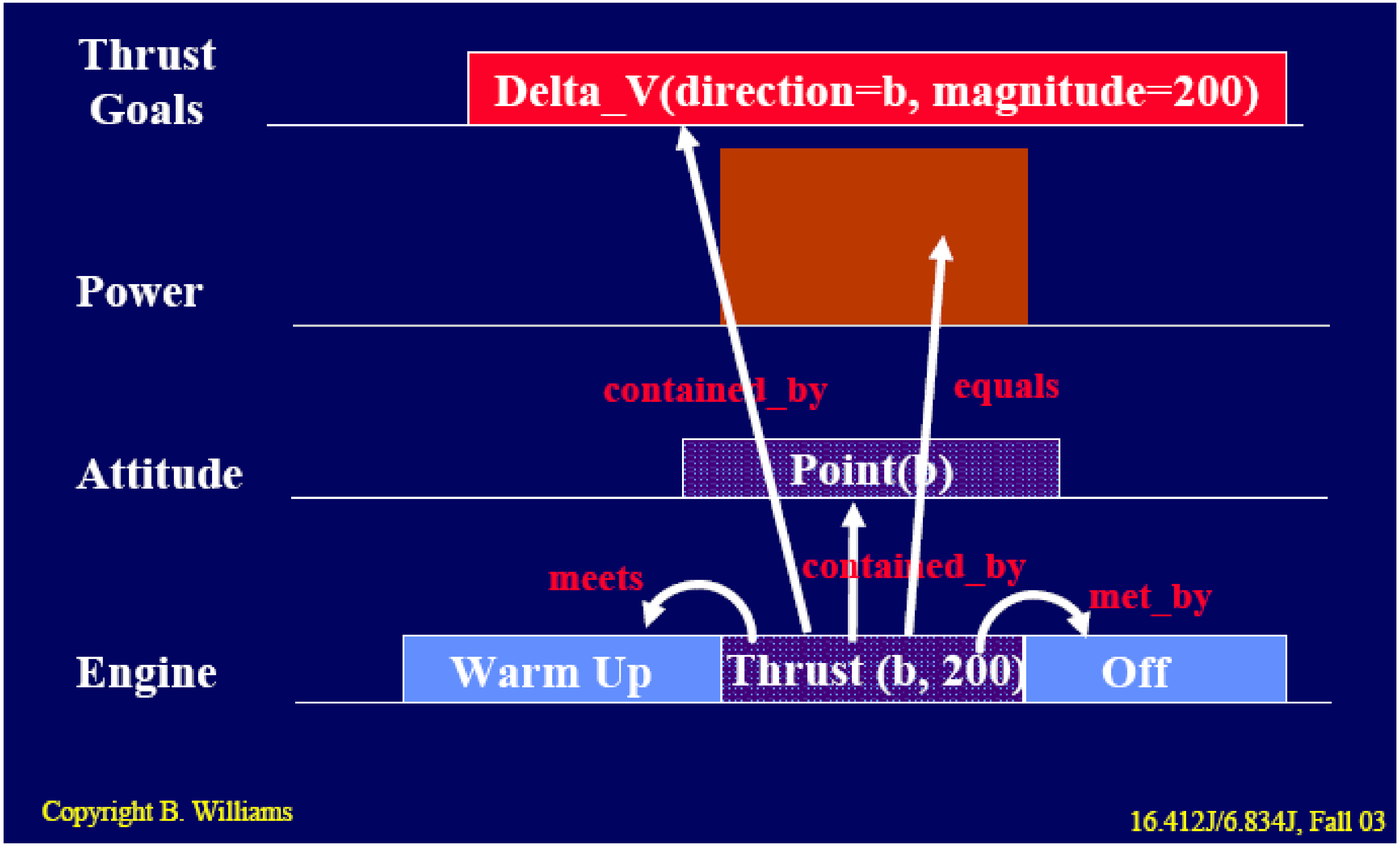
Remote Agent

- Final Plan



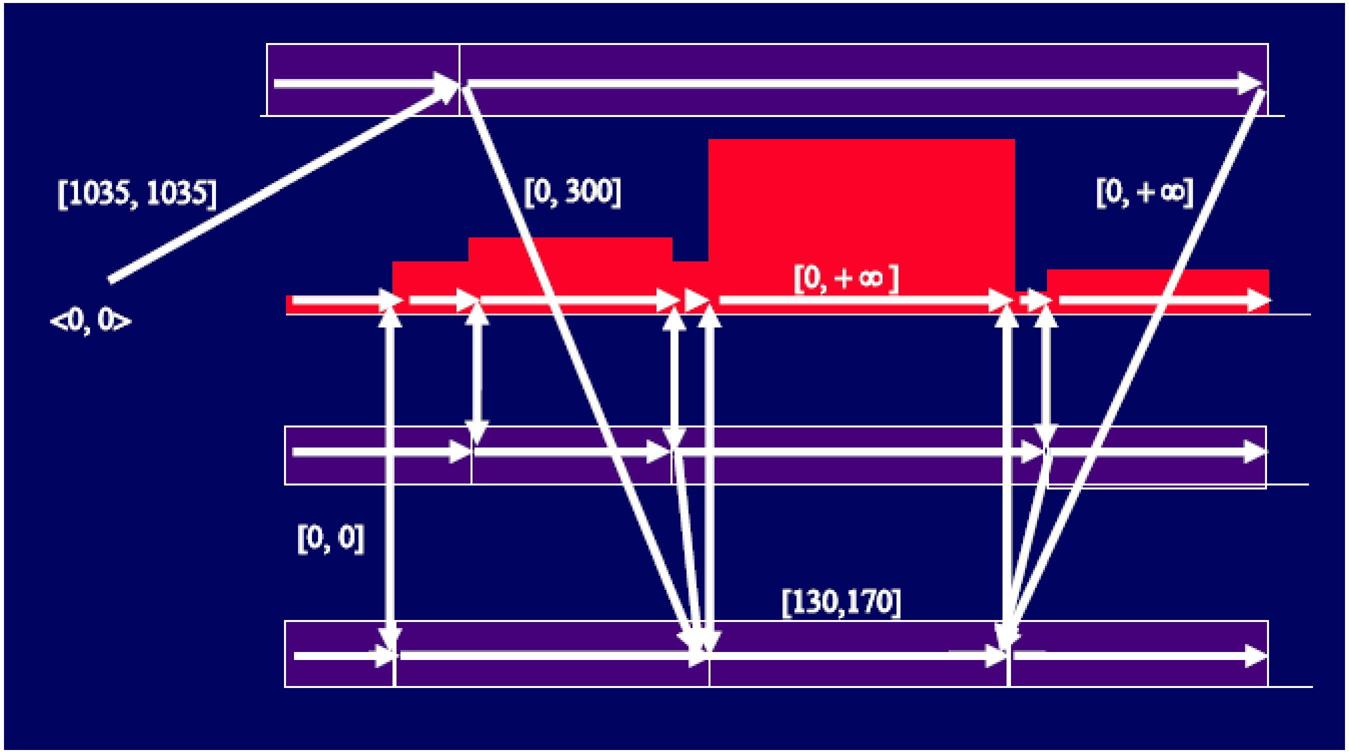
