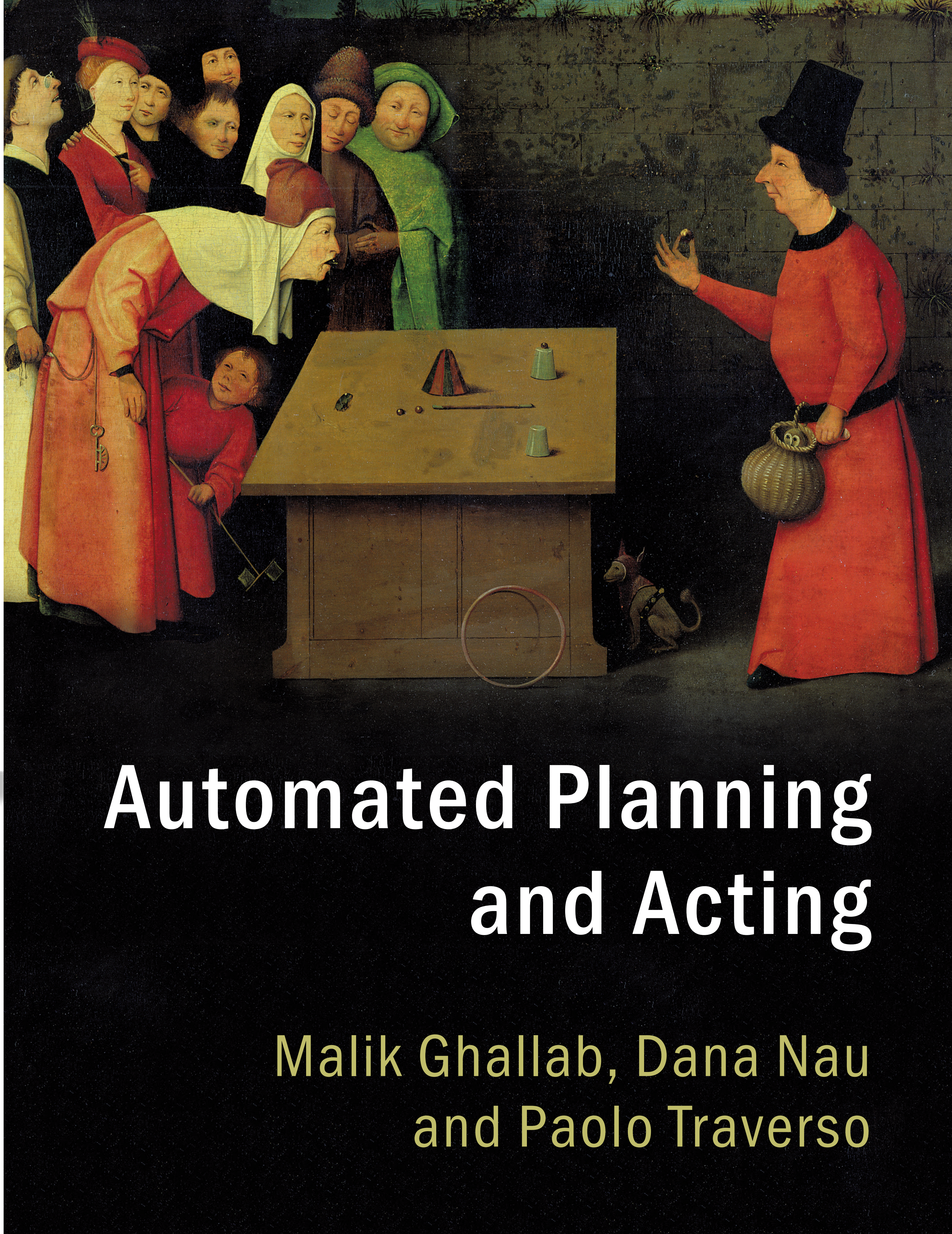
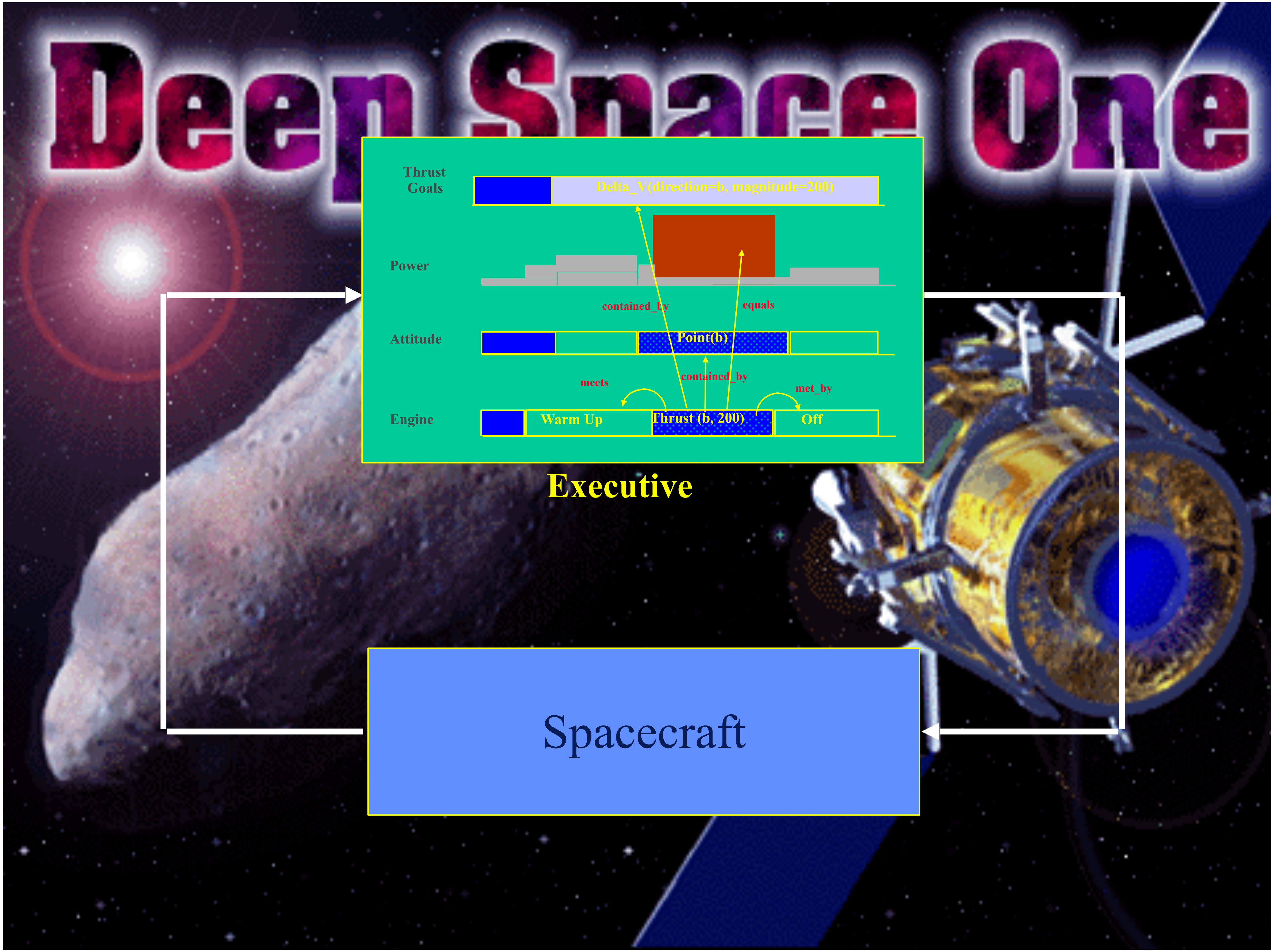
Deliberation with Temporal Models



IJCAI Tutorial
NY, July 2016

Planning and Acting Success Stories

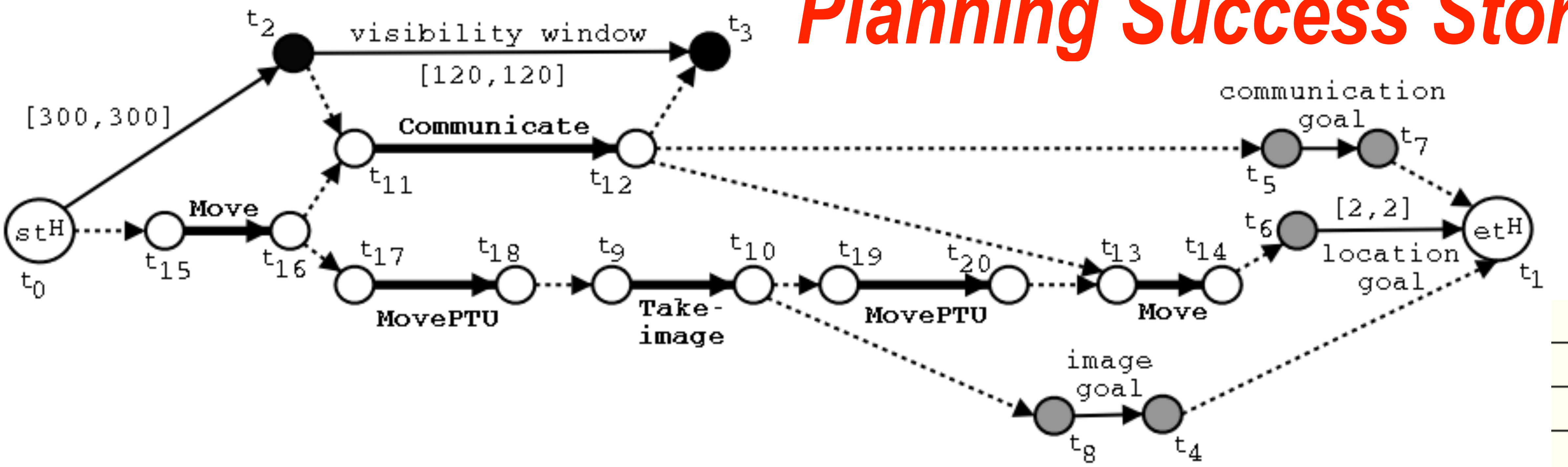


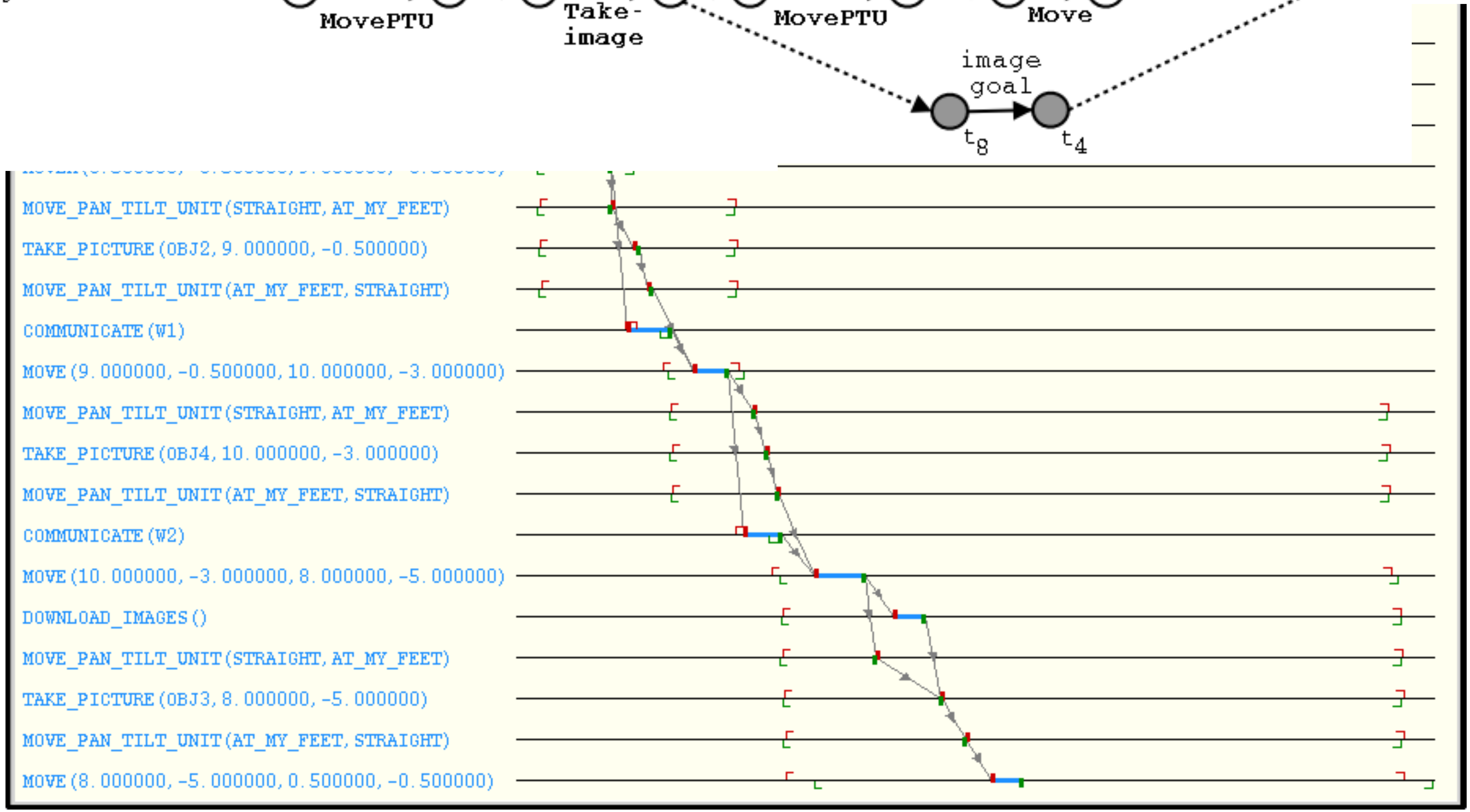
[RAX/PS, NASA AMES & JPL]

Planning Success Stories

[*IxTeT, LAAS*]

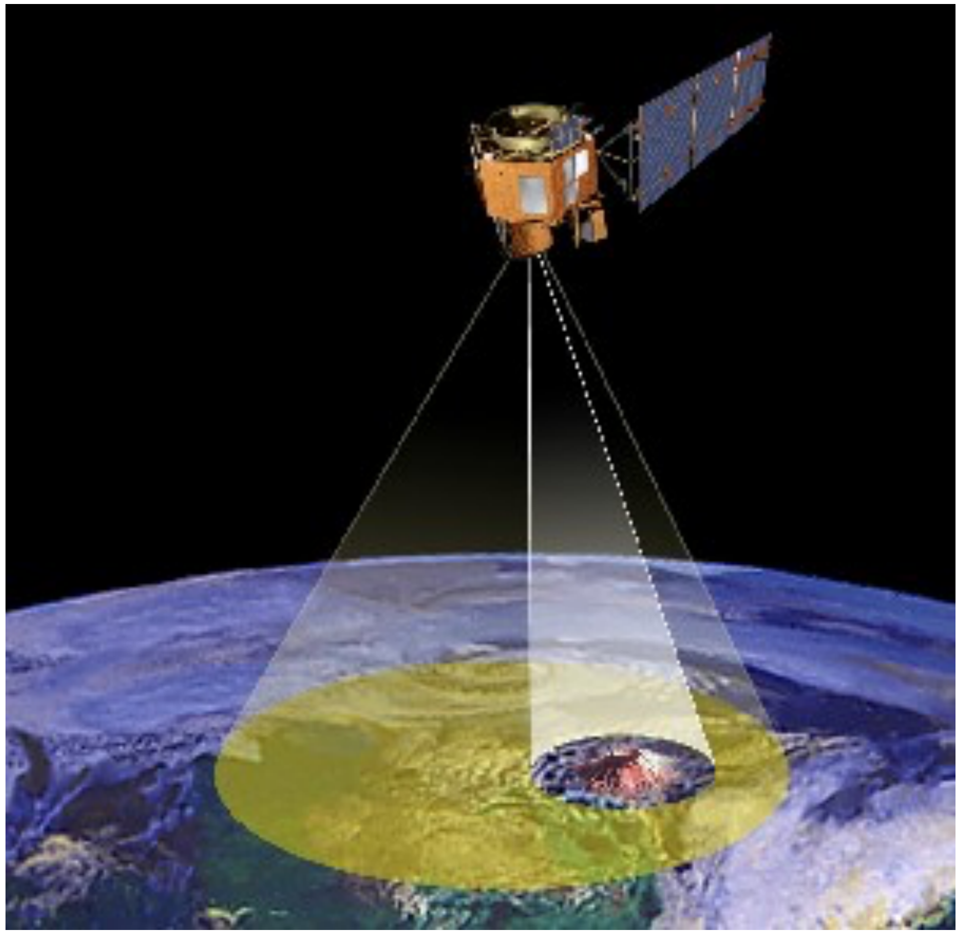
Planning Success Stories





[IxTeT, LAAS]

