Fault Tolerance in Robotics

David Powell, Raja Chatila,
Matthieu Gallien, Jérémie Guiochet,
Félix Ingrand, Marc-Olivier Killijian,
Benjamin Lussier

IFIP WG 10.4, February 16-17, 2006

Introduction

Context

- Prospective internal research project at LAAS
  - Robotics and Artificial Intelligence group
  - Dependable Computing and Fault Tolerance group

➔ *Dependability of autonomous robots in critical applications*

- Space exploration
- Medical assistance
- Service

Introduction

Dependability basics (cf. IEEE TDSC 1(1)11-33, 2004)

- Four complementary means to achieve dependability:
  - fault prevention
  - fault removal

  fault avoidance: how to *aim for* fault-free systems

  - fault forecasting
  - fault tolerance

  fault acceptance: how to *live with* systems that are subject to faults

- Fault classes
  - physical faults (natural hardware faults, environmental effects…)
  - interaction faults (humans, environmental adversities…)
  - development faults (hardware & software bugs)

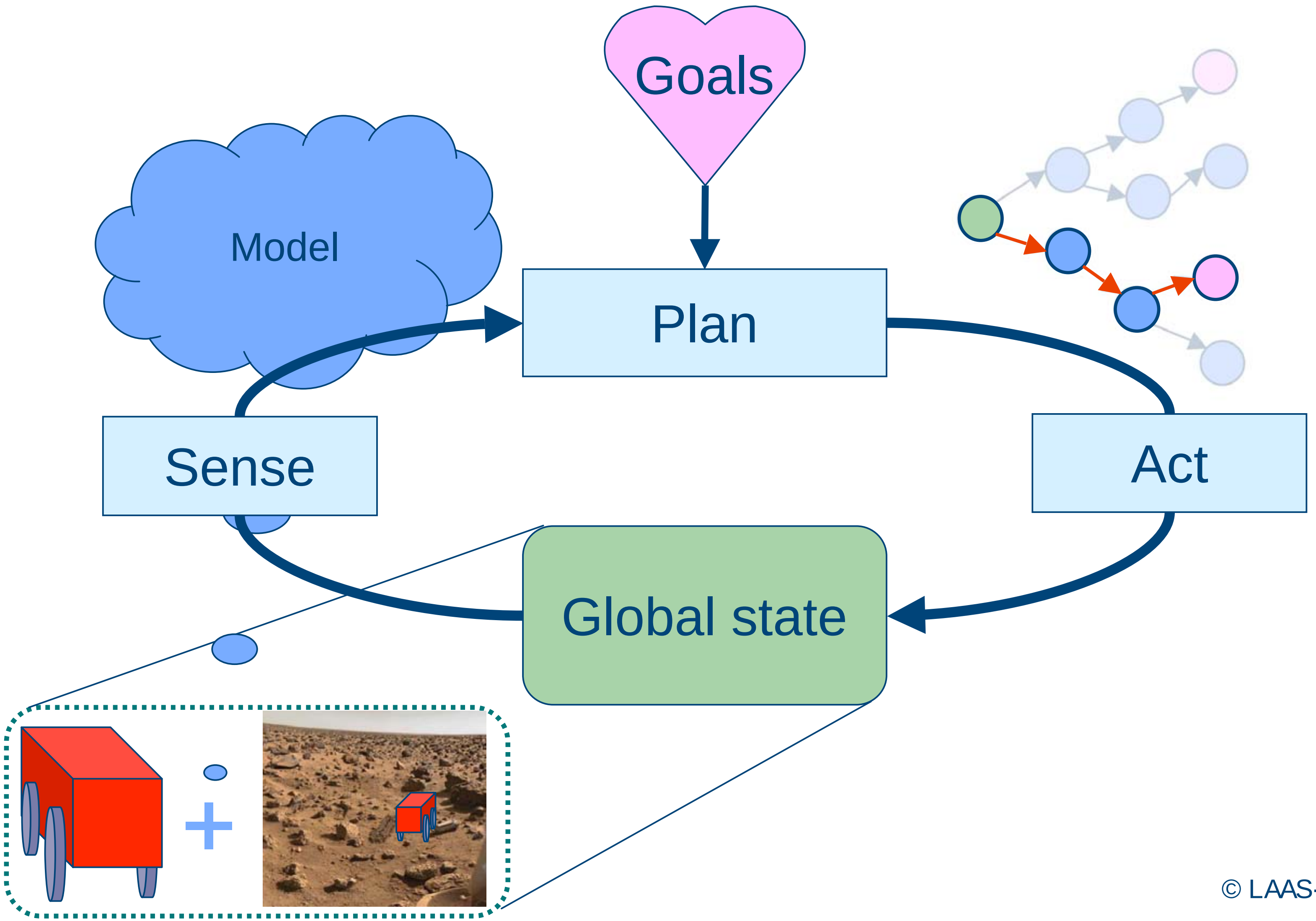
Introduction

Roadmap

- Introduction
- Sense-plan-act paradigm
- Target architecture
- The IxTeT temporal planner
- Tolerating planner faults
- Experimentation environment
- Conclusions