Remote Agent

- Constraints



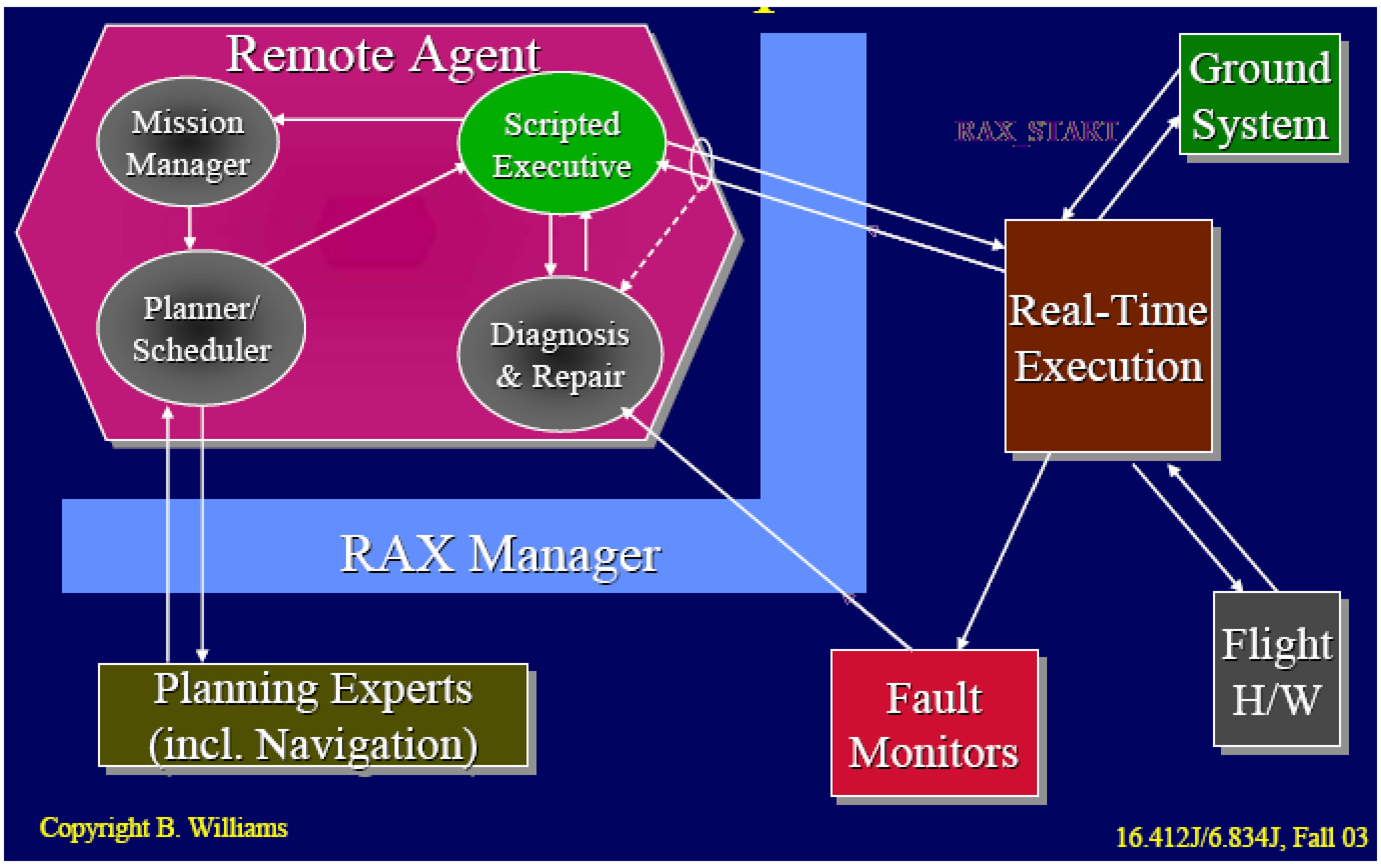
Remote Agent

- Flexible Temporal