Planning Success Stories



[*Casper, JPL*]

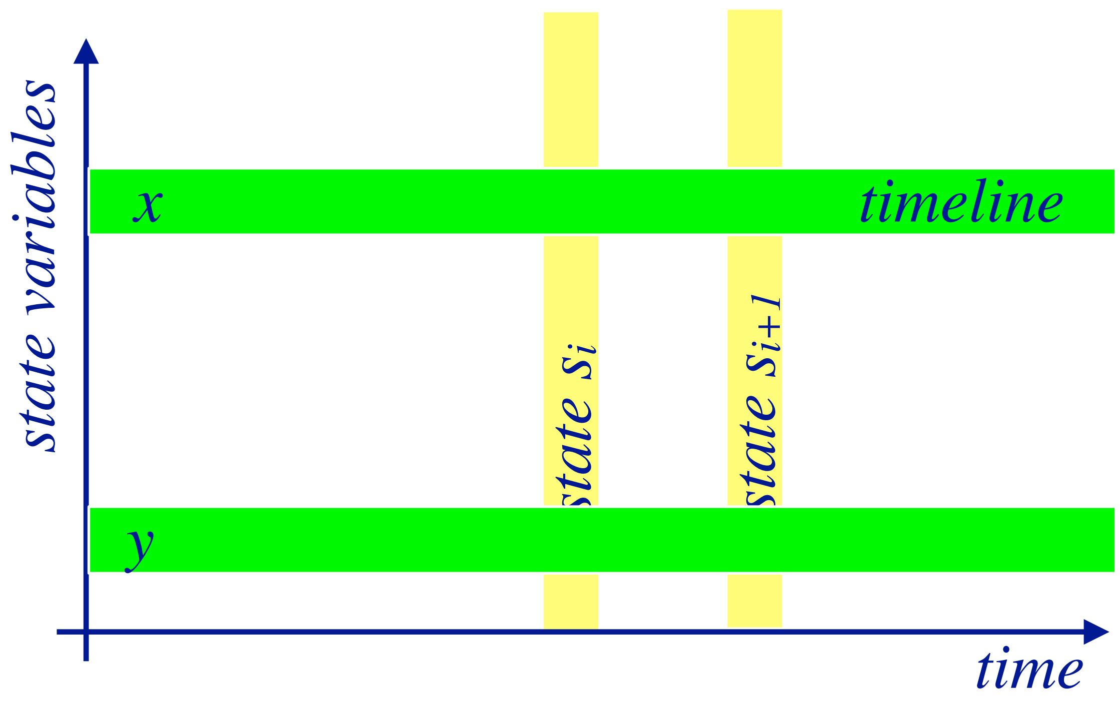
[*T-ReX, MBARI*]

Common point to these success stories:
explicit representation of time

Motivations for Temporal Models

- *Duration* of actions
- *Delayed effects*, conditions, and resources borrowed or consumed at various moments along an action duration
- *Timed goals* with relative or absolute temporal constraints
- *Exogenous events* expected to occur in the future time
- *Maintenance actions:* maintain a property (≠ changing a value), e.g., tracking a moving target, keeping a spring latch in position
- *Concurrency* of actions with interacting and joint effects
- *Delayed commitment*: instantiation at acting time

Temporal Models



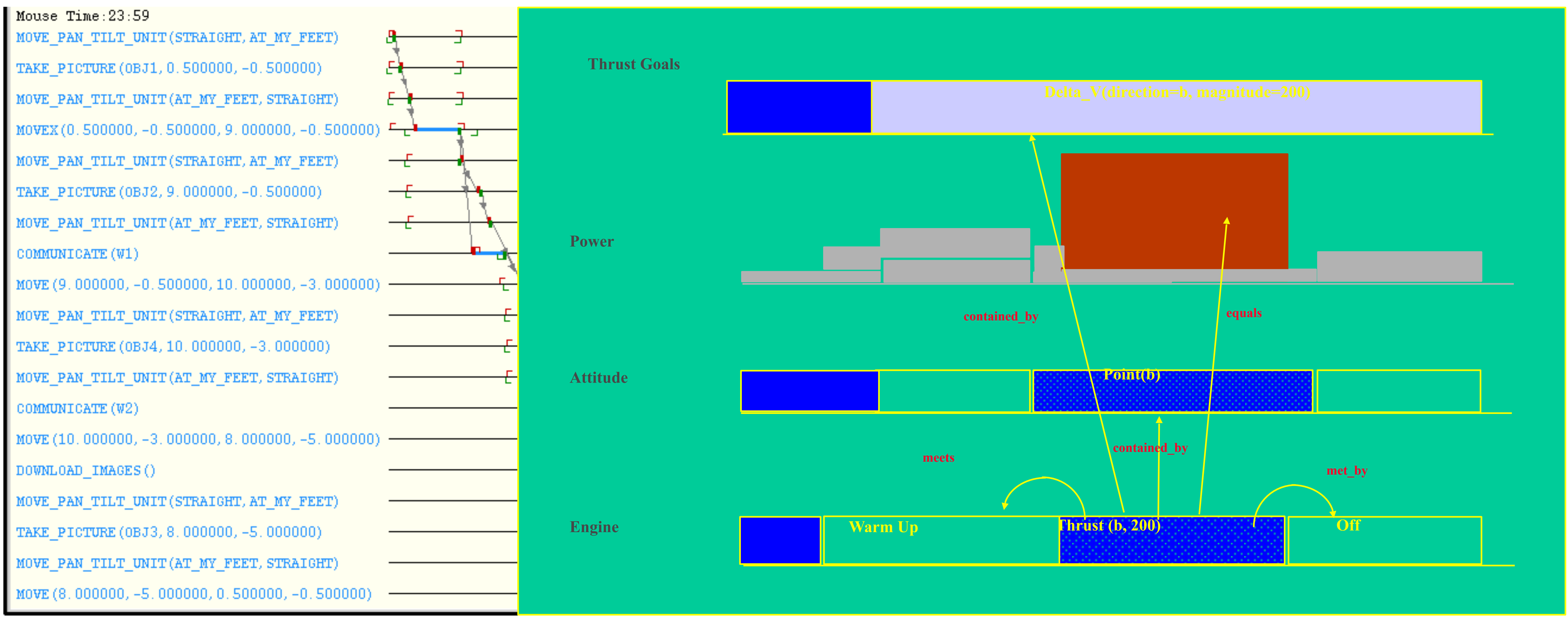
Motivations for Temporal Models

	States	Timelines
Duration of actions	√	√
Delayed effects	√	√
Timed goals	√	√
Exogenous events	~	√
Maintenance actions	√	√
Concurrency	–	√
Delayed commitment	–	√

Timeline

Timeline

- A set of constraints on state variables and events
- Reflects *predicted* actions and events
- Timeline planning akin to constraint-based planning



Outline

- ✓ Introduction
- Representation
  - Timelines
  - Actions and tasks
  - Chronicles
- Temporal planning
- Consistency and controllability
- Acting with executable primitives
- Acting with atemporal refinement
- Conclusion

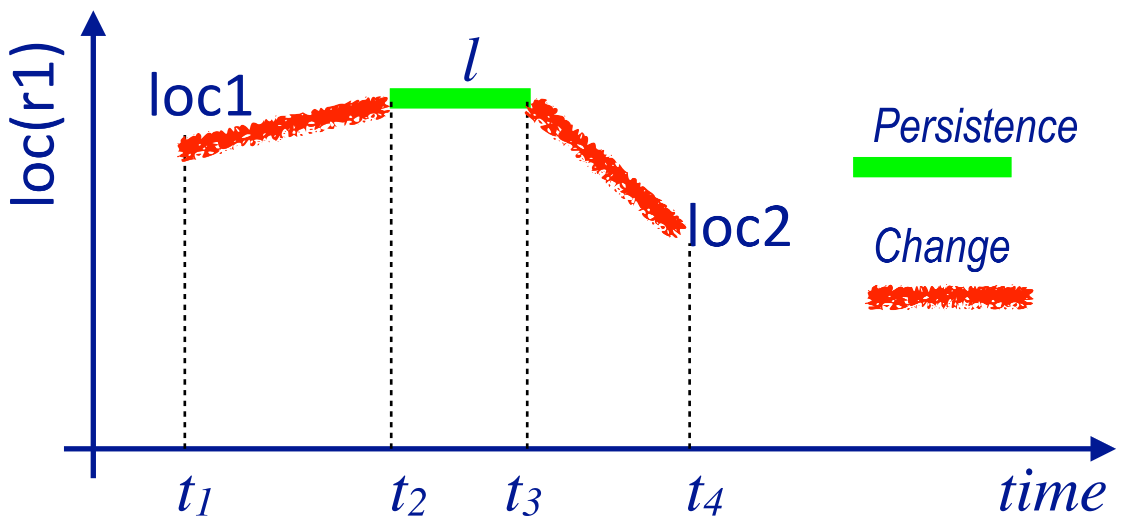
Representation

- Quantitative discrete model of time
  - variables referring to time points
  - simple constraints $d \leq t' - t \leq d'$
- Temporal assertions
  - *persistance* over an interval $[t_1, t_2]x = v$
  - *change* over an interval $[t_1, t_2]x{:}(v_1, v_2)$

Timeline

Partially predicted evolution of a state variable: a pair $(\mathcal{T}, \mathcal{C})$

- $\mathcal{T}$: temporal assertions
- $\mathcal{C}$: contraints



$$[t_1, t_2]\text{loc(r1)} : (\text{loc1}, l)$$
$$[t_2, t_3]\text{loc(r1)} = l$$
$$[t_3, t_4]\text{loc(r1)} : (l, \text{loc2})$$
$$t_1 < t_2 < t_3 < t_4$$
$$l \neq \text{loc1}$$
$$l \neq \text{loc2}$$

To restrict the value of loc(r1) in $[t_1, t_2]$

$$[t_1, t_1 + 1]\ \text{loc(r1):(loc1,route)}$$
$$[t_2 - 1, t_2]\ \text{loc(r1):(route}, l)$$
$$[t_1 + 1, t_2 - 1]\ \text{loc(r1)} = \text{route}$$

Consistent and Secure Timeline

- A ground instance of $(\mathcal{T}, \mathcal{C})$ is consistent if it satisfies $\mathcal{C}$ and no state variable in $\mathcal{T}$ has more than on value at the same time
- $(\mathcal{T}, \mathcal{C})$ is *consistent* if it has a consistent ground instance
- $(\mathcal{T}, \mathcal{C})$ is *secure* if it is consistent and every ground instance that satisfies $\mathcal{C}$ is consistent

$$[t_1, t_2]\mathsf{loc}(r) = \mathsf{loc1}$$
$$[t_2, t_3]\mathsf{loc}(r) : (\mathsf{loc1}, \mathsf{loc2})$$
$$t_1 < t_2 < t_3$$

secure timeline

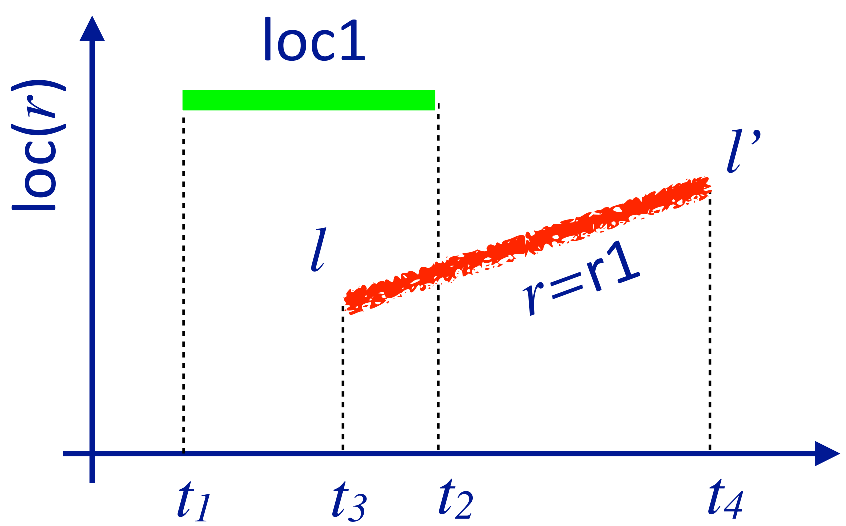
Consistent and Secure Timeline

- A ground instance of $(\mathcal{T},\mathcal{C})$ is consistent if it satisfies $\mathcal{C}$ and no state variable in $\mathcal{T}$ has more than on value at the same time
- $(\mathcal{T},\mathcal{C})$ is *consistent* if it has a consistent ground instance
- $(\mathcal{T},\mathcal{C})$ is *secure* if it is consistent and every ground instance that satisfies $\mathcal{C}$ is consistent

$$[t_1, t_2]\mathsf{loc}(r) = \mathsf{loc1}$$
$$[t_3, t_4]\mathsf{loc(r1)} : (l, l')$$
$$t_1 < t_2, t_3 < t_4$$

timeline consistent

but not secure



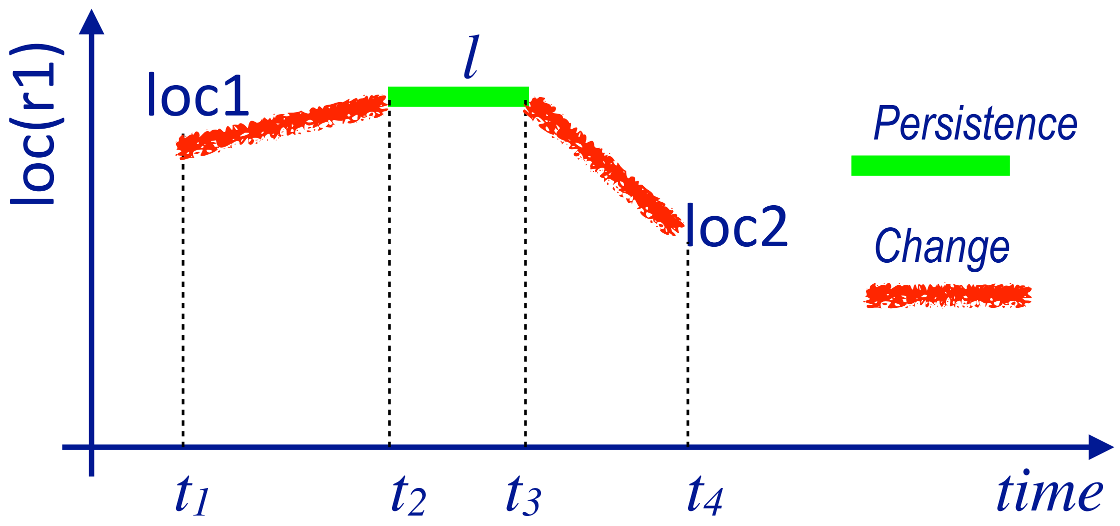
Conflicting assertions

=> separation constraints

$$r \neq \mathsf{r1}$$
$$t_2 < t_3$$
$$t_4 < t_1$$
$$t_2 = t_3, r = \mathsf{r1}, l = \mathsf{loc1}$$
$$t_4 = t_1, r = \mathsf{r1}, l' = \mathsf{loc1}$$