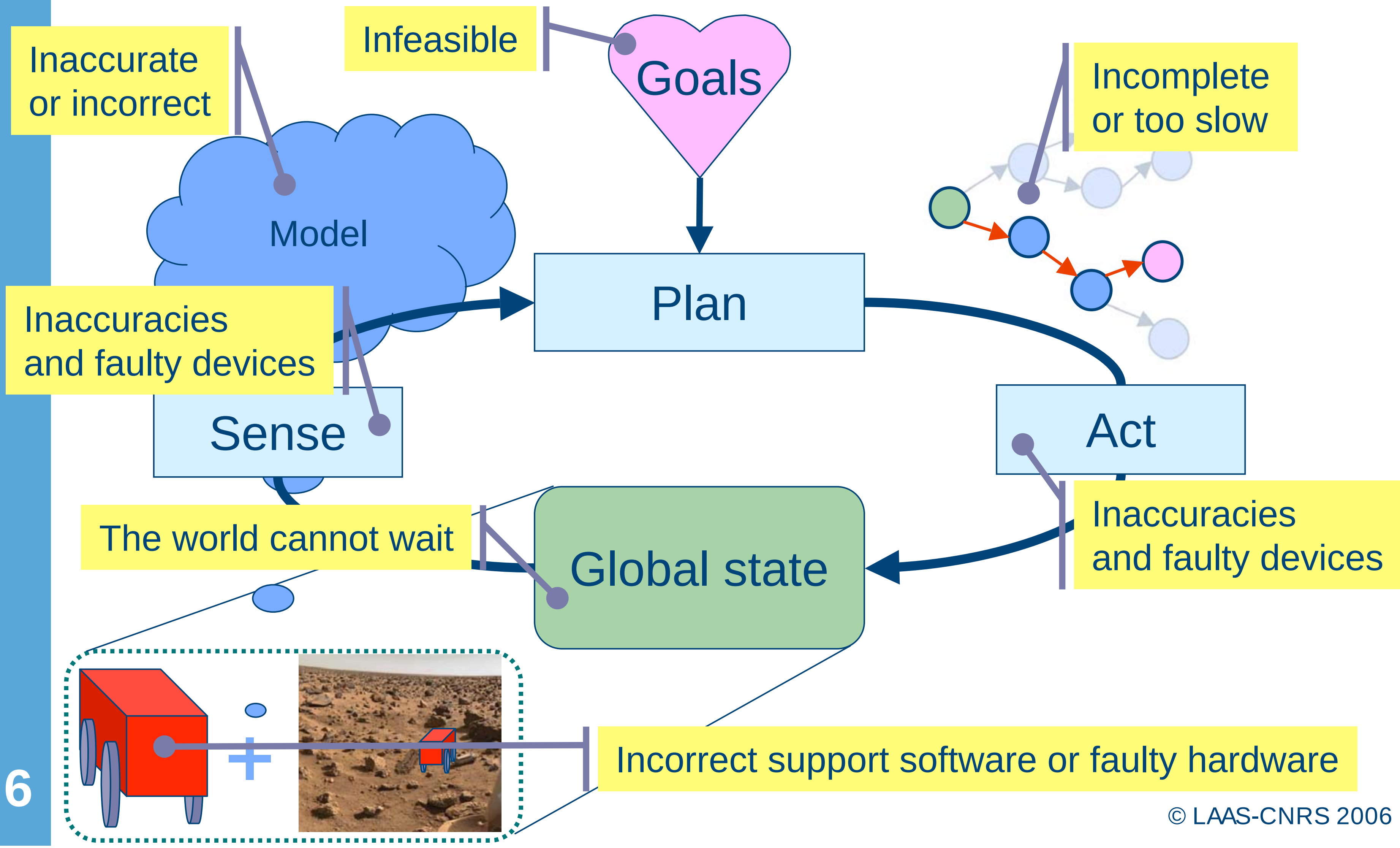
The Sense-Plan-Act Paradigm

Basics



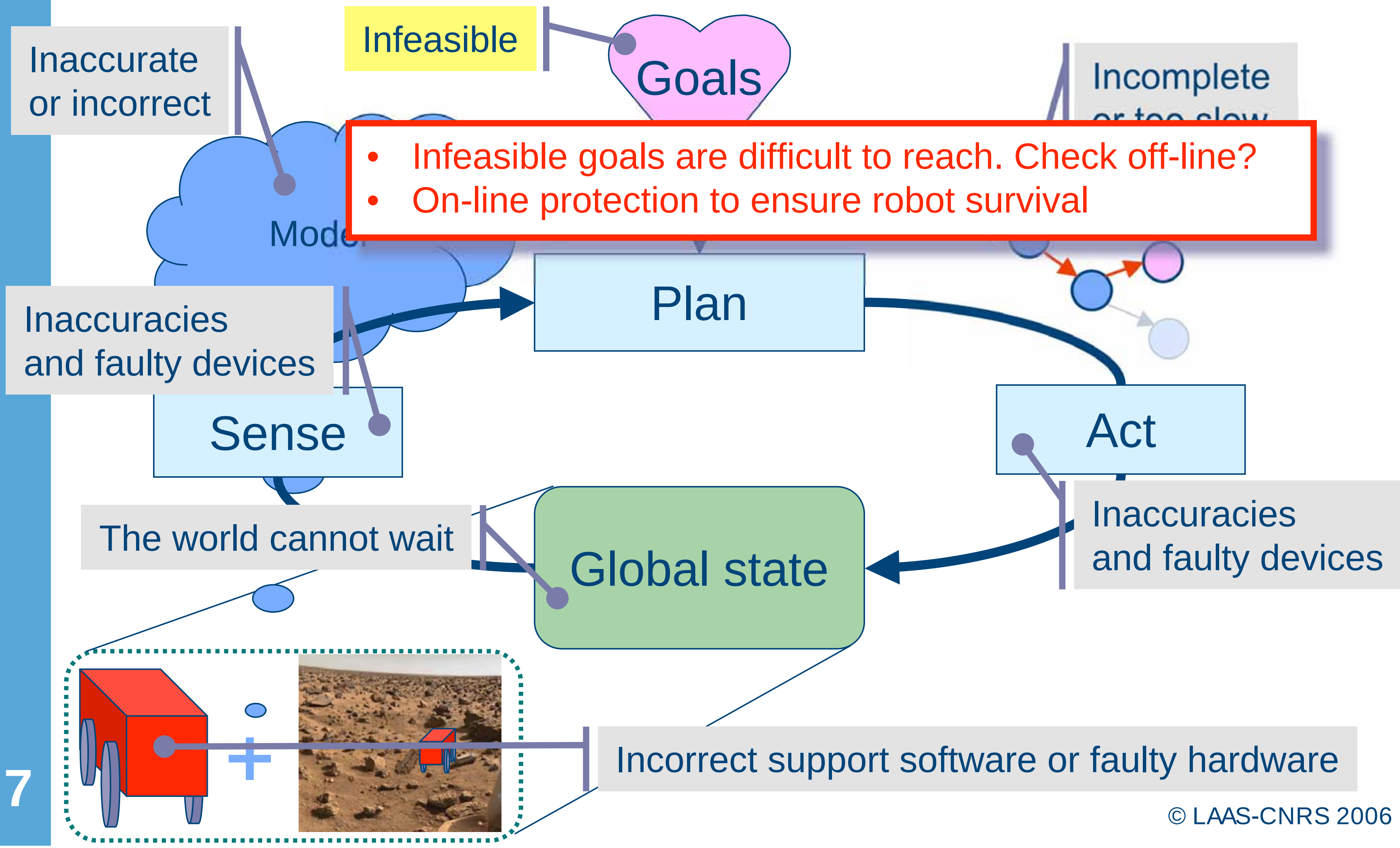
The Sense-Plan-Act Paradigm

What can go wrong ?



The Sense-Plan-Act Paradigm

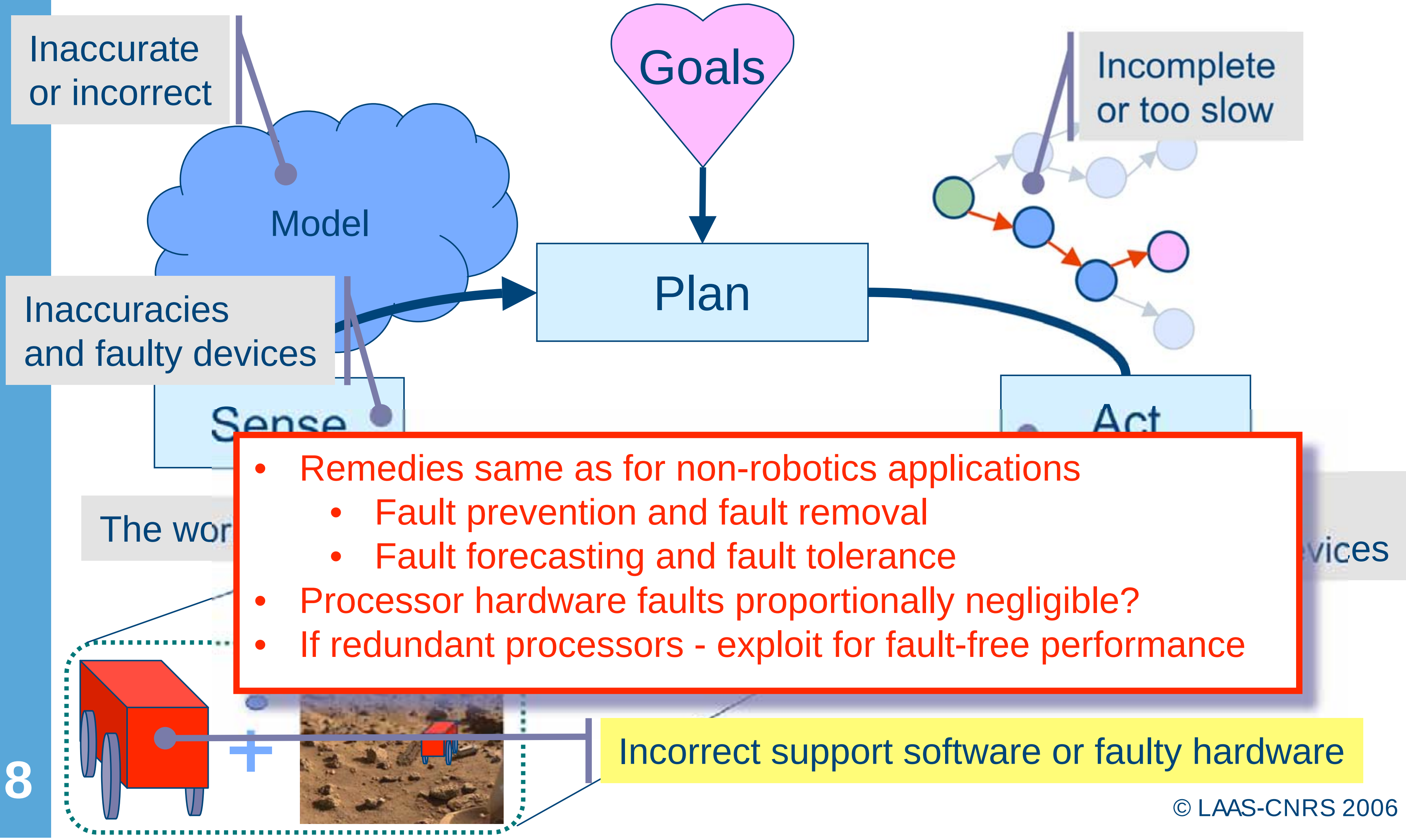
What can go wrong and what can be done about it ?