Plan through least commitment



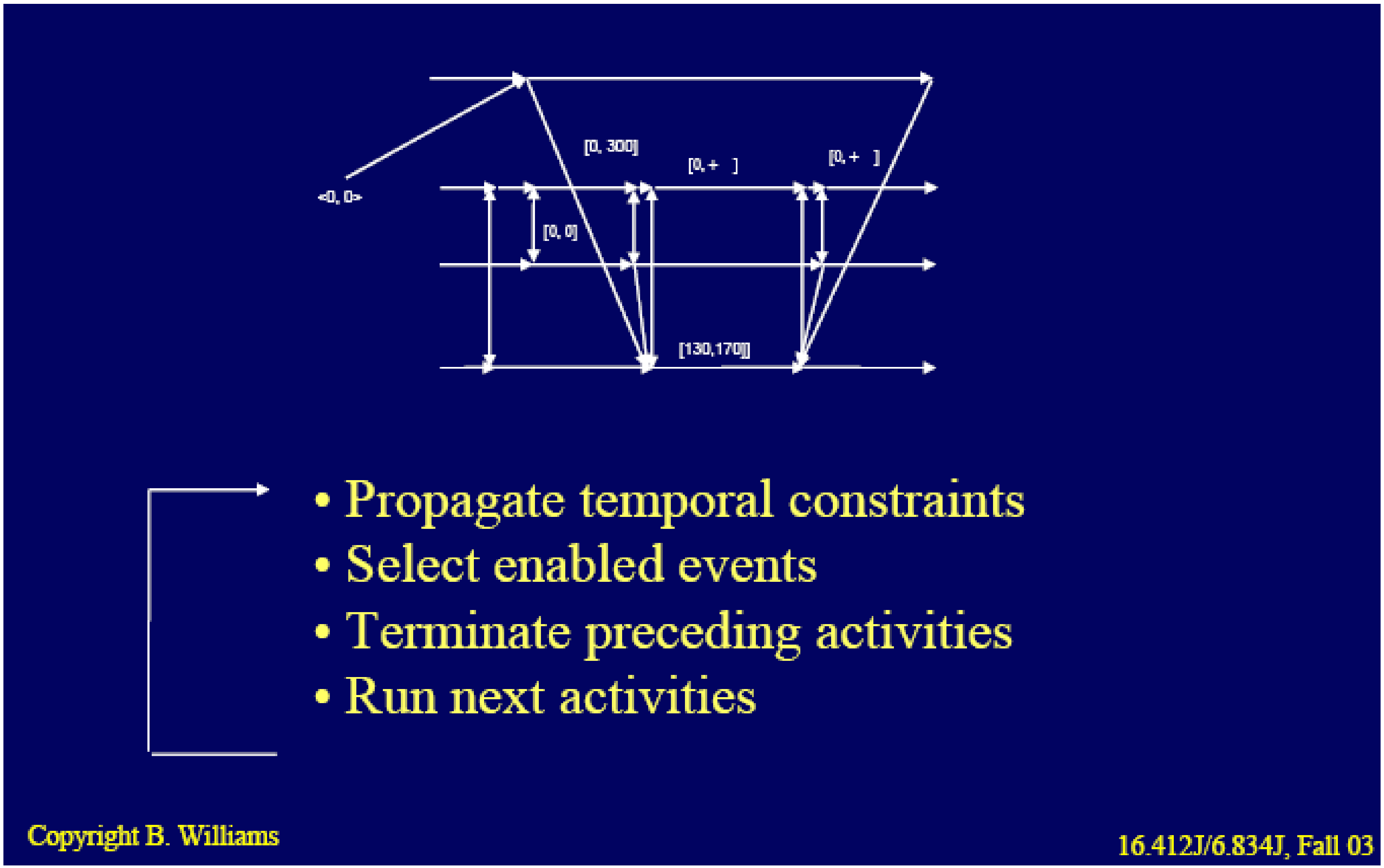
Remote Agent

- Executive system dispatch tasks