Causally supported timeline

- *Causal support* of the value of x : reasons that substantiate it
  - Prior knowledge about current state or dynamics of environment
  - Observation
  - Prediction of actions effects



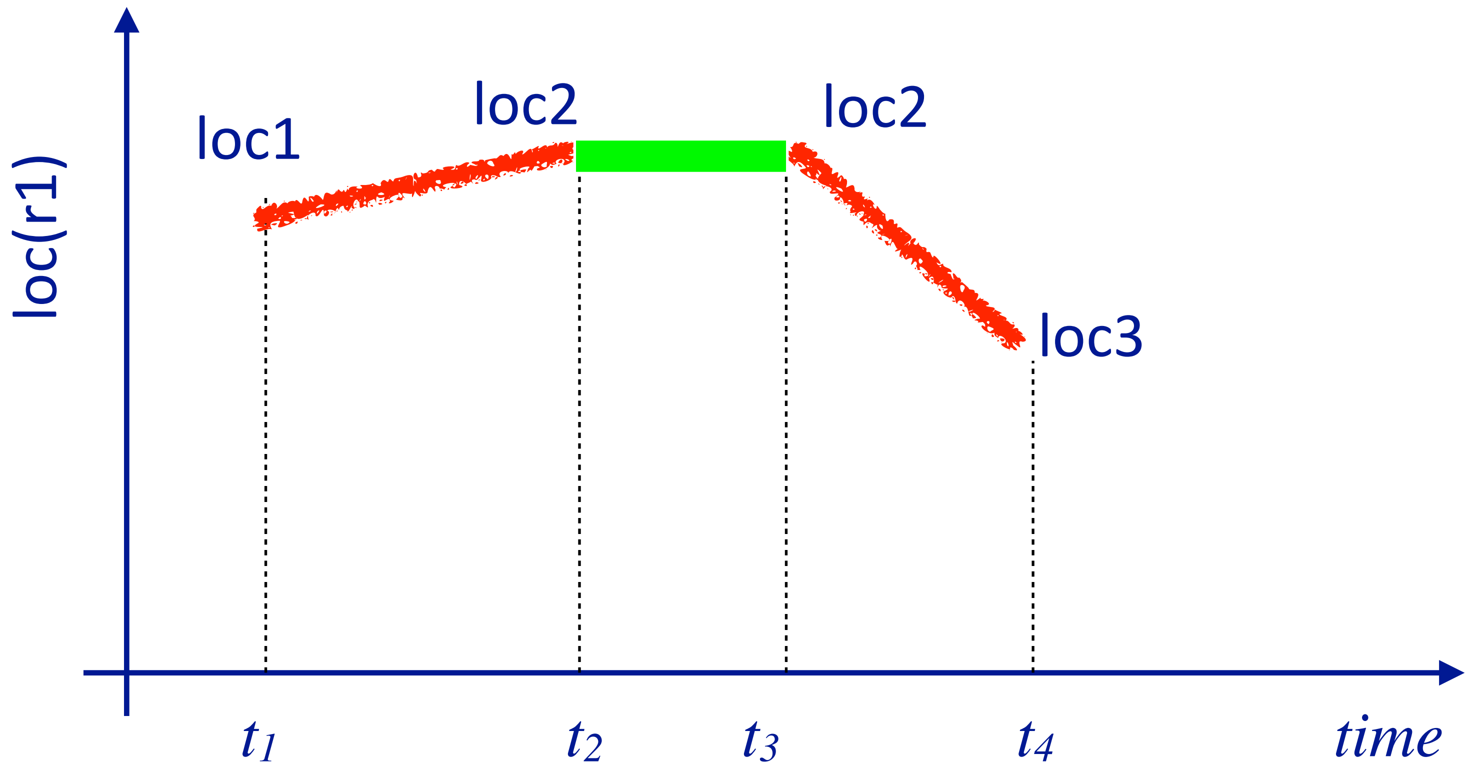
- Causally supported timeline:
  all its assertions have a causal support

Finding causal support

- Adding a persistence assertion

$[t_1, t_2]$loc(r1):(loc1,loc2), $[t_3, t_4]$loc(r1):(loc2,loc3)
$t_1 < t_2 < t_3 < t_4$

$[t_2, t_3]$ loc(r1)=loc2

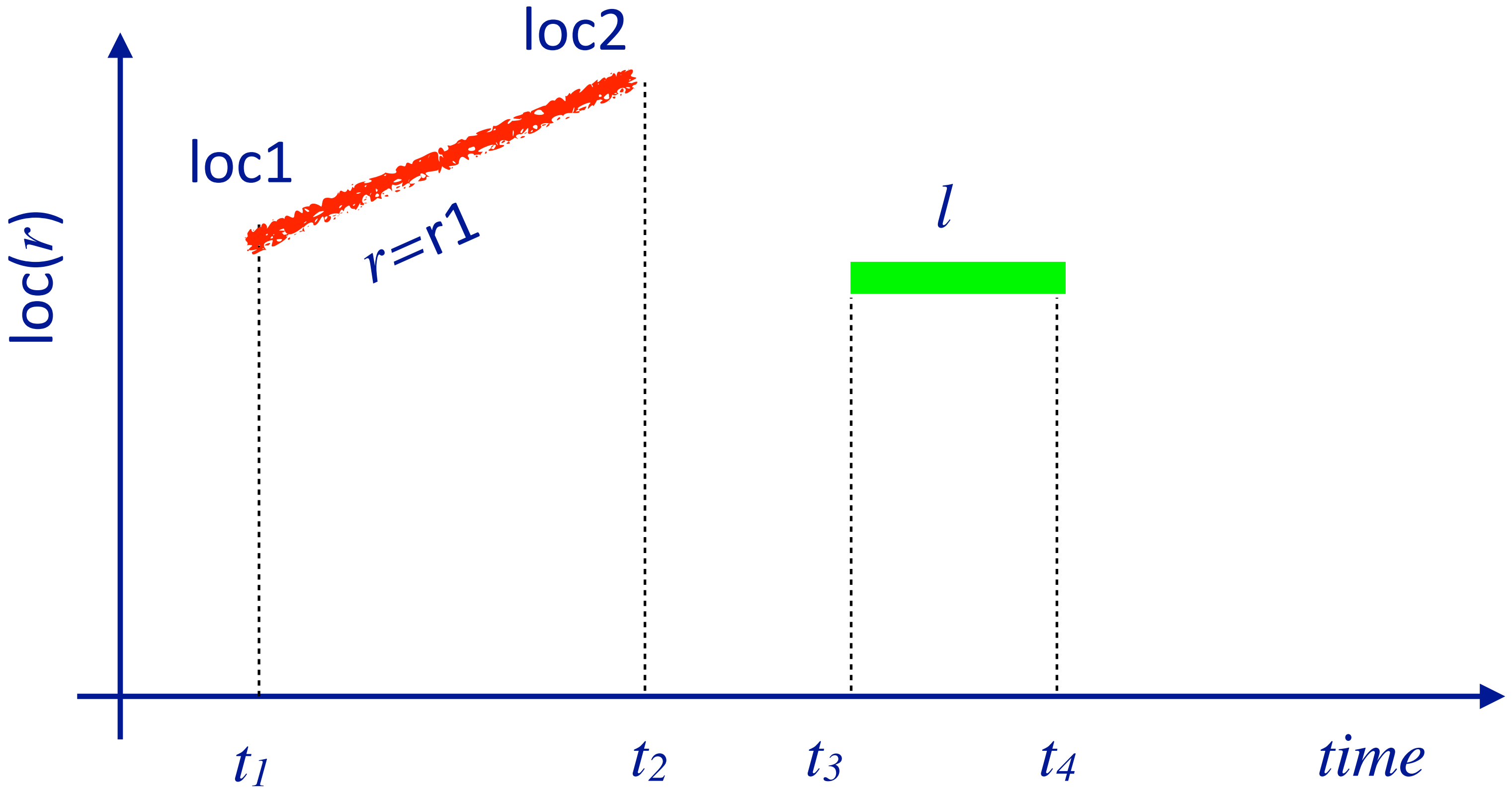


Finding causal support

- Adding a persistence assertion
- Adding constraints

$[t_1, t_2]$loc(r1):(loc1,loc2), $[t_3, t_4]\text{loc}(r) = l$
$t_1 < t_2 < t_3 < t_4$

$$t_2 = t_3, r = \text{r1}, l = \text{loc2}$$



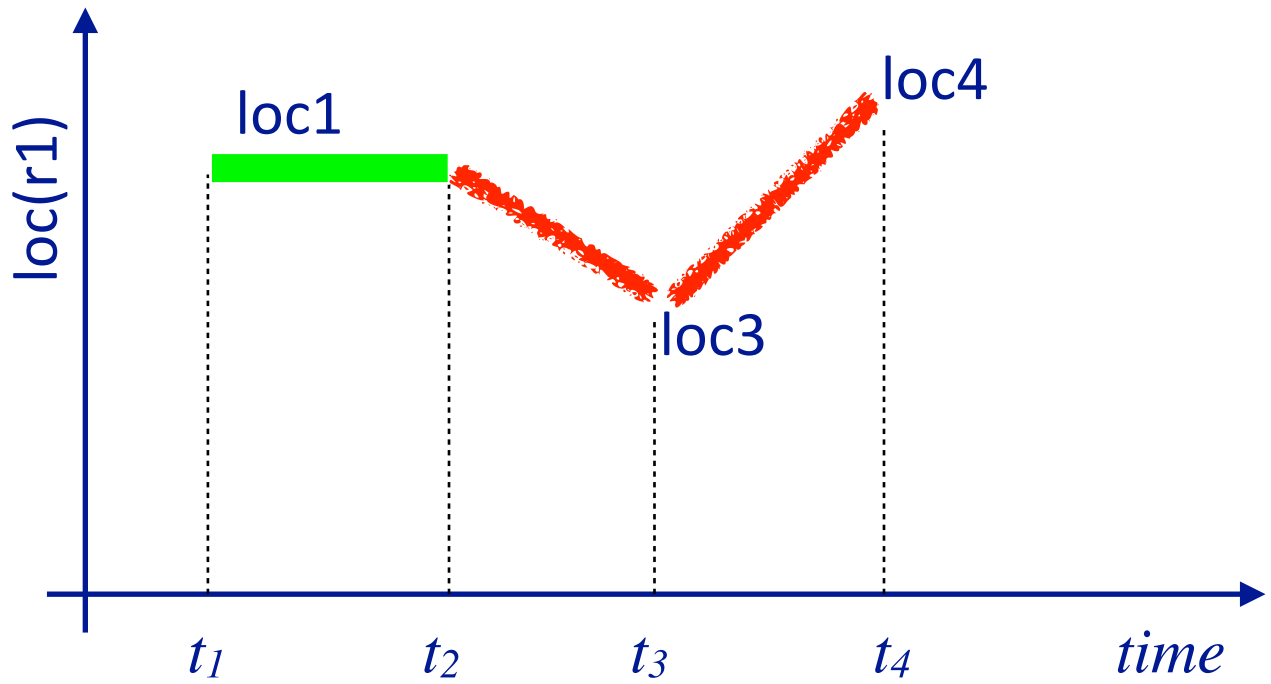
Finding causal support

- Adding a persistence assertion
- Adding constraints
- Adding a change assertion ➡ *corresponds to an additional action*

$[t_1, t_2]$loc(r1)= loc1, $[t_3, t_4]$loc(r1):(loc3,loc4)

$t_1 < t_2 < t_3 < t_4$

$[t_2, t_3]$loc(r1):(loc1,loc3)



Example

- Domain objects

$$r \in Robots,\ k \in Cranes,\ c \in Containers$$
$$p \in Piles,\ d \in Docks,\ w \in Waypoints$$

- State variables

$$\begin{aligned}
\mathsf{loc}(r) &\in Docks \cup Waypoints && \text{for } r \in Robots \\
\mathsf{freight}(r) &\in Containers \cup \{\mathsf{empty}\} && \text{for } r \in Robots \\
\mathsf{grip}(k) &\in Containers \cup \{\mathsf{empty}\} && \text{for } k \in Cranes \\
\mathsf{pos}(c) &\in Robots \cup Cranes \cup Piles && \text{for } c \in Containers \\
\mathsf{stacked\text{-}on}(c) &\in Containers \cup \{\mathsf{empty}\} && \text{for } c \in Containers \\
\mathsf{top}(p) &\in Containers \cup \{\mathsf{empty}\} && \text{for } p \in Piles \\
\mathsf{occupant}(d) &\in Robots \cup \{\mathsf{empty}\} && \text{for } d \in Docks
\end{aligned}$$

- Rigid relations

$$\mathsf{attached} \subseteq (Cranes \cup Piles) \times Docks$$
$$\mathsf{adjacent} \subseteq Docks \times Waypoints$$
$$\mathsf{connected} \subseteq \mathsf{Waypoints} \times \mathsf{Waypoints}$$

Example

- Primitive actions

$$
\begin{aligned}
\mathsf{leave}(r, d, w) &: \text{robot } r \text{ leaves dock } d \text{ to an adjacent waypoint } w \\
\mathsf{enter}(r, d, w) &: r \text{ enters } d \text{ from an adjacent waypoint } w \\
\mathsf{navigate}(r, w, w') &: r \text{ navigates from waypoint } w \text{ to a connected one } w' \\
\mathsf{stack}(k, c, p) &: \text{crane } k \text{ holding container } c \text{ stacks it on top of pile } p \\
\mathsf{unstack}(k, c, p) &: \text{crane } k \text{ unstacks a container } c \text{ from the top of pile } p \\
\mathsf{put}(k, c, r) &: \text{crane } k \text{ holding a container } c \text{ and puts it onto } r \\
\mathsf{take}(k, c, r) &: \text{crane } k \text{ takes container } c \text{ from robot } r
\end{aligned}
$$