- Infeasible goals are difficult to reach. Check off-line?
- On-line protection to ensure robot survival

The Sense-Plan-Act Paradigm

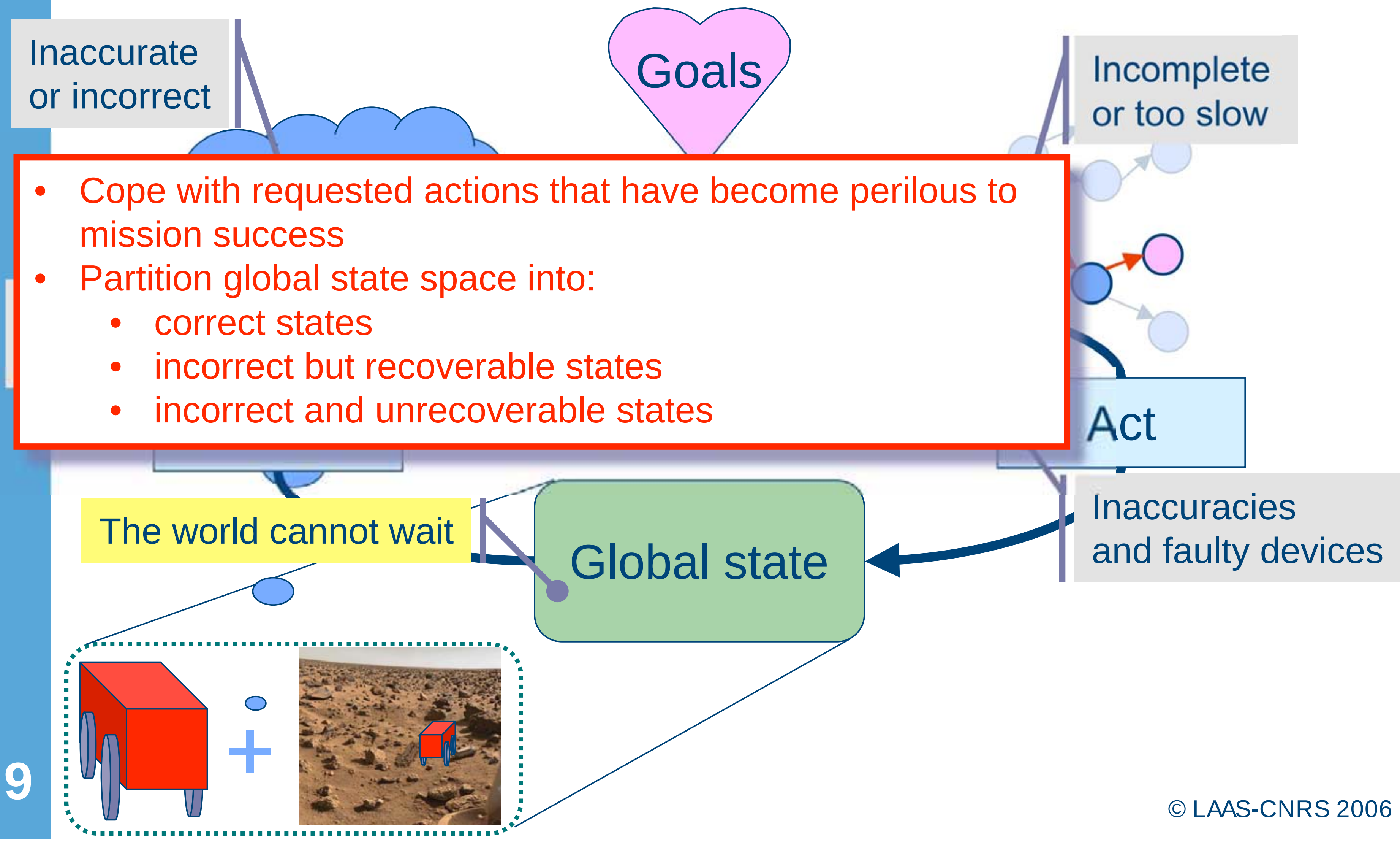
What can go wrong and what can be done about it ?



- Remedies same as for non-robotics applications
  - Fault prevention and fault removal
  - Fault forecasting and fault tolerance
- Processor hardware faults proportionally negligible?
- If redundant processors - exploit for fault-free performance

The Sense-Plan-Act Paradigm

What can go wrong and what can be done about it ?

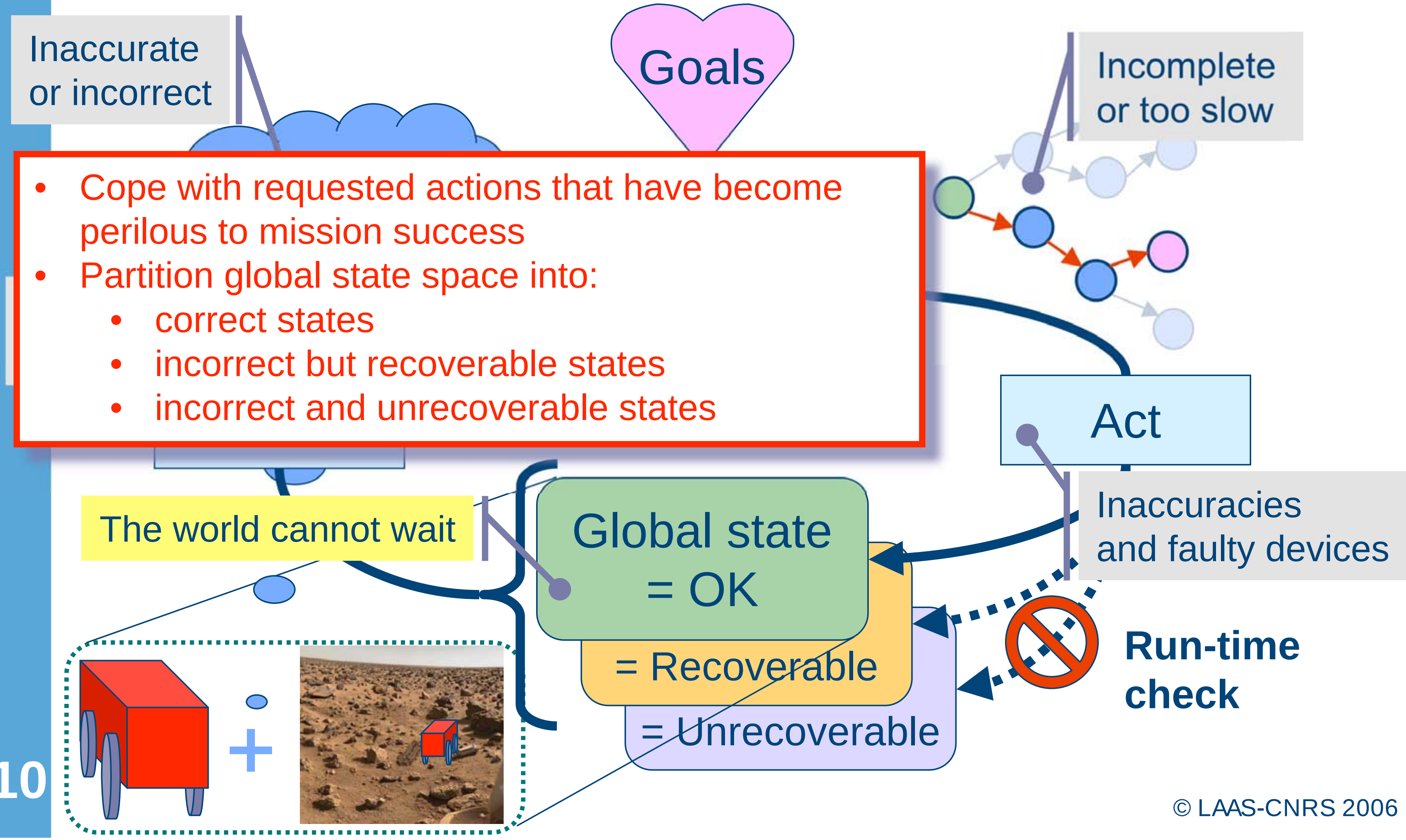


- Cope with requested actions that have become perilous to mission success
- Partition global state space into:
  - correct states
  - incorrect but recoverable states
  - incorrect and unrecoverable states

The Sense-Plan-Act Paradigm

What can go wrong and what can be done about it ?