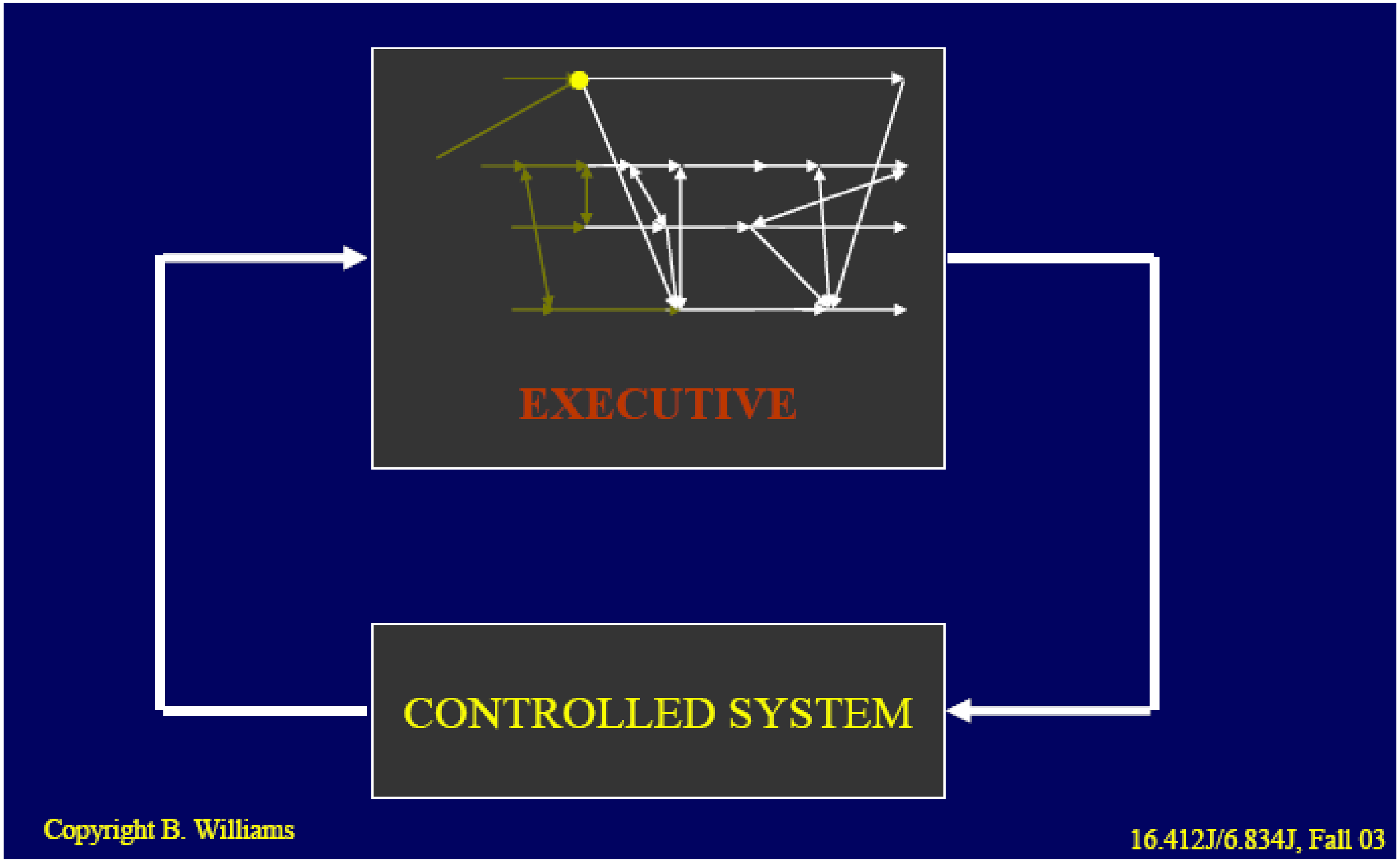
Remote Agent

- Executing Flexible Plans