Primitives

$\mathsf{take}(k, c, r)$

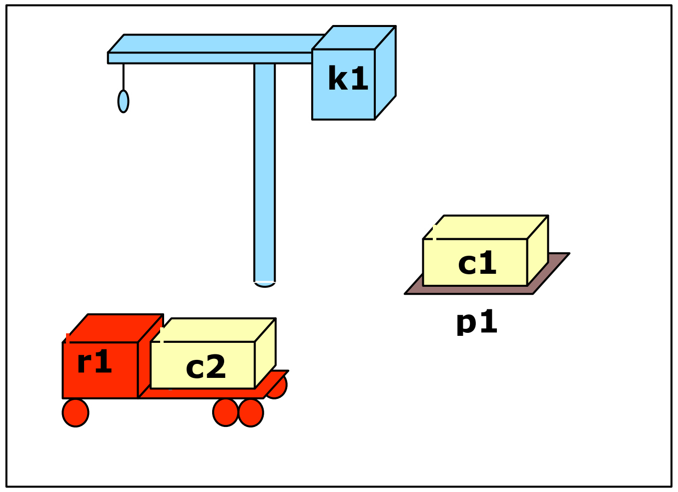
assertions: $[t_s, t_e]\mathsf{pos}(c){:}(r, k)$

$[t_s, t_e]\mathsf{grip}(k){:}(\mathsf{empty}, c)$

$[t_s, t_e]\mathsf{freight}(r){:}(c, \mathsf{empty})$

$[t_s, t_e]\mathsf{loc}(r){=}d$

constraints: $\mathsf{attached}(k, d)$



$\mathsf{leave}(r, d, w)$

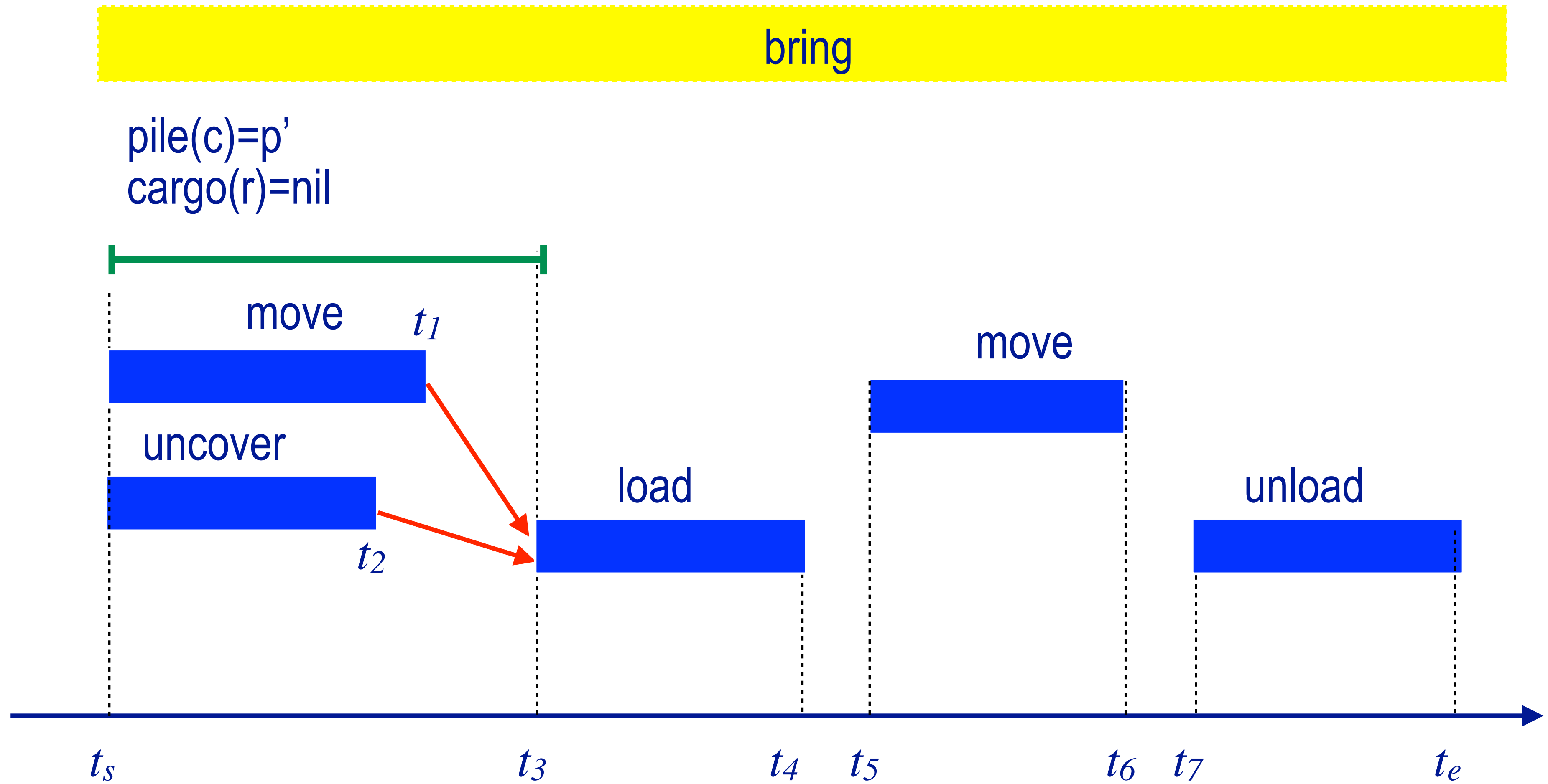
assertions: $[t_s, t_e]\mathsf{loc}(r){:}(d, w)$

$[t_s, t_e]\mathsf{occupant}(d){:}(r, \mathsf{empty})$

constraints: $t_e \leq t_s + \delta_1$

$\mathsf{adjacent}(d, w)$

Tasks and methods



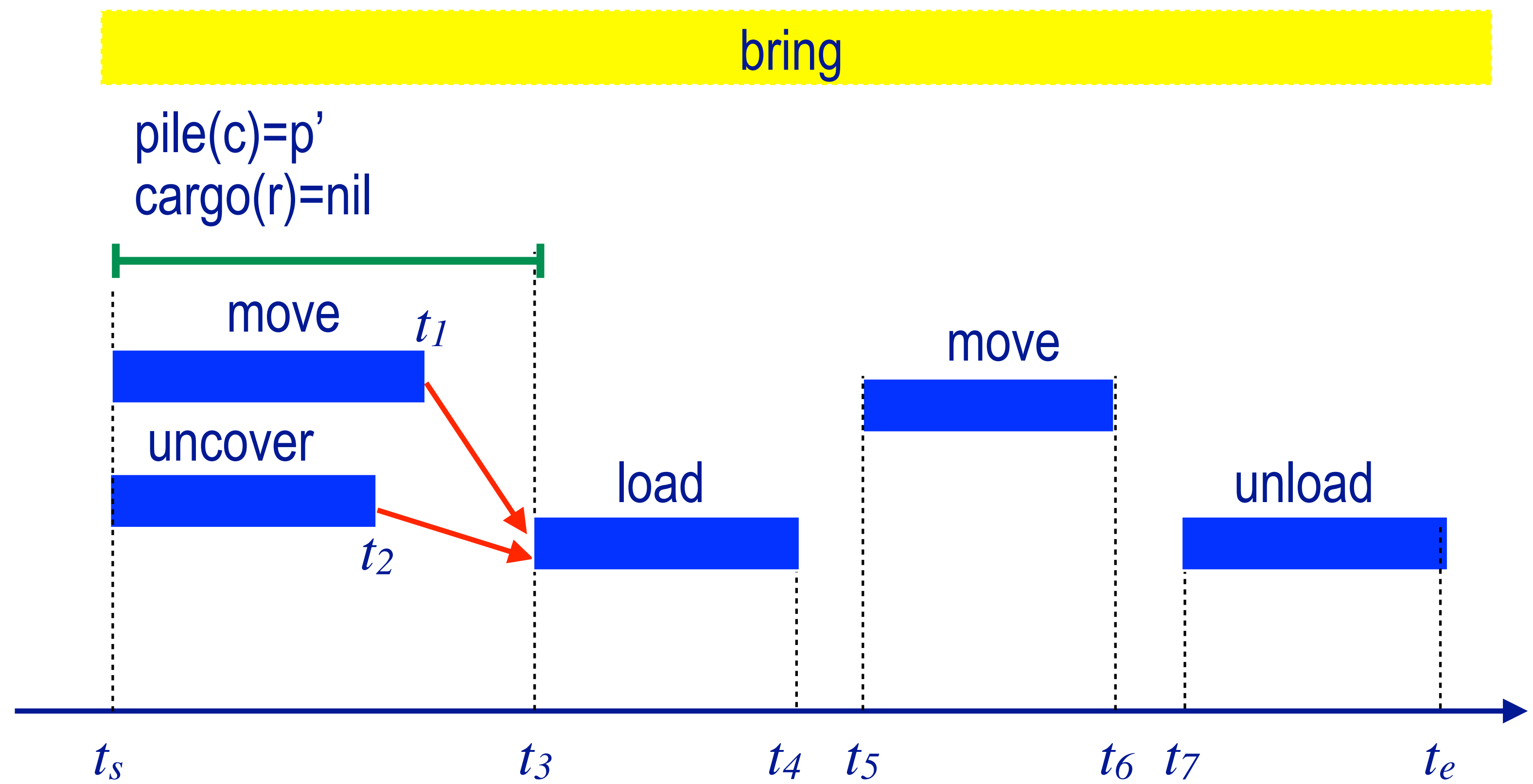
- Tasks

$[t_s, t_e]\text{bring}(r, c, p)$

$$[t_s, t_1]\text{move}(r, d)$$
$$[t_s, t_2]\text{uncover}(k, c, p)$$
$$[t_3, t_4]\text{load}(k, r, c, p)$$
$$[t_7, t_e]\text{unload}(k, r, c, p)$$

Tasks and methods

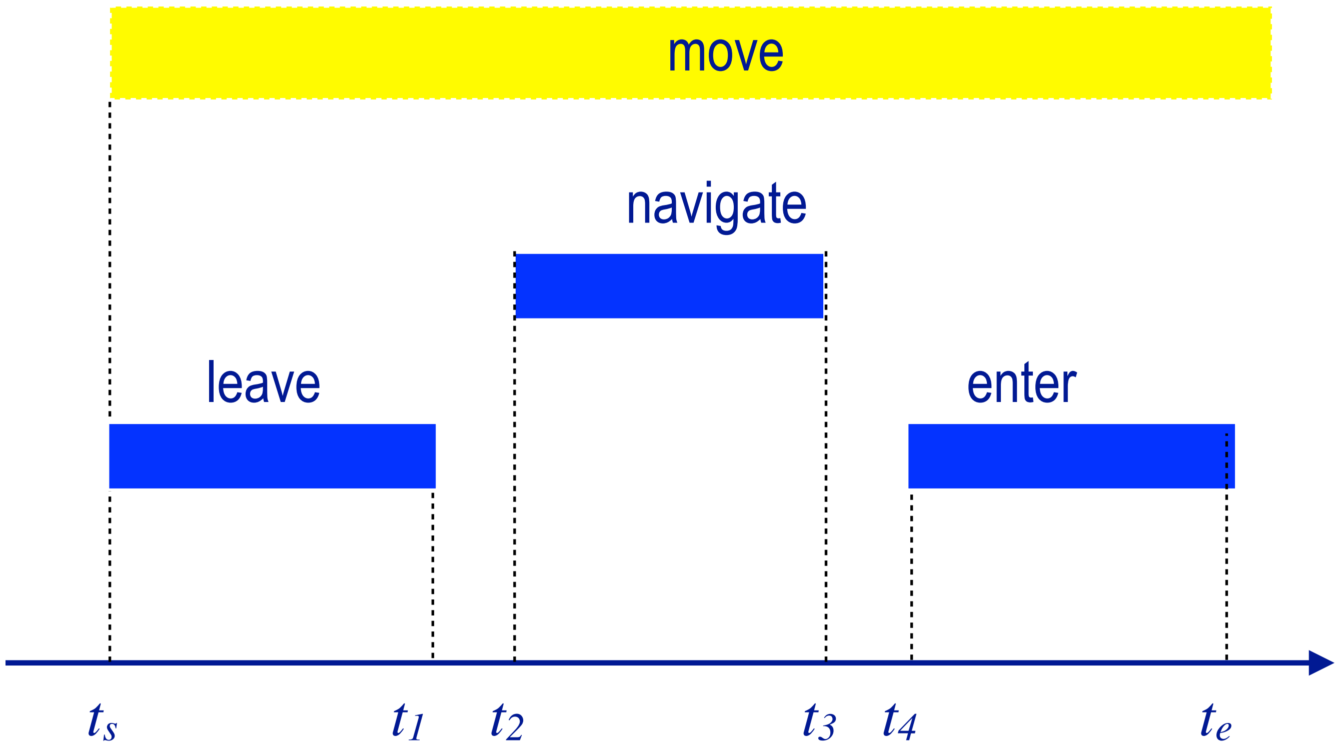
- Methods



$$
\begin{aligned}
&\textsf{m-bring}(r, c, p, p', d, d', k, k') \\
&\quad \text{task: } \textsf{bring}(r, c, p) \\
&\quad \text{refinement: } [t_s, t_1]\textsf{move}(r, d') \\
&\qquad [t_s, t_2]\textsf{uncover}(c, p') \\
&\qquad [t_3, t_4]\textsf{load}(k', r, c, p') \\
&\qquad [t_5, t_6]\textsf{move}(r, d) \\
&\qquad [t_7, t_e]\textsf{unload}(k, r, c, p) \\
&\quad \text{assertions: } [t_s, t_3]\textsf{pile}(c){=}p' \\
&\qquad [t_s, t_3]\textsf{freight}(r){=}\textsf{empty} \\
&\quad \text{constraints: } \textsf{attached}(p', d'),\ \textsf{attached}(p, d),\ d \neq d' \\
&\qquad \textsf{attached}(k', d'),\ \textsf{attached}(k, d) \\
&\qquad t_1 \leq t_3,\ t_2 \leq t_3,\ t_4 \leq t_5,\ t_6 \leq t_7
\end{aligned}
$$