- Cope with requested actions that have become perilous to mission success
- Partition global state space into:
  - correct states
  - incorrect but recoverable states
  - incorrect and unrecoverable states

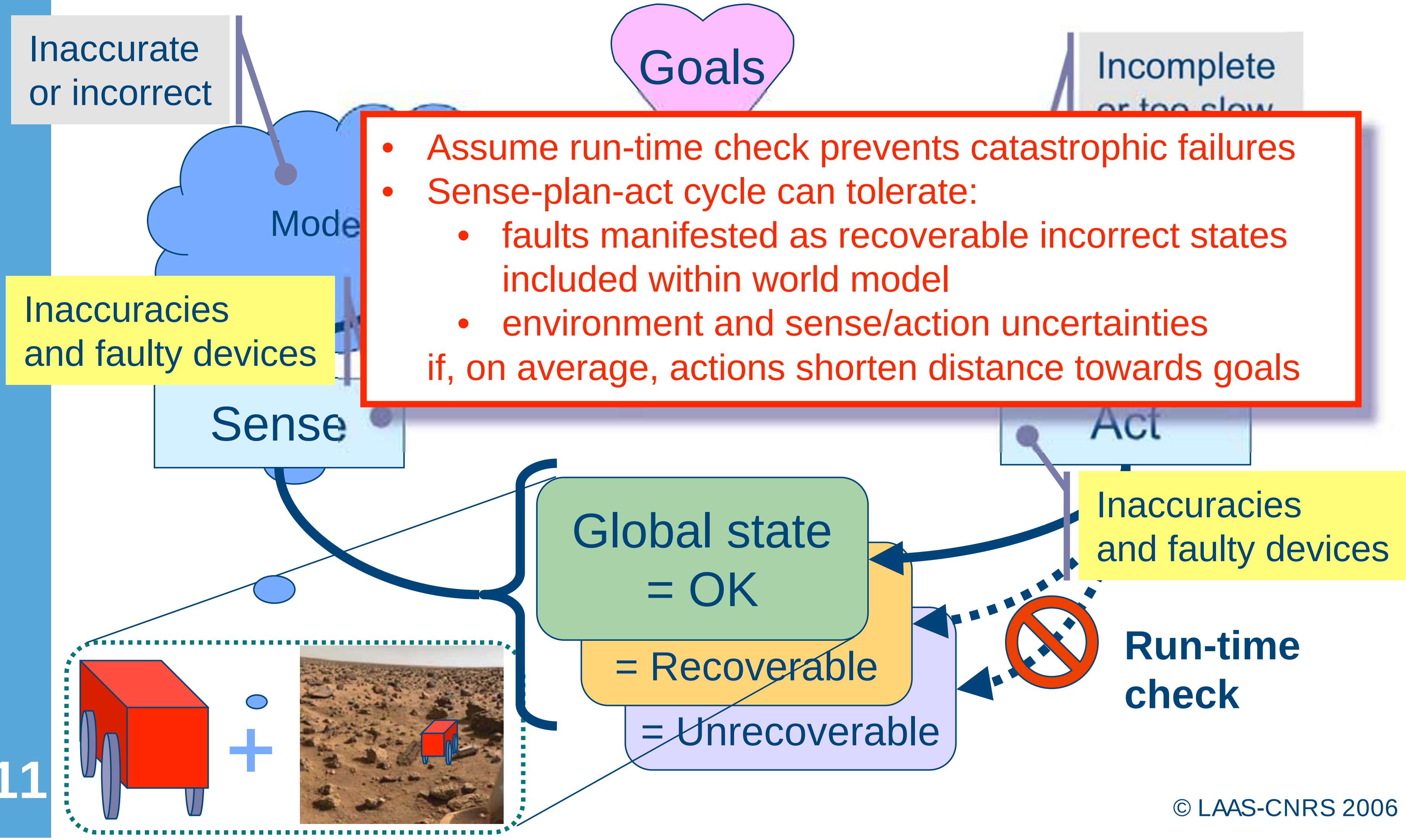


The Sense-Plan-Act Paradigm

What can go wrong and what can be done about it ?

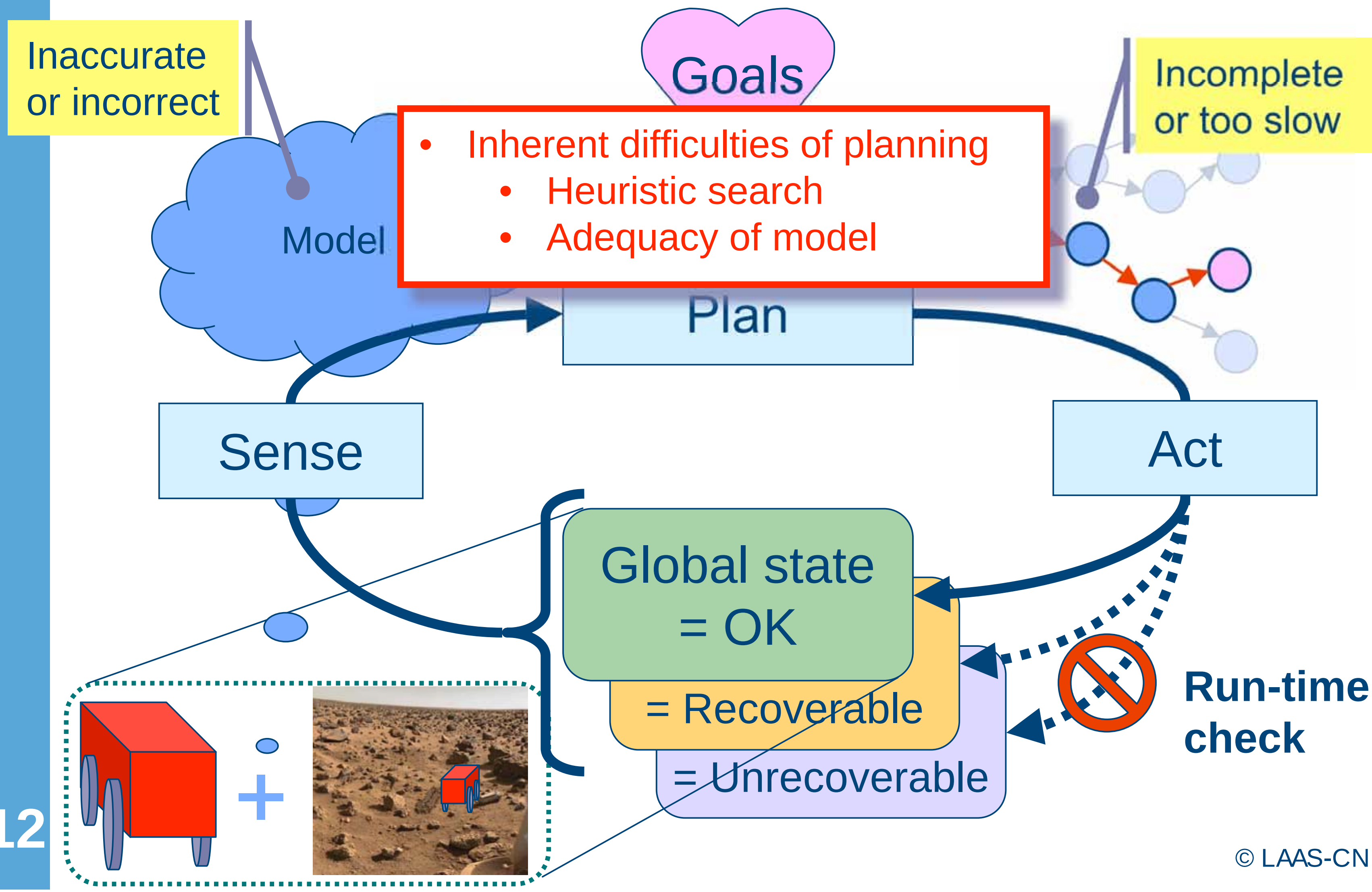
- Assume run-time check prevents catastrophic failures
- Sense-plan-act cycle can tolerate:
  - faults manifested as recoverable incorrect states included within world model
  - environment and sense/action uncertainties