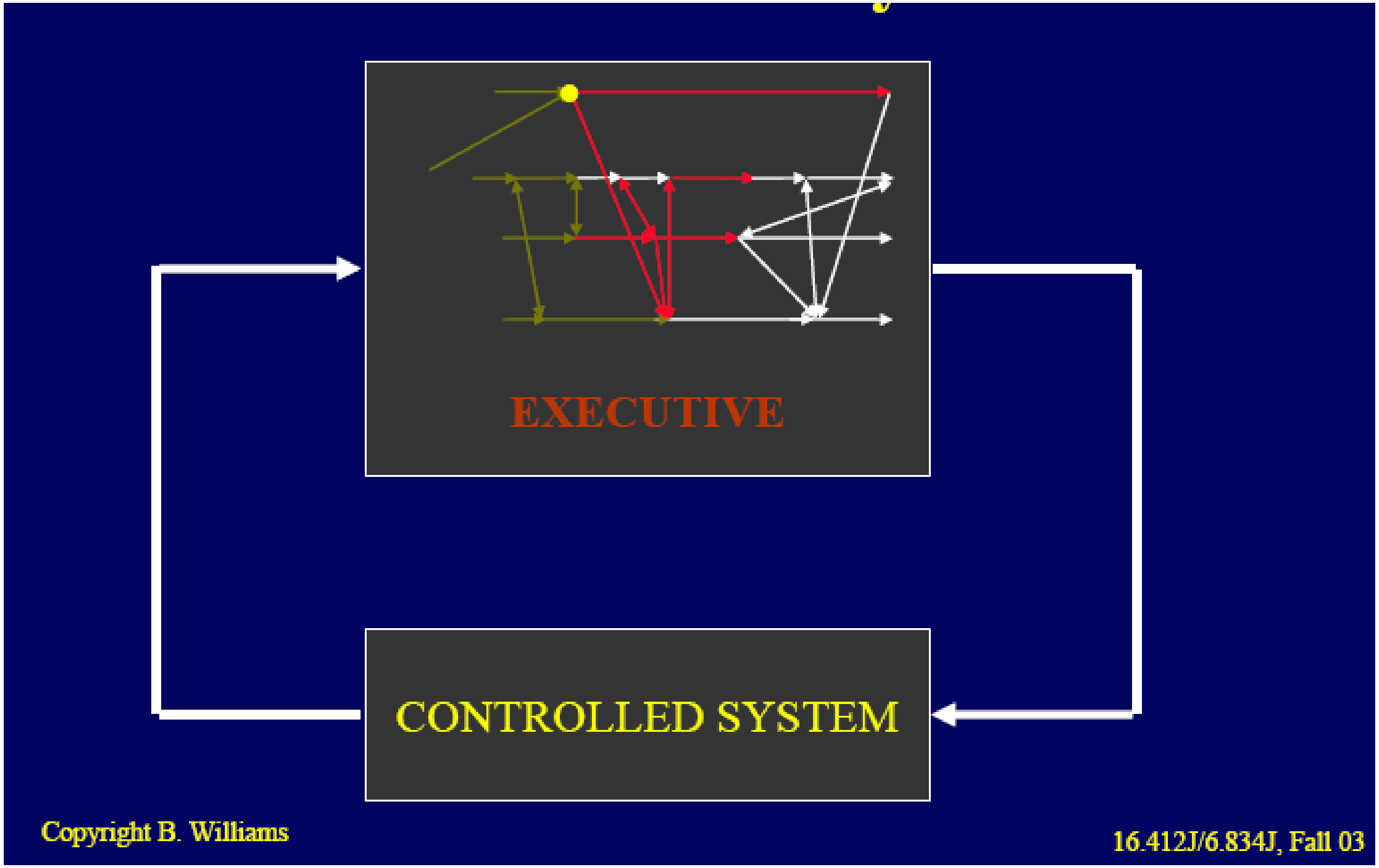
Remote Agent

- Constraint propagation can be costly



Remote Agent

- Constraint Propagation can be costly



Remote Agent

- Solution: compile temporal constraints to an efficient network