Tasks and methods

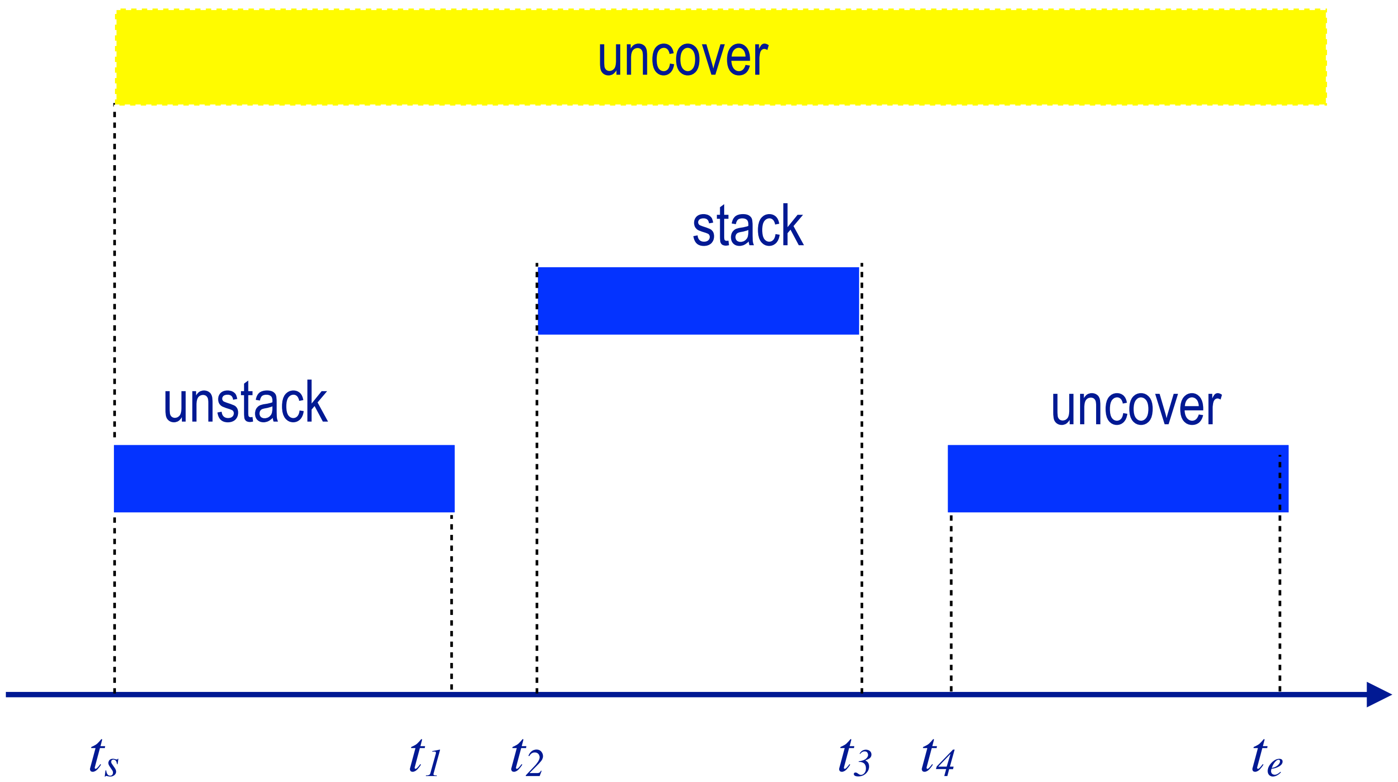
- Methods



$$
\begin{aligned}
&\text{m-move1}(r, d, d', w, w') \\
&\quad \text{task: } \text{move}(r, d) \\
&\quad \text{refinement: } [t_s, t_1]\text{leave}(r, d', w') \\
&\qquad [t_2, t_3]\text{navigate}(w', w) \\
&\qquad [t_4, t_e]\text{enter}(r, d, w) \\
&\quad \text{assertions: } [t_s, t_s + 1]\text{loc}(r){=}d' \\
&\quad \text{constraints: } \text{adjacent}(d, w),\ \text{adjacent}(d', w'),\ d \neq d' \\
&\qquad \text{connected}(w, w') \\
&\qquad t_1 \leq t_2,\ t_3 \leq t_4
\end{aligned}
$$

Tasks and methods

- Methods



m-uncover(c, p, k, d, p')

task: uncover(c, p)

refinement: $[t_s, t_1]$unstack(k, c', p)
$[t_2, t_3]$stack(k, c', p')
$[t_4, t_e]$uncover(c, p)

assertions: $[t_s, t_s + 1]$pile$(c){=}p$
$[t_s, t_s + 1]$top$(p){=}c'$
$[t_s, t_s + 1]$grip$(k){=}$empty

constraints: attached(k, d), attached(p, d)
attached(p', d), $p \neq p'$, $c' \neq c$
$t_1 \leq t_2$, $t_3 \leq t_4$

Chronicles

- Chronicle $\phi = (\mathcal{A}, \mathcal{S}_\mathcal{T}, \mathcal{T}, \mathcal{C})$
  - $\mathcal{A}$: temporally qualified actions and tasks
  - $\mathcal{S}_\mathcal{T}$: a priori supported assertions
  - $\mathcal{T}$: temporally qualified assertions
  - $\mathcal{C}$: constraints

- ϕ represents
  - Current state and future predicted events
  - Tasks to be performed
  - Assertions and constraints to be satisfied

    => planning problems and (partial) plans

Chronicles

ϕ_0 :

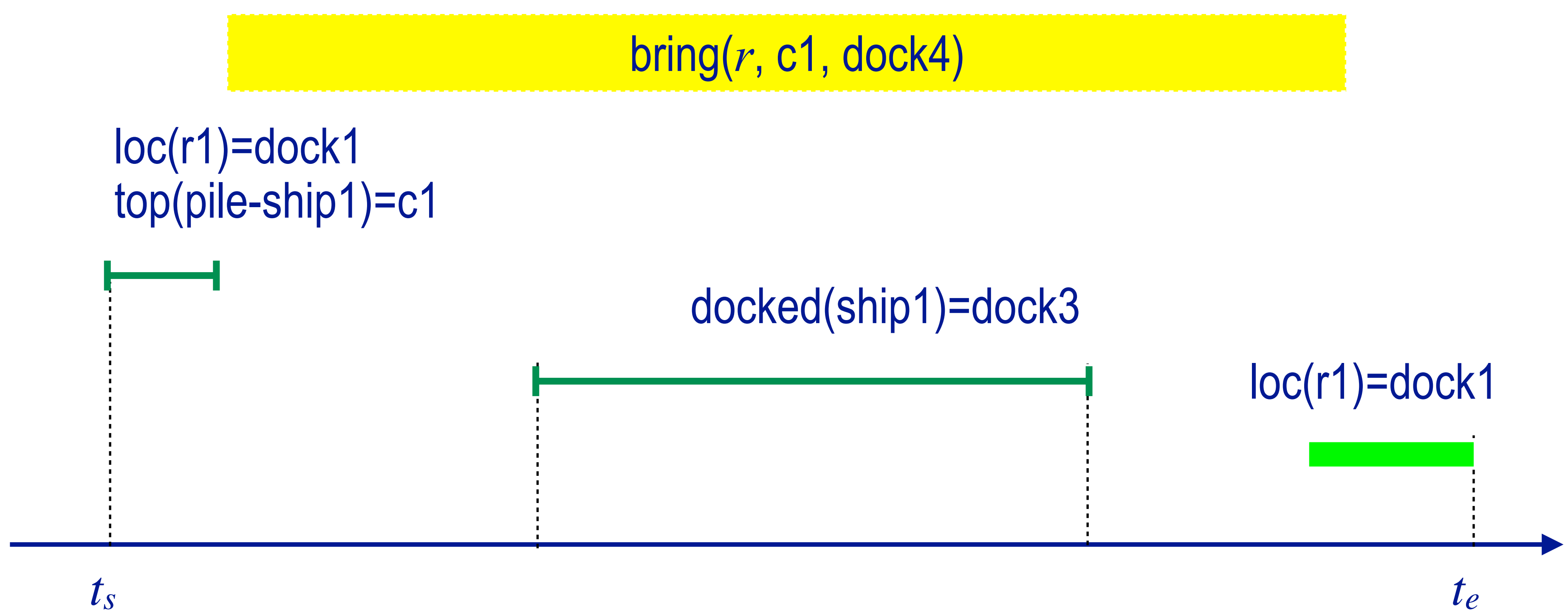
tasks: $[t, t']$bring(r, c1, dock4)

supported: $[t_s]$loc(r1)=dock1
$[t_s]$loc(r2)=dock2
$[t_s + 10, t_s + \delta]$docked(ship1)=dock3
$[t_s]$top(pile1)=c1
$[t_s]$pos(c1)=pallet

assertions: $[t_e]$loc(r1) = dock1
$[t_e]$loc(r2) = dock2

constraints: attached(pile1,ship1)
$t_s < t < t' < t_e$, $20 \leq \delta \leq 30$, $t_s = 0$

Initial chronicle



- Partial plan

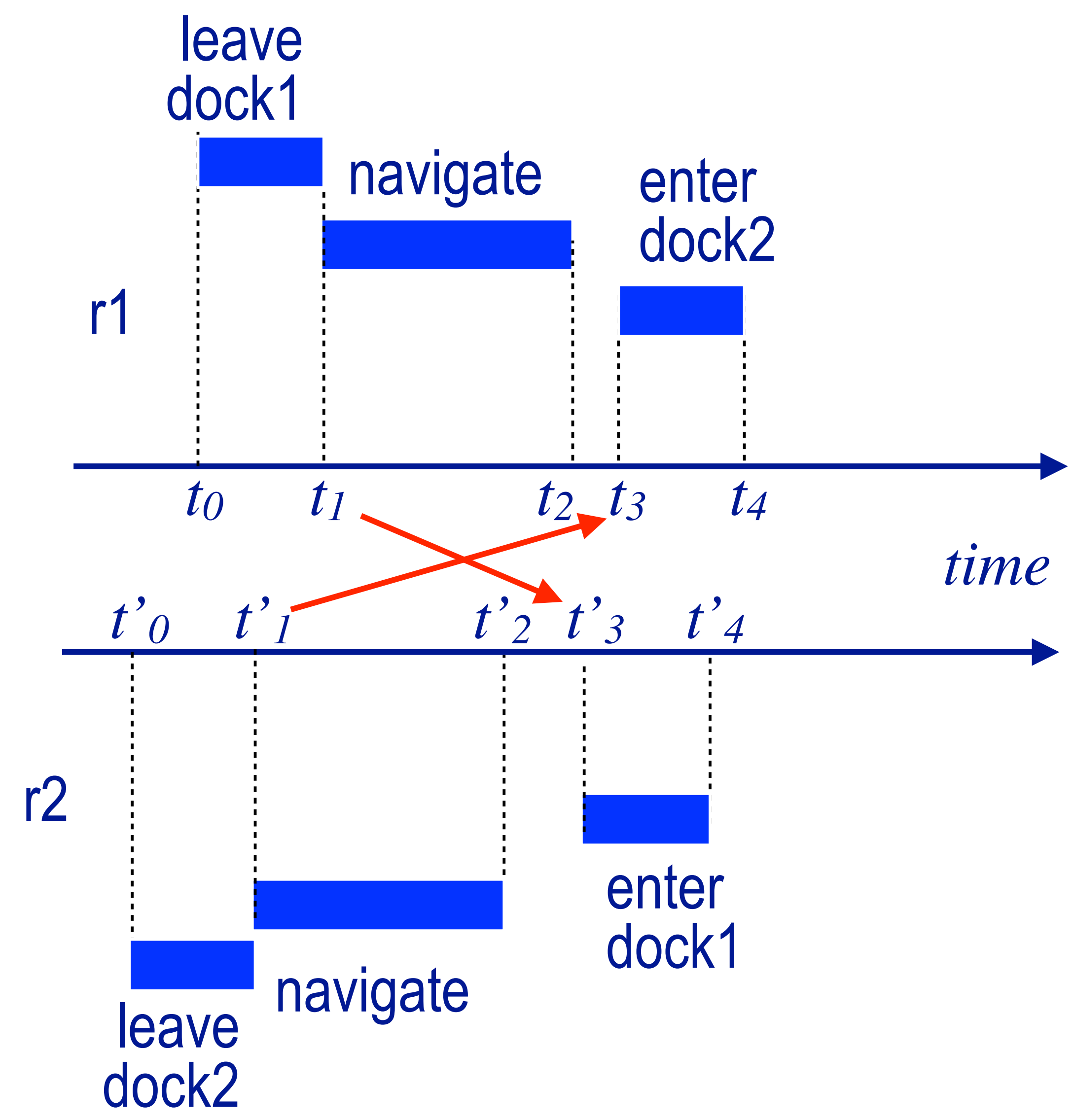
ϕ :

tasks: $[t_0, t_1]$leave(r1,dock1,w1)
$[t_1, t_2]$navigate(r1,w1,w2)
$[t_3, t_4]$enter(r1,dock2,w2)
$[t'_0, t'_1]$leave(r2,dock2,w2)
$[t'_1, t'_2]$navigate(r2,w2,w1)
$[t'_3, t'_4]$enter(r2,dock1,w1)

supported: $\mathcal{S}_{\mathcal{T}}$

assertions: $\mathcal{T}$

constraints: $t'_1 < t_3,\ t_1 < t'_3,\ t_s < t_0$
$t_s < t'_0,\ t_4 < t_e,\ t'_4 < t_e$
adjacent(dock1,w1), adjacent(dock2,w2)
connected(w1,w2)



Outline

- ✓ Introduction
- ✓ Representation
- Temporal planning
  - Resolvers and flaws
  - Search space
- Consistency and controllability
- Acting with executable primitives
- Acting with atemporal refinement
- Conclusion

Temporal Planning

Starting from an *initial chronicle*:

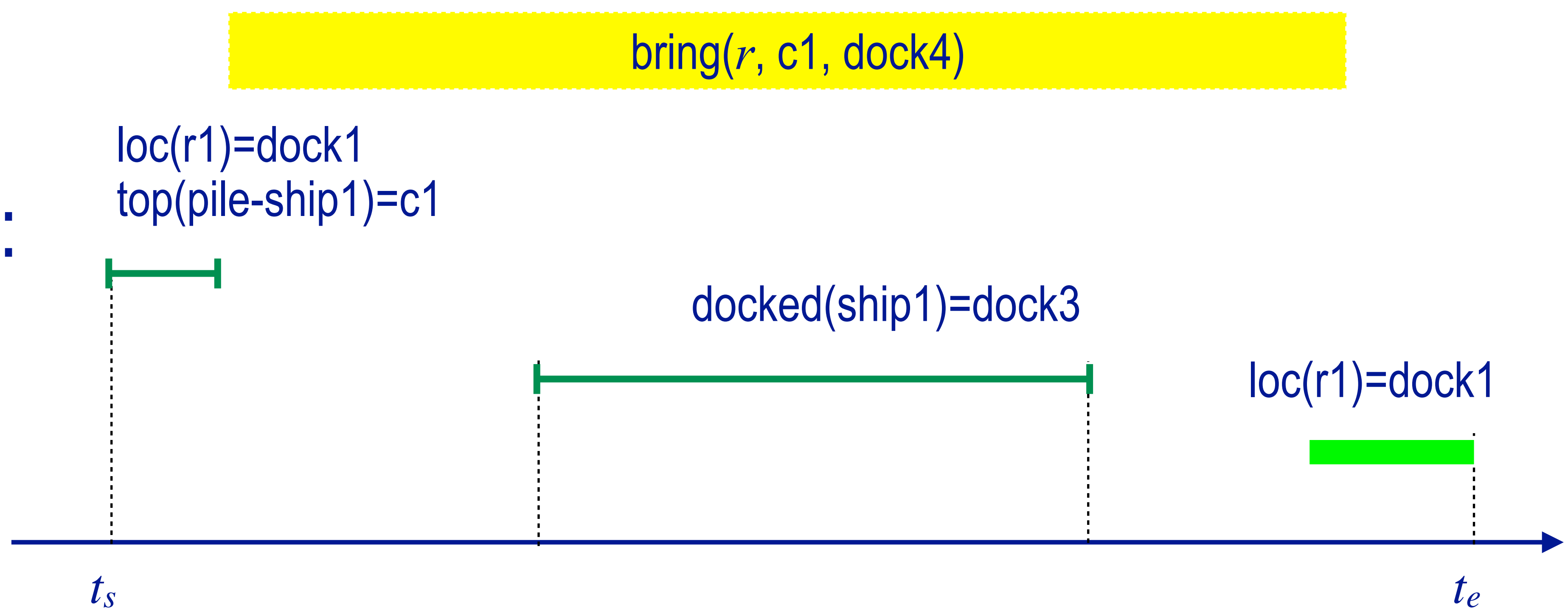
- Refine into primitive actions
- Add causal supports to
- Add separation constraints for

nonrefined tasks

nonsupported assertion

conflicting assertions

flaws



Resolvers of flaws:

- Methods
- Actions
- Persistence assertions and constraints

A chronicle ϕ is a valid solution plan iff

- ϕ does not contain nonrefined tasks
- all assertions in ϕ are causally supported, either by supported assertions initially in ϕ_0 or by assertions from methods and primitives in the plan
- the chronicle ϕ is secure

no conflicting assertions in consistant instances

Temporal Planning

TemPlan(ϕ, Σ)