  if, on average, actions shorten distance towards goals



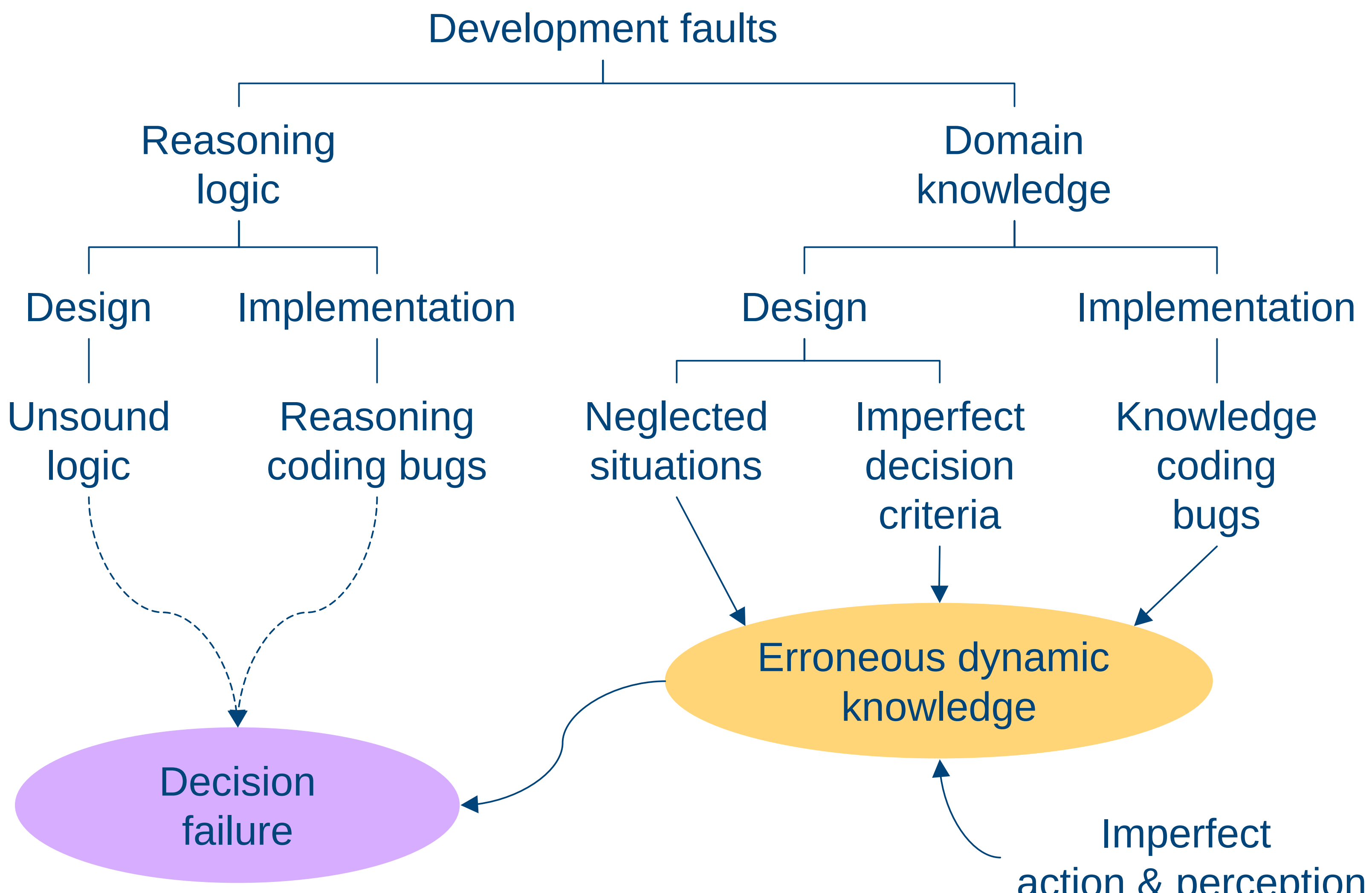
The Sense-Plan-Act Paradigm

What can go wrong and what can be done about it ?



The Sense-Plan-Act Paradigm

Fault and errors affecting decisional mechanisms



Target Architecture

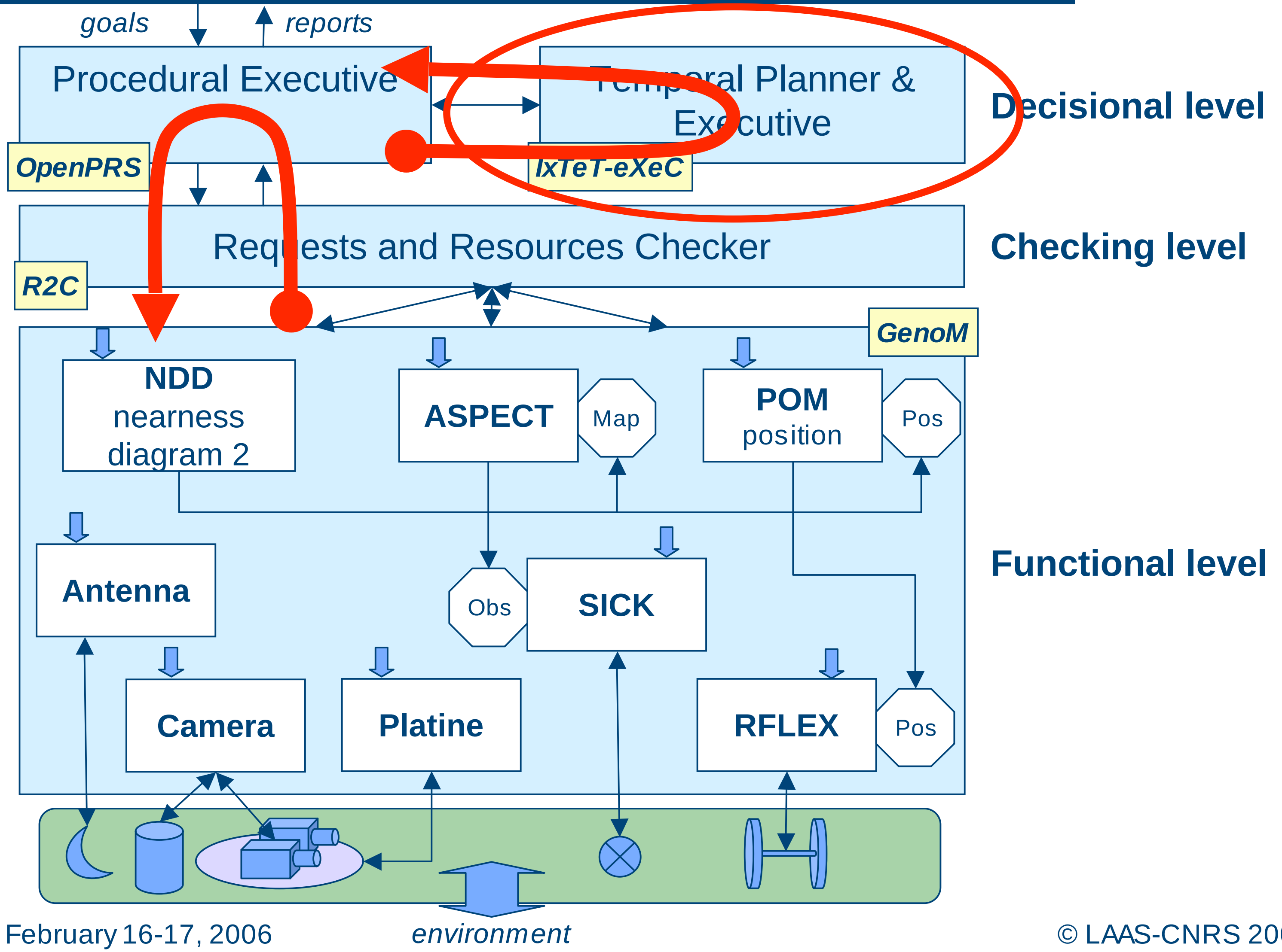
Example: the "Dala" planetary exploration rover

- Commercial ATRV robot platform
- Sensors
  - odometer
  - stereo cameras
  - laser range-finder
- Actuators
  - wheels (differential drive)
  - camera bench Pan & Tilt Unit (PTU)