$Flaws \leftarrow$ set of flaws of ϕ

if $Flaws = \emptyset$ then return ϕ

arbitrarily select $f \in Flaws$

$Resolvers \leftarrow$ set of resolvers of f

if $Resolvers = \emptyset$ then return failure

nondeterministically choose $\rho \in Resolvers$

$\phi \leftarrow$ Transform(ϕ, ρ)

Templan(ϕ, Σ)

Combines in CSP-based approach

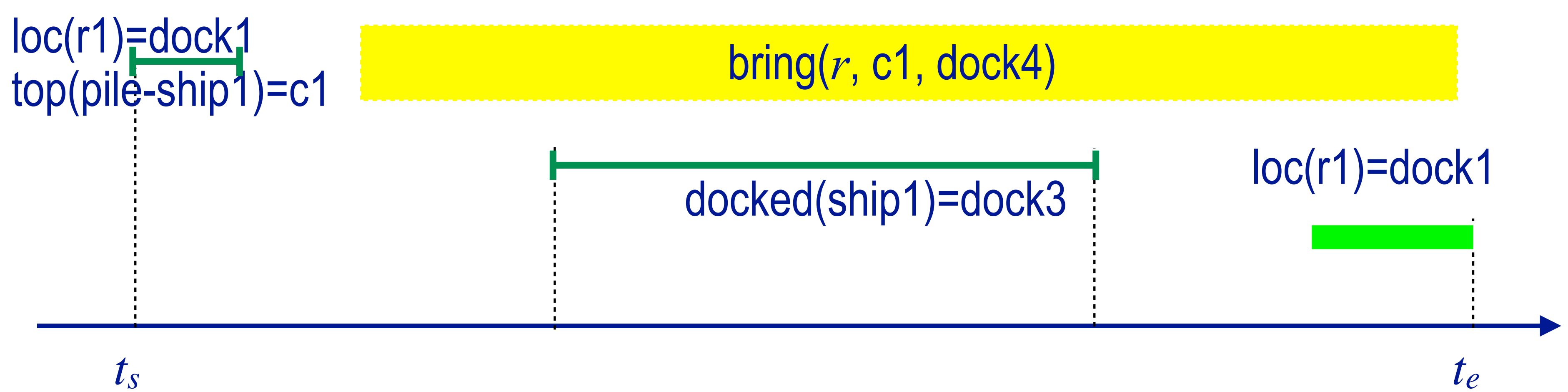
- *task decomposition planning*
- *plan-space planning*
- *temporal planning*

Resolvers for flaws

Resolver for a nonrefined *task* in ϕ:
an instance m of a method applicable to the *task* s.t. all the constraints of m are consistent with those of ϕ.

Transforming $\phi = (\mathcal{A}, \mathcal{S}_{\mathcal{T}}, \mathcal{T}, \mathcal{C})$ with resolver m:

- replace in $\mathcal{A}$ the task by the subtasks and actions of m
- add the assertions of m and those of the primitives in m either to $\mathcal{S}_{\mathcal{T}}$ if these assertions are causally supported or to $\mathcal{T}$
- add to $\mathcal{C}$ the constraints of m and those of its actions.

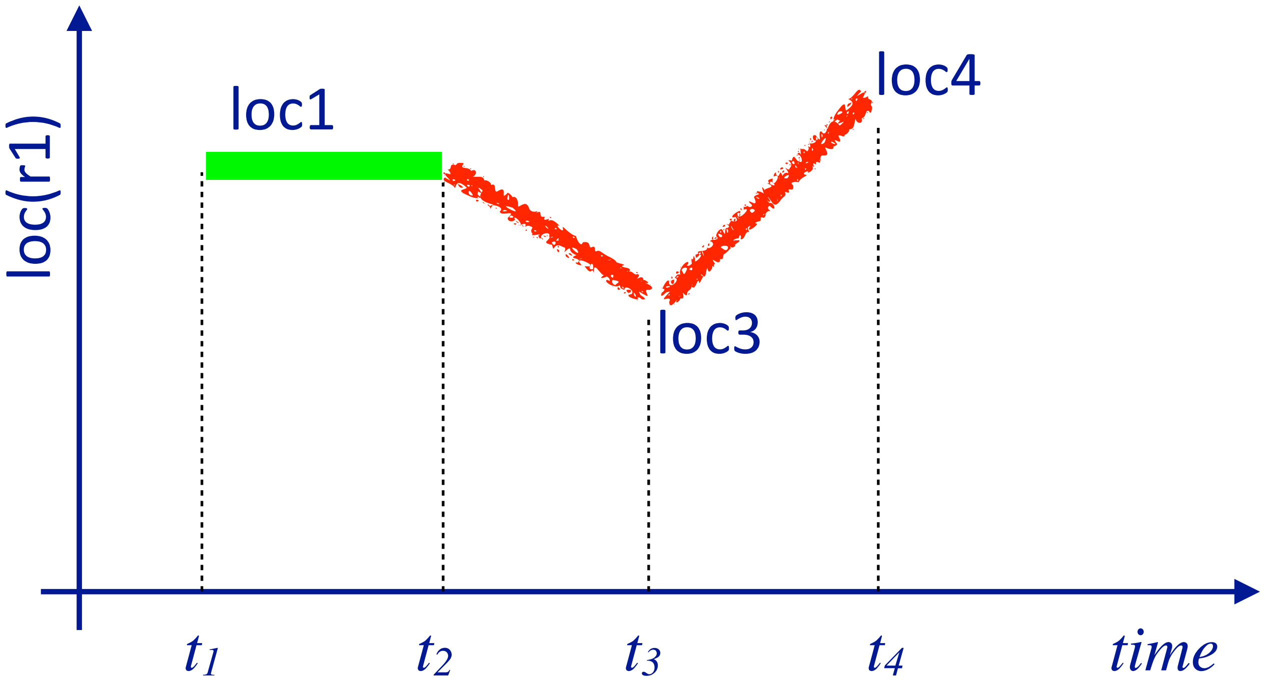


Resolvers for flaws

Nonsupported assertions in $\phi = (\mathcal{A}, \mathcal{S}_{\mathcal{T}}, \mathcal{T}, \mathcal{C})$: those initially in ϕ_0 plus those from the refinement of tasks and the insertion of actions

Different ways to support an assertion $\alpha \in \mathcal{T}$:

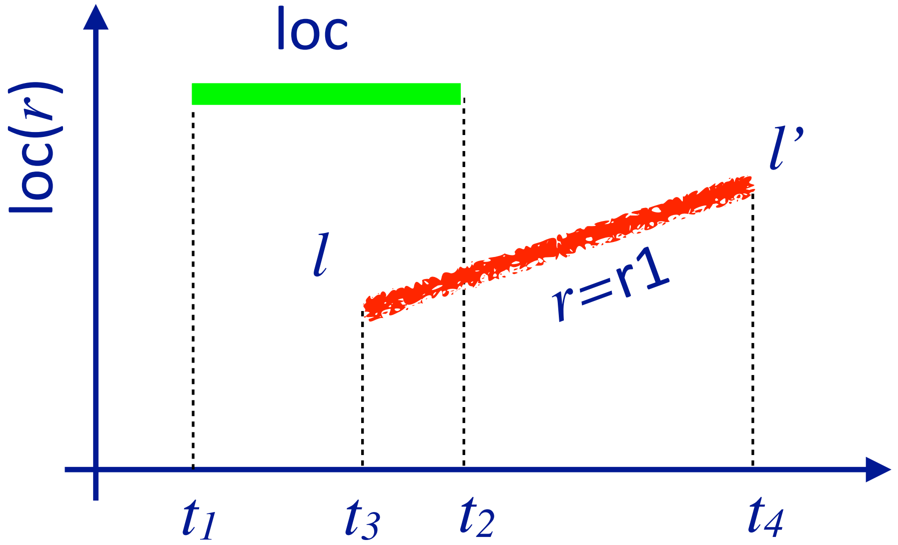
- add in $\mathcal{C}$ constraints on object and temporal variables
- add a persistence assertion in $\mathcal{S}_{\mathcal{T}}$
- add in $\mathcal{A}$ a task or an action that brings an assertion supporting α



Resolvers for flaws

Flaws due to conflicting assertions handled incrementally by maintaining ϕ a secure chronicle:

- Detect possible conflicts for each new assertion in ϕ
- Find sets of separation constraints consistent with the constraints in current ϕ
- Add separation constraints to ϕ



Search Space

- Planning search space: a directed graph
  - Node: a chronicle ϕ
  - Edge (ϕ, ϕ'): $\phi' = \text{Transform}(\phi, \varrho)$, ϱ resolver for flaw in ϕ
  - *Acyclic graph*
  - *Not necessarily finite*
- Heuristics
  - Flaw with smallest numbers of resolvers (as in *variable-ordering*)
  - Resolver the least constraining to current ϕ (as in *value-ordering)*
  - Elaborate heuristics based on domain transition graphs and reachability graphs
- Critical operation: maintain consistency of *C*

Outline

- ✓ Introduction
- ✓ Representation
- ✓ Temporal planning
- Consistency and controllability
  - Object constraints
  - Temporal constraint
  - Controllability of an STNU
- Acting with executable primitives
- Acting with atemporal refinement
- Conclusion

Consistency

- Object constraints maintained by Templan

  $l \neq$ loc2, $l \in$ {loc3, loc4}

  $r =$ r1, $o \neq o'$

  loc(r) $\neq l'$

- Temporal constraints maintained by Templan

  $a < t$

  $t < t'$

  $a \leq t - t \leq b$

- Possibly coupled constraints

  $t < f(l, r)$

  => Assume no coupled constraint

Object Constraint

- Unary and binary constraints on object variables due to binding and separation constraints and rigid relations
- Corresponds to maintaining the consistency of a general CSP over finite domains => NP-complete problem
- Incremental arc or path consistency algorithms
  - Not complete, but efficient trade-off for filtering inconsistent instances
  - Do not reduce the completeness of the algorithm, just prune fewer nodes in search tree
- Combined with complete algorithms, e.g., forward-checking on the free variables remaining in the final plan

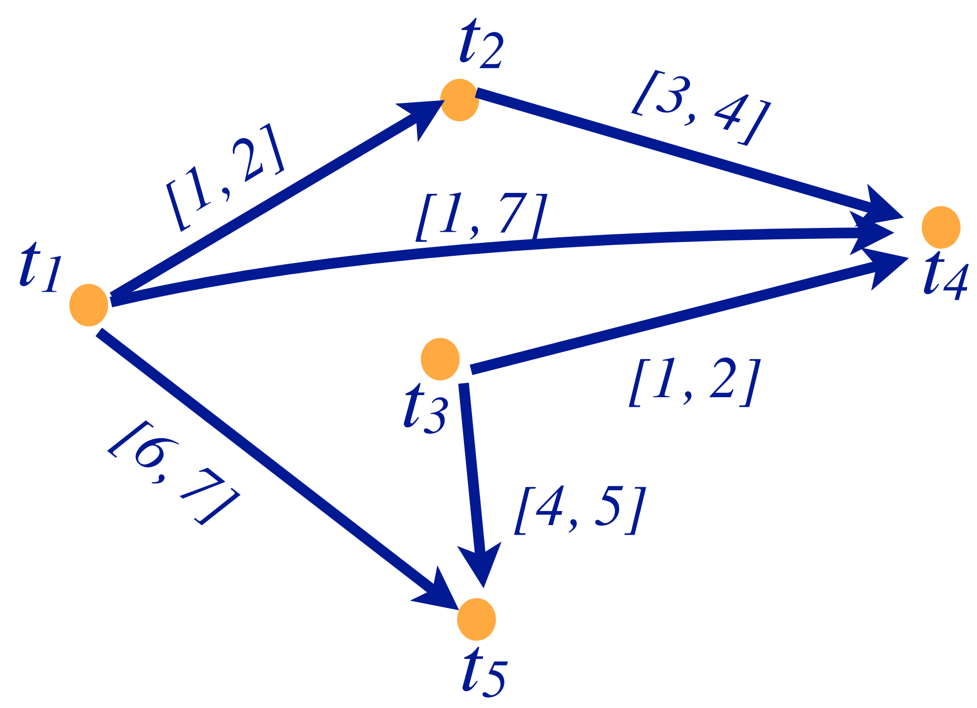
Temporal Constraints

- Simple temporal networks (STN)

  $a \leq t_j - t_i \leq b$

  notation $r_{ij} = [a,b]$

  entails $r_{ji} = [-b,-a]$



STN

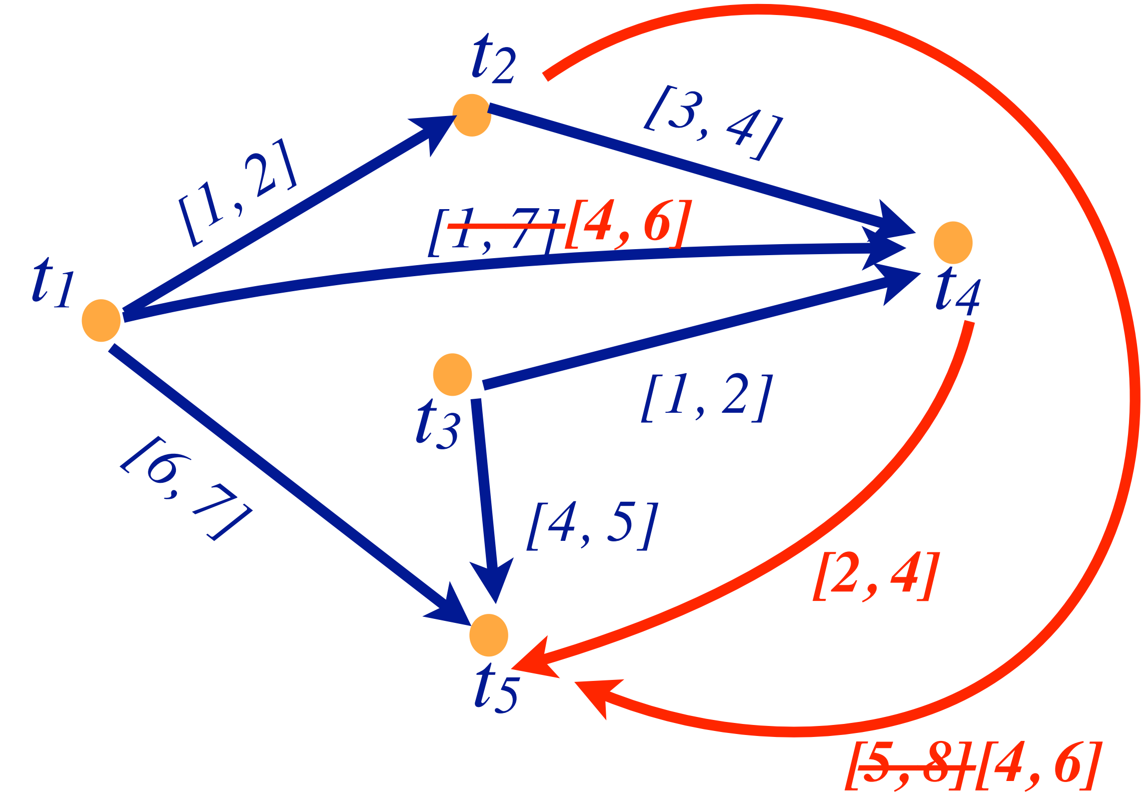
- Incrementally *synthesized* by Templan starting from ϕ_0
- Incrementally *instantiated* at acting time
- *Maintained consistent throughout planning and acting*