+ simulated communication facility

Target Architecture

LAAS architecture for autonomous systems (LAAS)

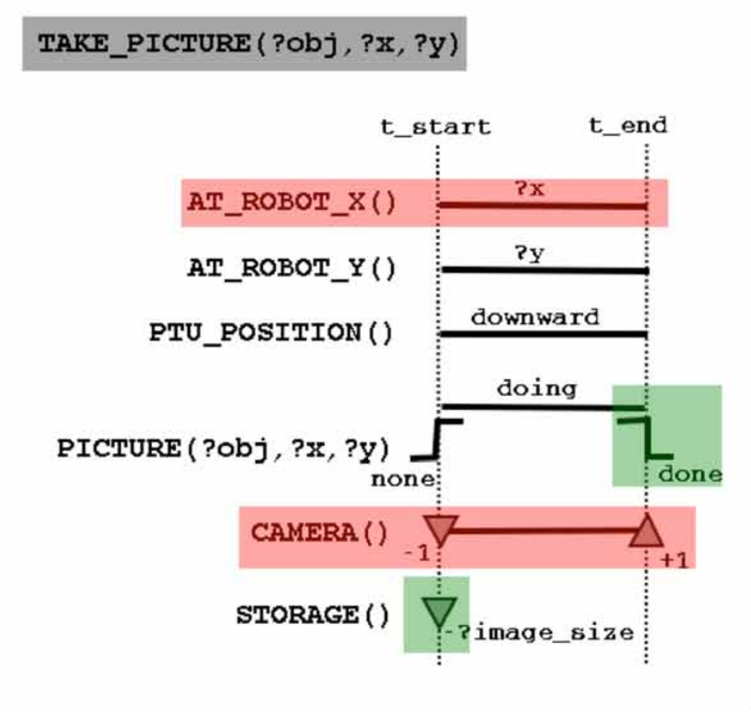


The IxTeT Temporal Planner

Model based on temporal statements about attributes

- **Logical attributes** (hold/event)
  Resource attributes (use/consume/produce)
- **Actions** (specification of the evolution of attributes of interest)

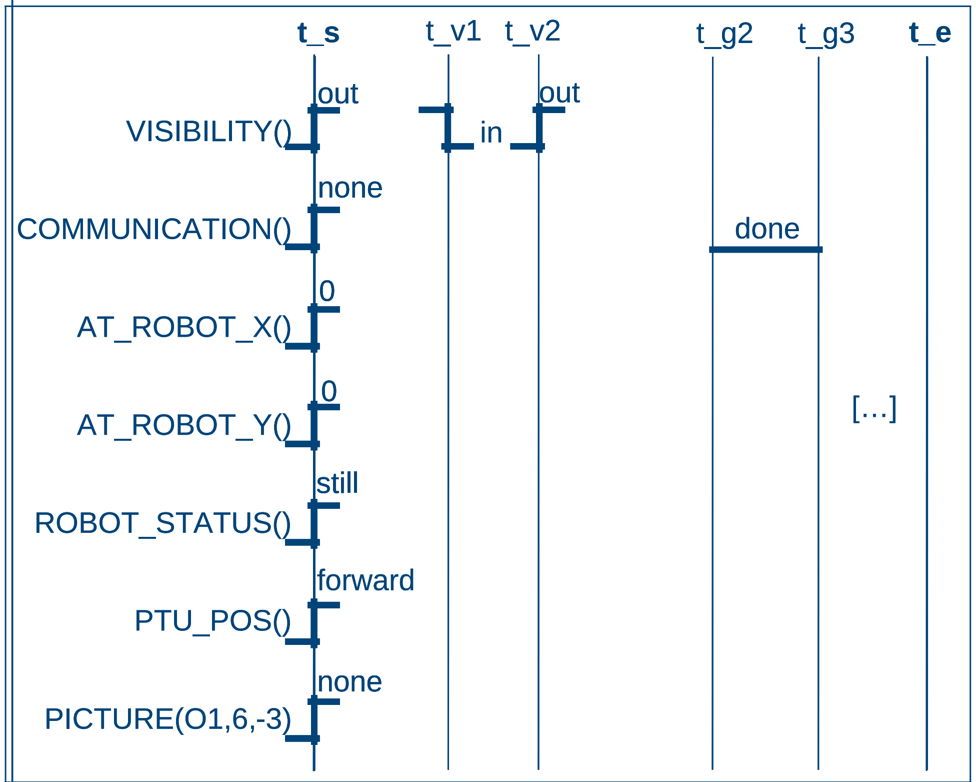
```
task TAKE_PICTURE(?obj,?x,?y)(t_start,t_end){
   ?obj in OBJECTS;
   ?x in ]-oo,+oo[; ?y in ]-oo,+oo[;

  hold(AT_ROBOT_X():?x,(t_start,t_end));
  hold(AT_ROBOT_Y():?y,(t_start,t_end));
  hold(PTU_POSITION():downward,(t_start,t_end));

  event(PICTURE(?obj,?x,?y):(none,doing),t_start);
  hold(PICTURE(?obj,?x,?y):doing,(t_start,t_end));
  event(PICTURE(?obj,?x,?y):(doing,done),t_end);

  use(CAMERA():1,(t_start,t_end));
  variable ?image_size;
  variable ?cr;
  compression_rate(?cr);
  ?image_size = 175610 * ?cr;
  consume(STORAGE():?image_size,t_start);

   (t_end - t_start) in ]0,60];
}nonPreemptive
```



The IxTeT Temporal Planner

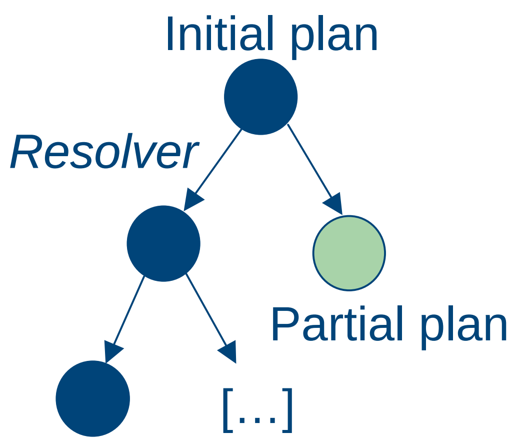
POCL (Partial Order Causal Link) planning

```
task Init() (t_s,t_e){
  timepoint t_v1, t_v2,t_g1,…;
  // Initial State
  explained event(AT_ROBOT_X():(?,0),t_s);
  explained event(AT_ROBOT_Y():(?,0),t_s);
  explained event(ROBOT_STATUS():(?,still),t_s);
  explained event(PTU_POS():(?,forward),t_s);
  explained event(COMMUNICATION():(?,none),t_s);
  variable ?x1,?y1;
  ?x1 in ]-oo,+oo[; ?y1 in ]-oo,+oo[;
  explained event(PICTURE(O1,?x1,?y1):(?,none),t_s);
  // Visibility window
  contingent event(VISIBILITY():(?,out),t_s);
  contingent event(VISIBILITY():(out,in),t_v1);
  contingent event(VISIBILITY():(in,out),t_v2);
  (t_v2-t_v1) in [120,120];
  (t_v1-t_s) in [300,300];
  // Goals
  hold(AT_ROBOT_X():0,(t_g1,t_e)) goal(3,0);
  hold(AT_ROBOT_Y():0,(t_g1,t_e)) goal(3,0);
  hold(COMMUNICATION():done,(t_g2,t_g3)) goal(2,0);
  hold(PICTURE(O1,6,-3):done,(t_g4,t_g5)) goal(1,0);
  // Horizon
  (t_e – t_s) in [1000,1000];
} latePreemptive
```



The IxTeT Temporal Planner

POCL (Partial Order Causal Link) planning



Def. [partial plan] **P = (A,C,L,F)**

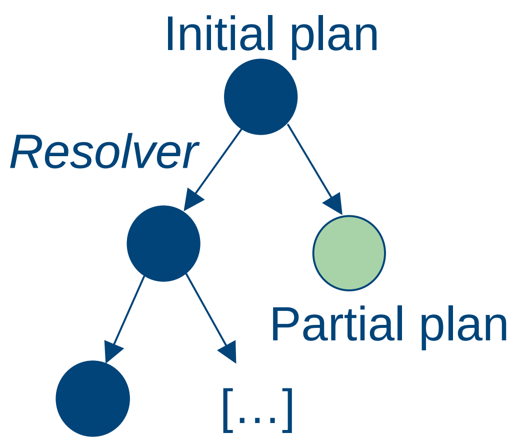
A : *actions*
C : *constraints*
L : *causal links*
F : *flaws*