- Simple temporal networks (STN)

$$a \leq t_j - t_i \leq b$$

notation $r_{ij} = [a,b]$

entails $r_{ji} = [-b,-a]$



- Constraint propagation rules

Conditions	Propagated constraint	
$t_1 \xrightarrow{[a,b]} t_2\ ,\ t_2 \xrightarrow{[a',b']} t_3$	$t_1 \xrightarrow{[a+a',b+b']} t_3$	$r_{12} \bullet r_{23}$
$t_1 \xrightarrow{[a,b]} t_2\ ,\ t_1 \xrightarrow{[a',b']} t_2$	$t_1 \xrightarrow{[max\{a,a'\},min\{b,b'\}]} t_2$	$r_{12} \cap r'_{12}$

Path consistency algorithm

$\mathsf{PC}(\mathcal{V}, \mathcal{E})$
    for each $k : 1 \leq k \leq n$ do
        for each pair $i, j : 1 \leq i < j \leq n, i \neq k, j \neq k$ do
            $r_{ij} \leftarrow r_{ij} \cap [r_{ik} \bullet r_{kj}]$
            if $r_{ij} = \varnothing$ then return *inconsistent*

- Algorithm complete
- Returns a *minimal* network when consistent
- Complexity in time $O(n^3)$, incremental update in $O(n^2)$

Controllability

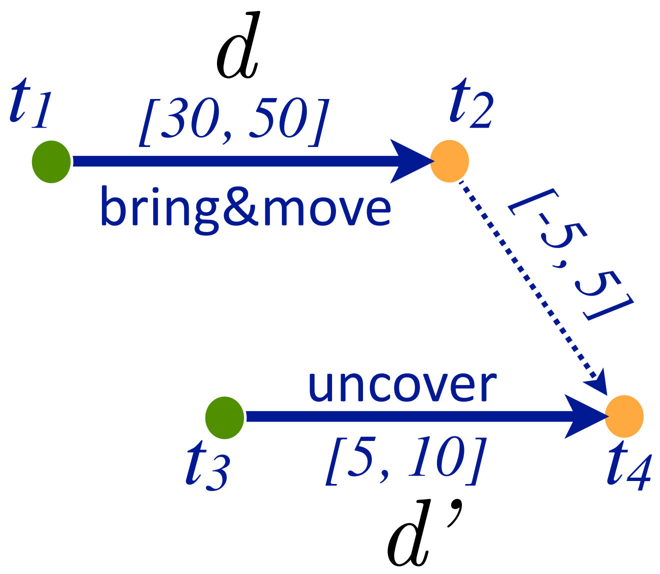
- Controllable *vs* contingent time points
  - t_1 and t_3 : controllable
  - t_2 and t_4 : contingent

    random variables that are known

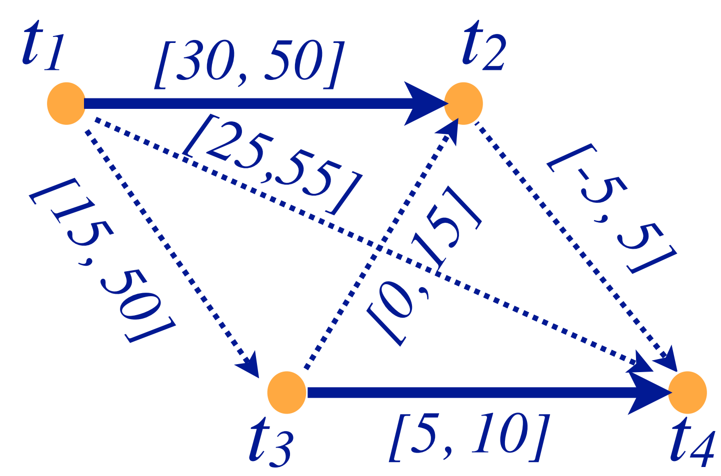
    to satisfy some constraints

- PC cannot be allowed to constrain a contingent time point
- Even if minimal network does not constrain any contingent time point the corresponding plan may not be feasible

Controllability



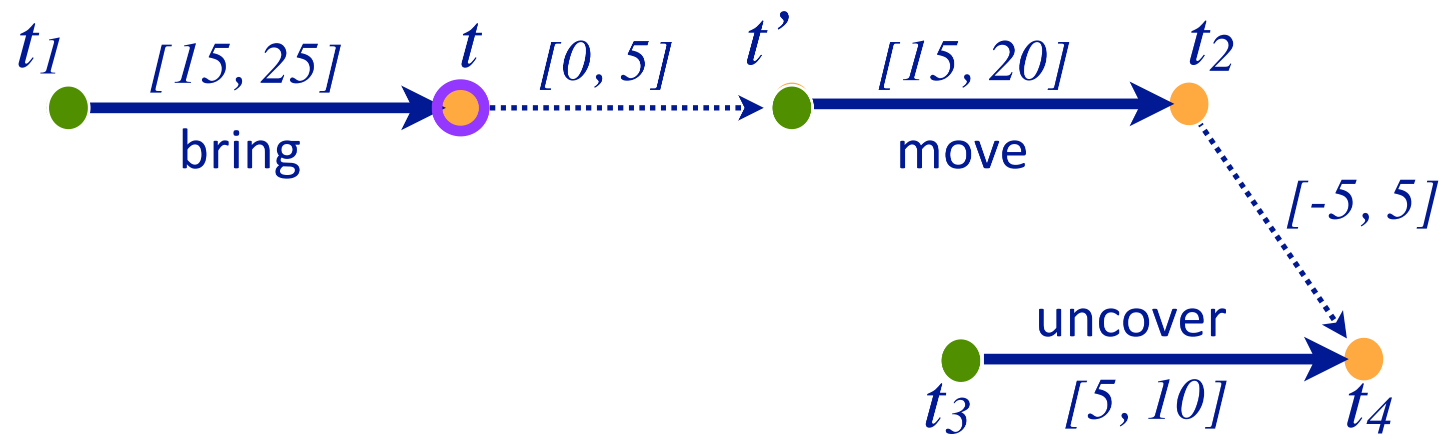
minimal network



$$d - d' - 5 \leq t_3$$
$$t_3 \leq d - d' + 5$$

$$\Rightarrow 40 \leq t_3 \leq 25 \text{ !!}$$

Dynamic Controllability



observe t

assign t' at any moment after t in $[0, 5]$

assign t_3 10 units after t'

Dynamic Controllability

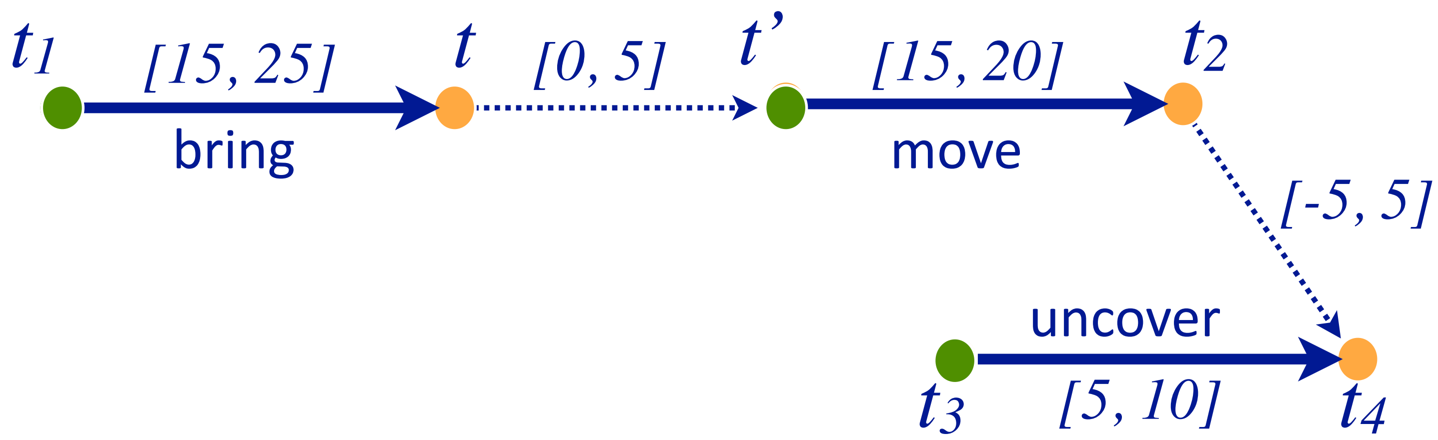
- Simple temporal network with uncertainty (STNU)
  - Controllable and contingent time points
  - Constraints, as in STN, controllable and contingent

- Controllable STNU: there exist values for controllable points that meets all constraints
  - *Strong controllability*: solution works for *all* possible values of contingent points in their predicted intervals
  - *Weak controllability*: solution as a function of the values of contingent points, if *known* in advance
  - *Dynamic controllability*: solution that is built dynamically, for each controllable point given the *observation* of past contingent points

Dynamic Controllability

- A *dynamic execution strategy* for an STNU:
  online procedure for assigning, in some order, a value to each controllable point t, (i.e., triggering commands at right moment)
  - such that all controllable constraints are met, and
  - given that the values of all contingent variables *preceding* t are known and fit their assumed constraints
- An STNU is *dynamically controllable* if there exists a dynamic execution strategy for it
- Assigning a value to controllable t = triggering a command

Dynamic Controllability



observe t

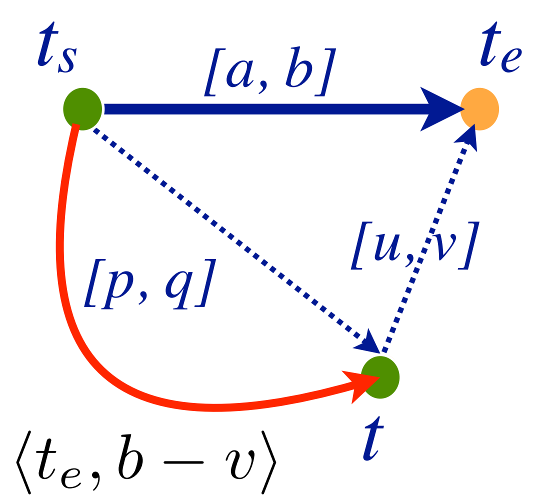
assign t' at any moment after t in $[0, 5]$

assign t_3 10 units after t'

Checking dynamic controllability of an STNU

- For a chronicle $\phi = (\mathcal{A}, \mathcal{S}_\mathcal{T}, \mathcal{T}, \mathcal{C})$ temporal constraints in $\mathcal{C}$ correspond to an STNU
- TemPlan: maintains incrementally STNU dynamically controllable
  - If Path Consistency reduces a contingent constraint => not dynamically controllable
  - Otherwise: test of dynamic controllability as an extension of Path Consistency with additional constraint propagation rules

Dynamic Controllability Checking



if $u < 0$ and $v \geq 0$ then
t should wait until either $t_s + b - v$
or t_e occurs

Constraint propagation rules

Conditions	Propagated constraint
$t_s \xRightarrow{[a,b]} t_e\ ,\ t \xrightarrow{[u,v]} t_e\ ,\ u \geq 0$	$t_s \xrightarrow{[b',a']} t$
$t_s \xRightarrow{[a,b]} t_e\ ,\ t \xrightarrow{[u,v]} t_e\ ,\ u < 0\ ,\ v \geq 0$	$t_s \xrightarrow{\langle t_e,b' \rangle} t$
$t_s \xRightarrow{[a,b]} t_e\ ,\ t_s \xrightarrow{\langle t_e,u \rangle} t$	$t_s \xrightarrow{[min\{a,u\},\infty]} t$
$t_s \xrightarrow{\langle t_e,b \rangle} t\ ,\ t' \xrightarrow{[u,v]} t$	$t_s \xrightarrow{\langle t_e,b' \rangle} t'$
$t_s \xrightarrow{\langle t_e,b \rangle} t\ ,\ t' \xRightarrow{[u,v]} t\ ,\ t_e \neq t$	$t_s \xrightarrow{\langle t_e,b-u \rangle} t'$

$a' = a - u, b' = b - v$

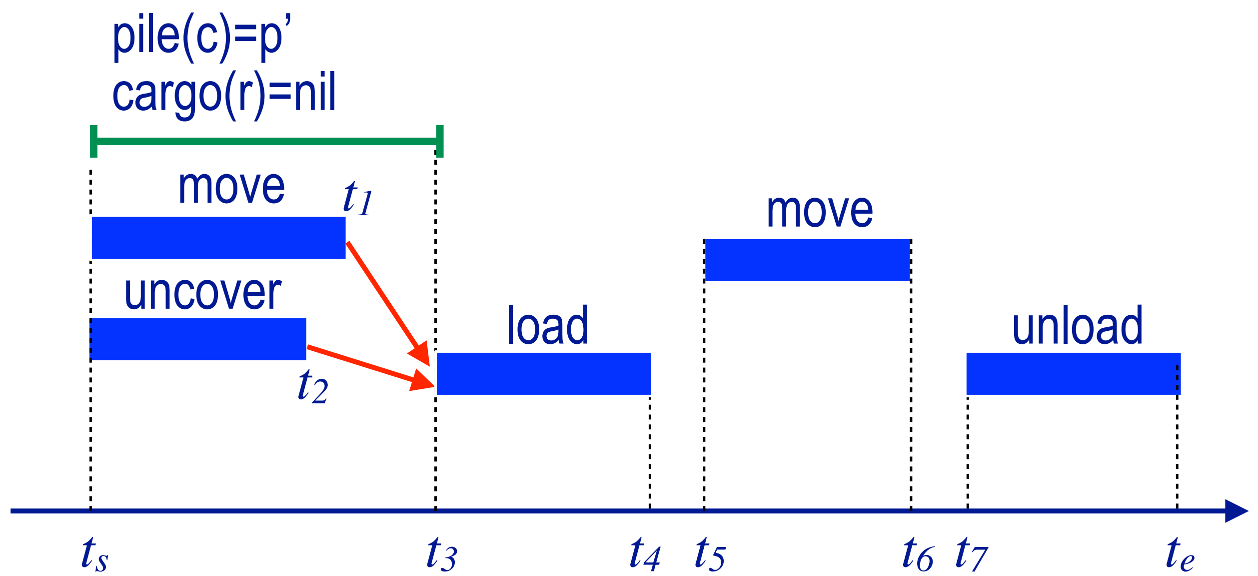
Outline

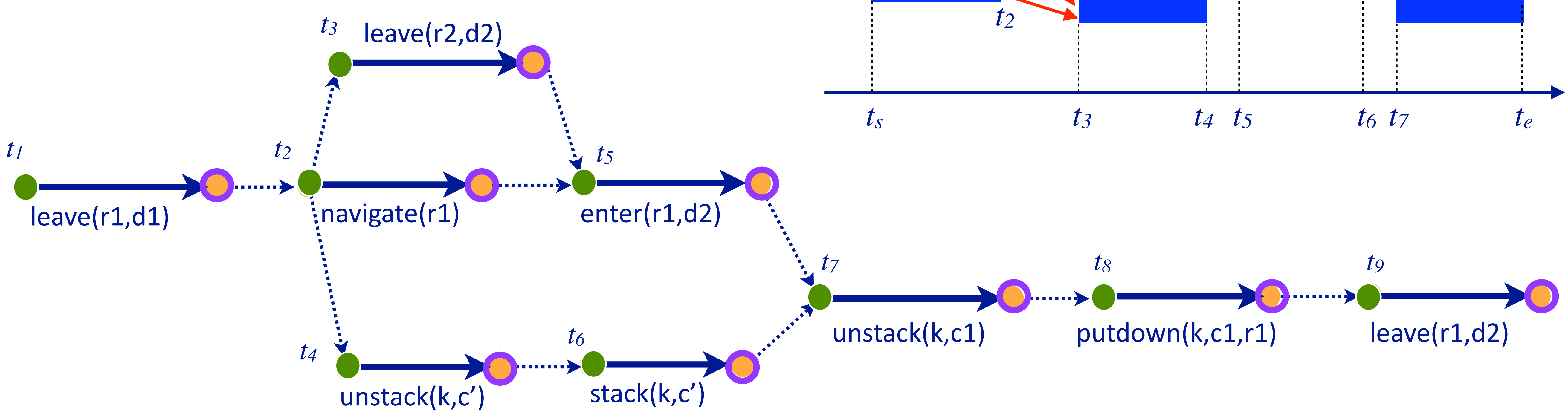
- ✓ Introduction
- ✓ Representation
- ✓ Temporal planning
- ✓ Consistency and controllability
- Acting with executable primitives
  - Dispatching
  - Observation actions
- Acting with atemporal refinement
- Conclusion