- open conditions
- threats
- resource conflicts

A partial plan is a solution plan if F = Ø

The IxTeT Temporal Planner

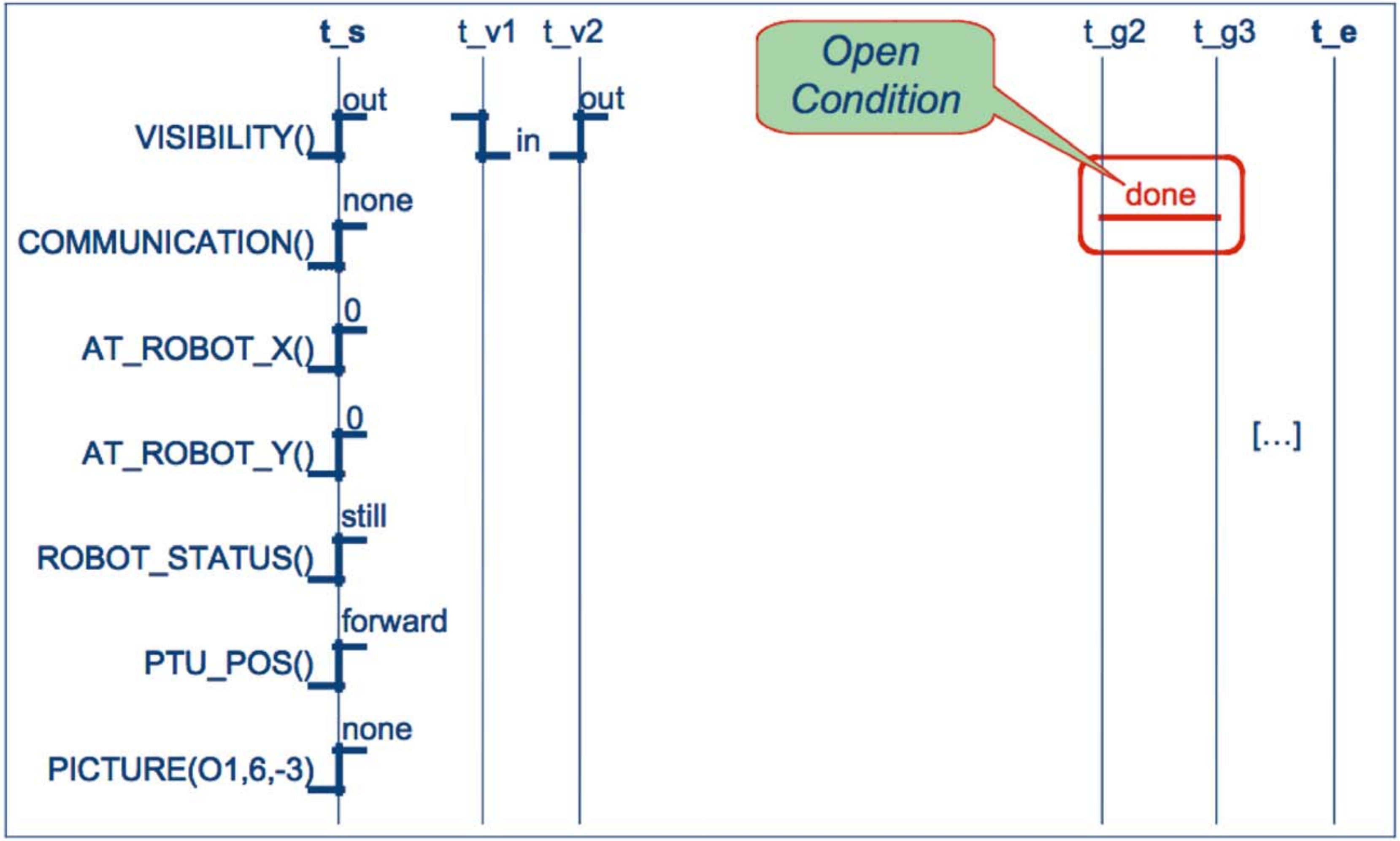
POCL (Partial Order Causal Link) planning



- **A planning step**
  - **Analyze** ⇒ flaws + resolvers
    - Open Conditions
      - Establishing event + causal link
    - Threats
      - Precedence constraint or variable binding
    - Resource Conflicts
      - Precedence constraint or action insertion
  - **Flaw selection**
    - Abstraction hierarchy
    - Least commitment
  - **Resolver selection**
    - A_ε algorithm or Ordered Depth-first search
  - **Resolver insertion**