Dispatching Algorithm

Problem

- Given a dynamically controllable plan with executable primitives
- Trigger corresponding commands from online observations





Dispatching Algorithm

Plan grounded in realtime: when constrained w.r.t. absolute bounds or when execution starts

- Future point t is bounded with absolute bounds $[l_t, u_t]$
- Past point is instantiated

A controllable time point t that remains in the future

- t is alive if the current time $now \in [l_t, u_t]$
- t is enabled if
  - t is alive,
  - for every precedence constraint $t' < t$, t' has occurred, and
  - for every wait constraint $\langle t_e, \alpha \rangle$, either t_e has occurred or α has expired

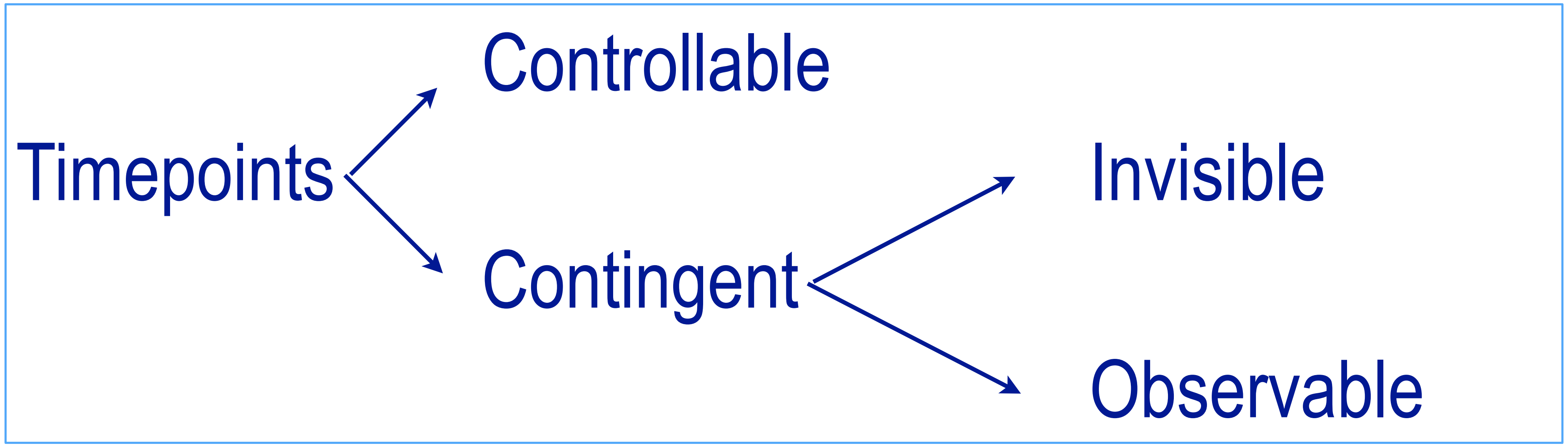
Dispatching Algorithm

Dispatch($plan$)
  initialize the network
  while there are controllable points that have not occurred do
    update now
    update contingent points that have been observed
    $enabled \leftarrow$ set of enabled points
    for every $t \in enabled$ such that $now = u_t$ do
      trigger t
    arbitrarily choose other points in $enabled$; trigger them
    propagate in the network the values of triggered points

- Temporal monitoring

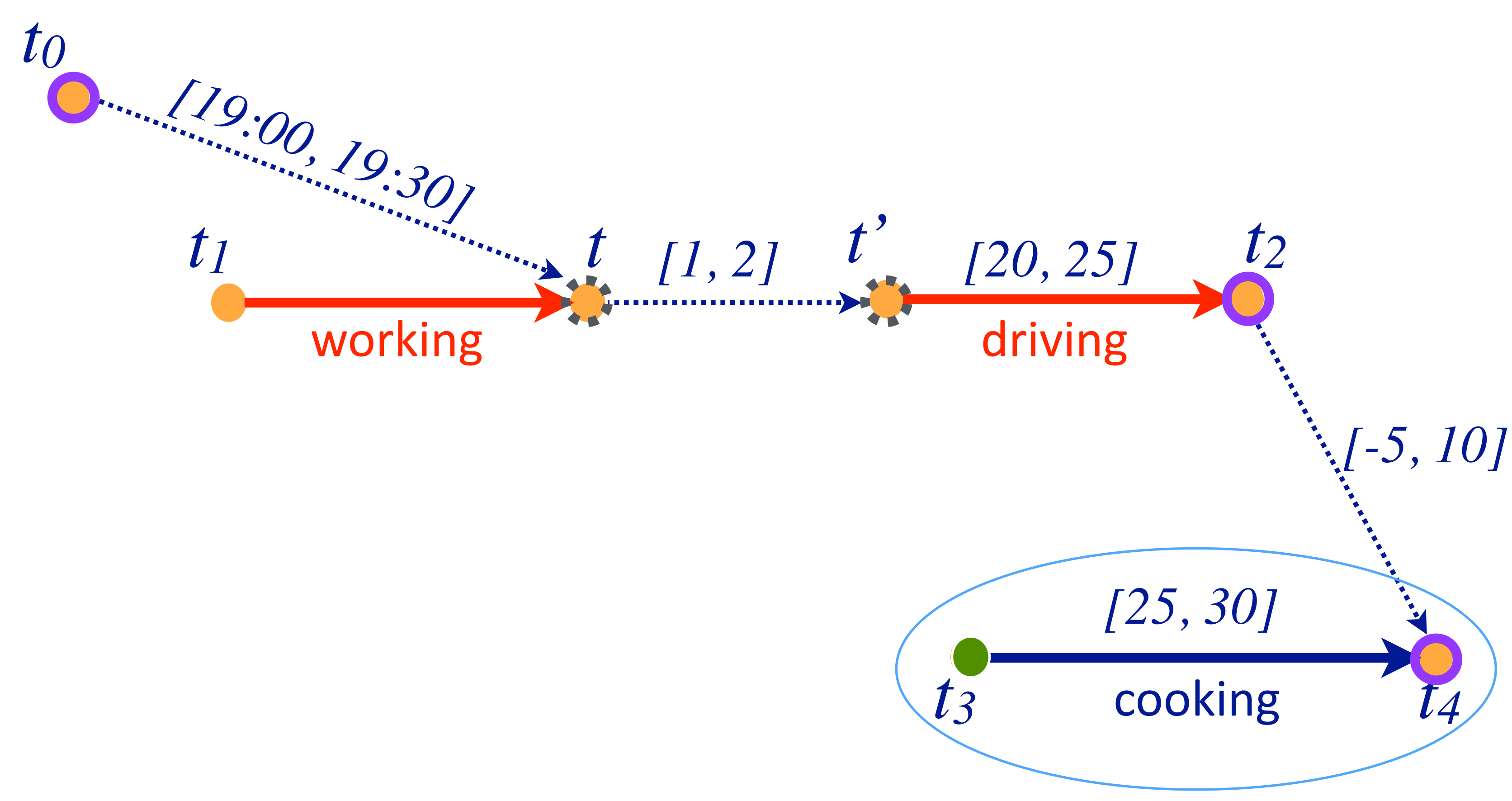
Observation Actions

- Assumption: all occurrences of contingent events are *observable*
  - Observation needed for dynamic controllability
  - In general not all events are observable
- Refining STNU into POSTNU



- Is POSTNU dynamically controllable ?

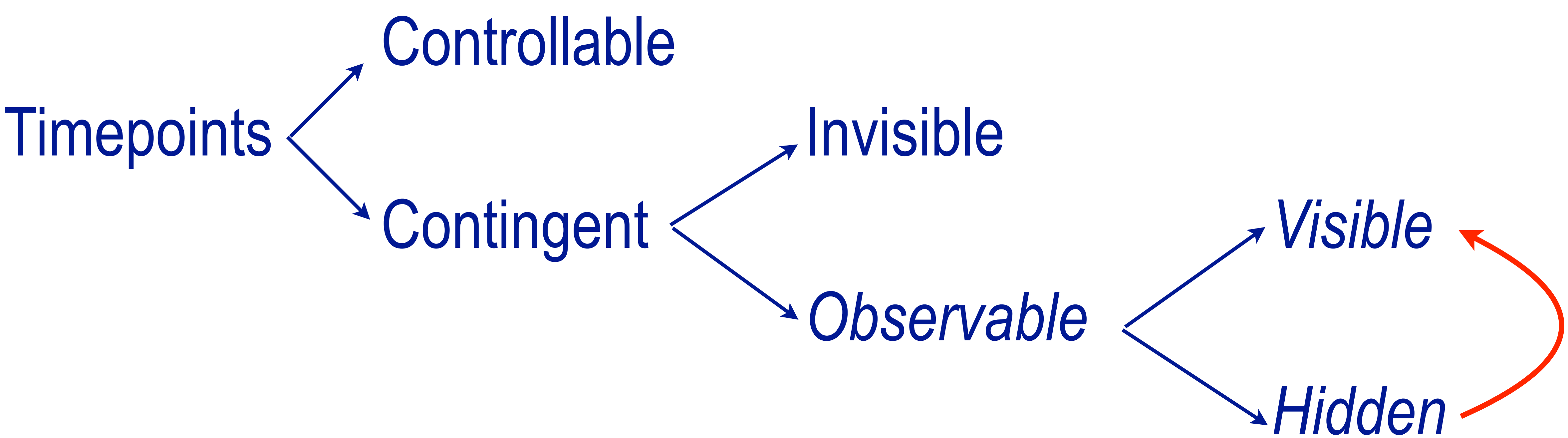
Observation Actions



- Controllable
- Contingent { Invisible / observable

Observation Actions

- POSTNU dynamically controllable if there exists an execution strategy that chooses future controllable points to meet all the contraints, given the observation of past *visible* points
- Observable ≠ visible
  - Observable means *it will be known when observed*
  - It can be temporarily hidden



Outline

- ✓ Introduction
- ✓ Representation
- ✓ Temporal planning
- ✓ Consistency and controllability
- ✓ Acting with executable primitives
- Acting with atemporal refinement
- Conclusion

Atemporal Refinement of Primitive Actions

- Planning primitives are compound tasks at the acting level
- Refined into commands with refinement methods in RAE

$$
\begin{aligned}
&\text{m-leave}(r, d, w, e)\\
&\quad \text{task: } \text{leave}(r, d, w)\\
&\quad \text{pre: } \text{loc}(r){=}d, \text{adjacent}(d, w), \text{exit}(e, d, w)\\
&\quad \text{body: until } \text{empty}(e) \ \text{wait}(1)\\
&\qquad\qquad \text{goto}(r, e)
\end{aligned}
$$

Operational model

Descriptive model

$$
\begin{aligned}
&\text{leave}(r, d, w)\\
&\quad \text{assertions: } [t_s, t_e]\text{loc}(r){:}(d, w)\\
&\qquad\qquad\qquad [t_s, t_e]\text{occupant}(d){:}(r, \text{empty})\\
&\quad \text{constraints: } t_e \le t_s + \delta_1\\
&\qquad\qquad\qquad \text{adjacent}(d, w)
\end{aligned}
$$

Atemporal Refinement of Primitive Actions

- Planning primitives are compound tasks at the acting level
- Refined into commands with refinement methods in RAE

m-unstack(k, c, p)
  task: unstack(k, c, p)
  pre: pos(c)=p, top(p)=c, grip(k)=empty
  attached(k, d), attached(p, d)
  body: locate-grasp-position(k, c, p)
  move-to-grasp-position(k, c, p)
  grasp(k, c, p)
  until firm-grasp(k, c, p) ensure-grasp(k, c, p)
  lift-vertically(k, c, p)
  move-to-neutral-position(k, c, p)

Operational model

Atemporal Refinement

- Pros
  - Simple online refinement with RAE
  - Avoids breaking down uncertainty of contingent duration
  - Can be augmented with temporal monitoring functions in RAE
    e.g., watchdogs, methods with duration preferences
- Cons
  - Does not handle temporal requirements at the command level,
    e.g., concurrency synchronization

Summary

- Rich chronicle representation with temporal refinement
- Planning with chronicle refinement
- Consistency and controllability
- Acting with chronicle dispatching and refinement

  - *ANML modeling language*
  - *FAPE Acting and Planning Environment*

Conclusion

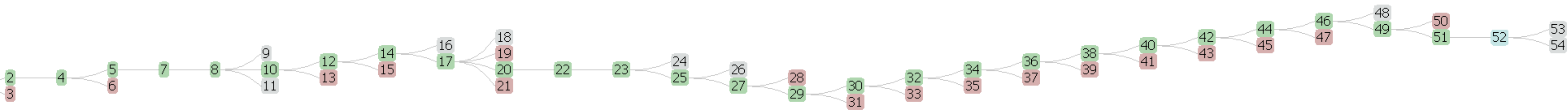
- Temporal models enrich descriptive and operational models of actions
- Chronicle-based approach very flexible for integrating generative and task decomposition techniques
- Acting refinement methods can be extended to integrate temporal construct of chronicles

Example: dwr

```
action uncover (containers c, piles p){
  [start] c.in == p ;
  :decomposition {[all] p.top == c ; };
  :decomposition {
    constant (…) ;
    [start] p.top == prevtop ;
    p != otherp;  c != prevtop ;
    k.attached == d; p.ondock == d ;
    otherp.ondock ==d ;
    [all] p.available == true ;
    [all]  otherp.available == true ;
    [all] contains {
    s1 : unstack(k,prevtop,p) ;
    s2: stack(k,prevtop,otherp);
    s3:   uncover(c,p) ;};
    end(s1) <= start(s2);
    end(s2) <= start(s3); };
};
```

```
action goto (robots r, docks to){
  constant  docks from;
  constant waypoints wa, wb ;
   [start] r.loc == from ;
  :decomposition {from == to ; };
  :decomposition {from != to ;
    adjacent(from, wa) ;
    adjacent(to, wb) ;
    [all] contains {
      s1 : leave(r, from, wa) ;
      s2: navigate(r, wa, wb) ;
      s3 : enter(r, to, wb) ;};
    end(s1) <= start(s2);
    end(s2) <= start(s3);};
};
```

Example: search tree



Example: plan found