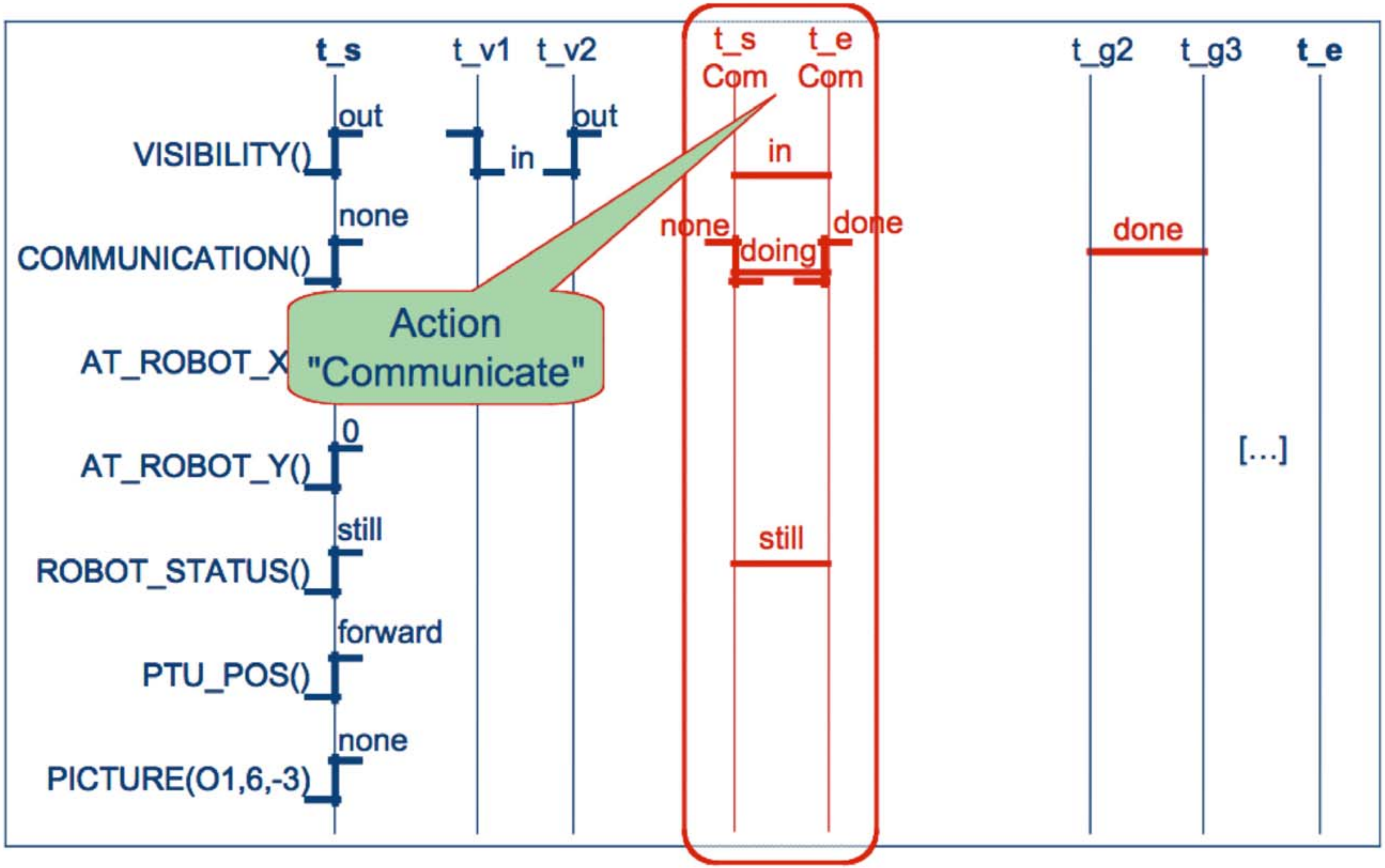
The IxTeT Temporal Planner

POCL (Partial Order Causal Link) planning



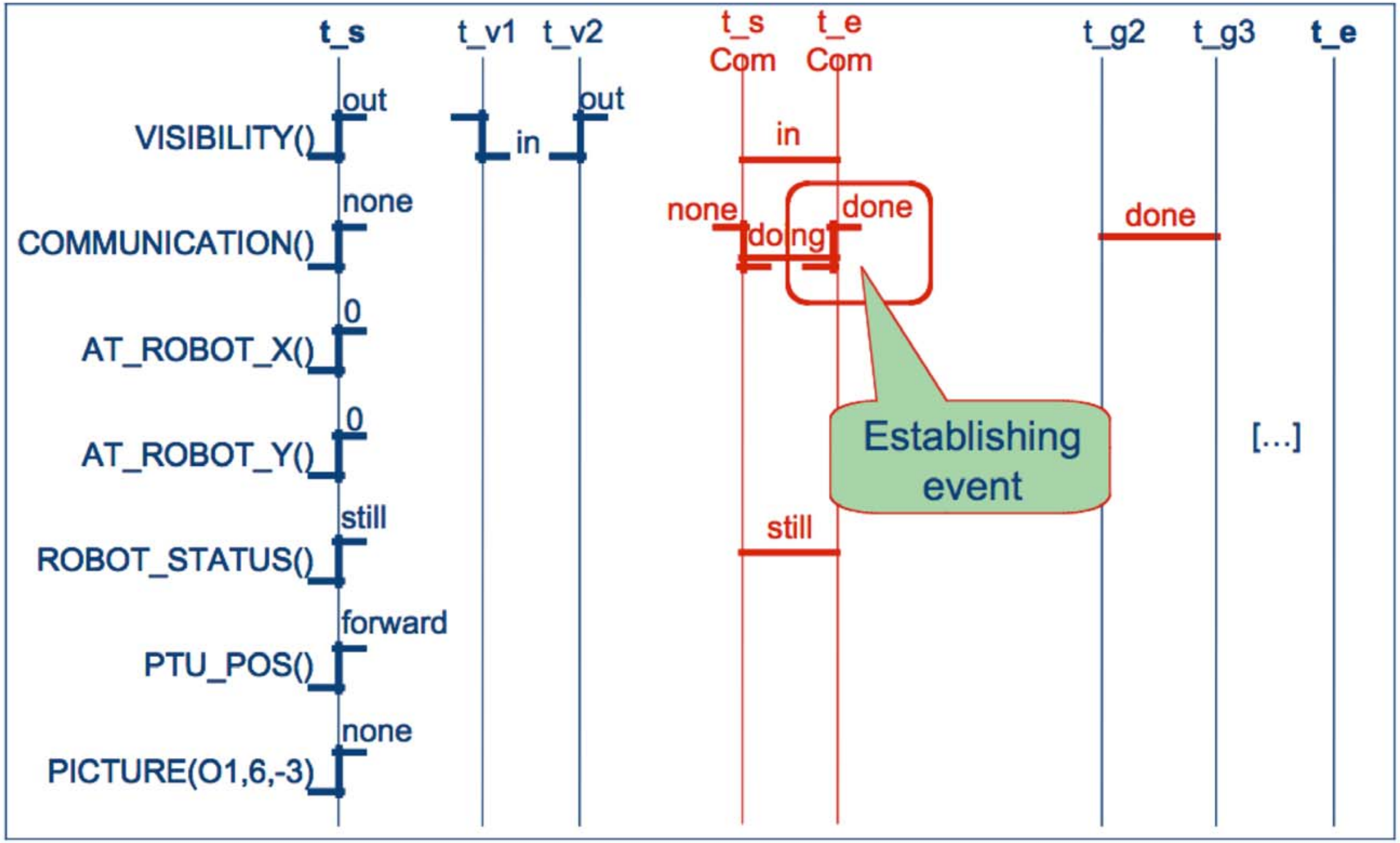
The IxTeT Temporal Planner

POCL (Partial Order Causal Link) planning



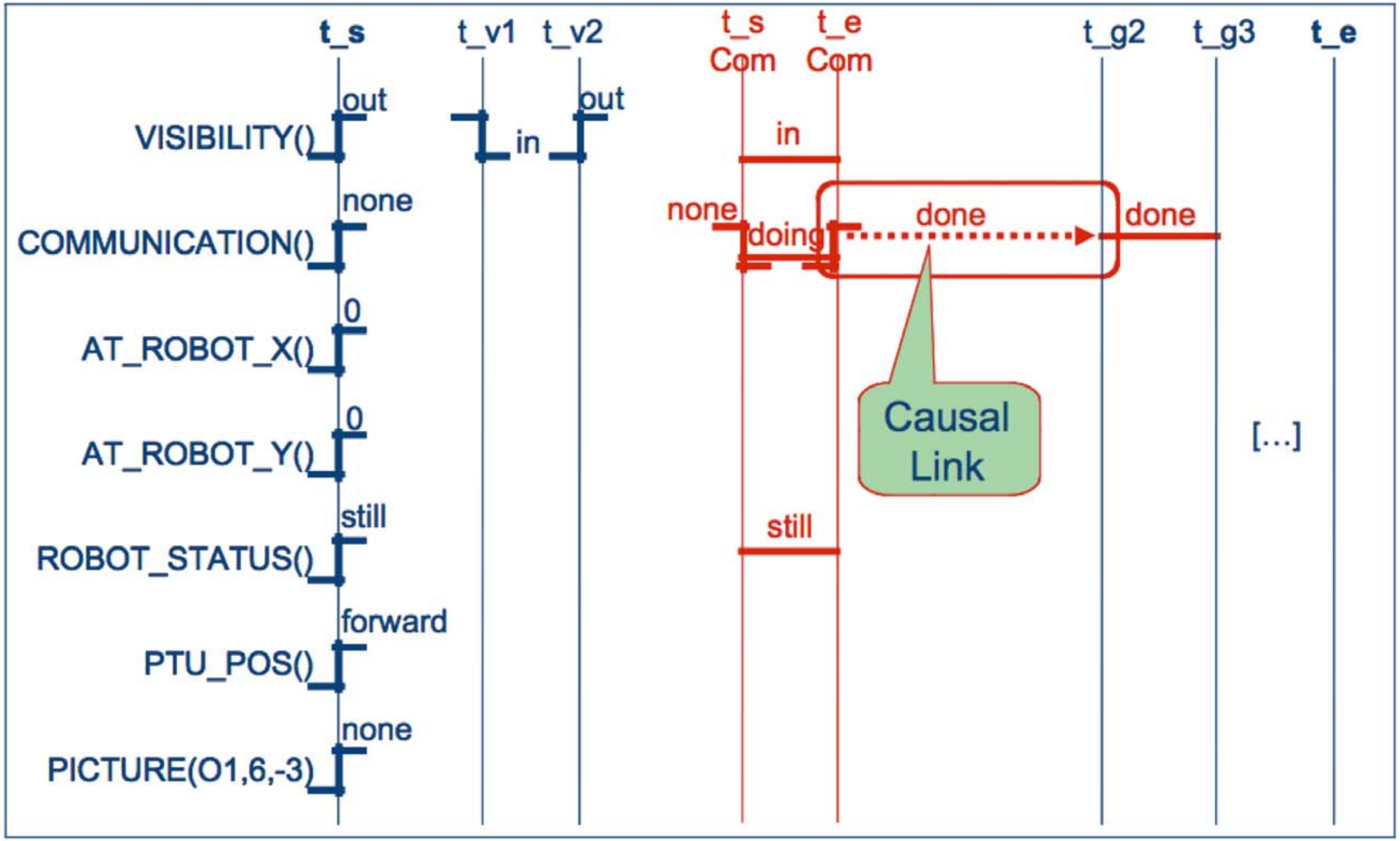
The IxTeT Temporal Planner

POCL (Partial Order Causal Link) planning



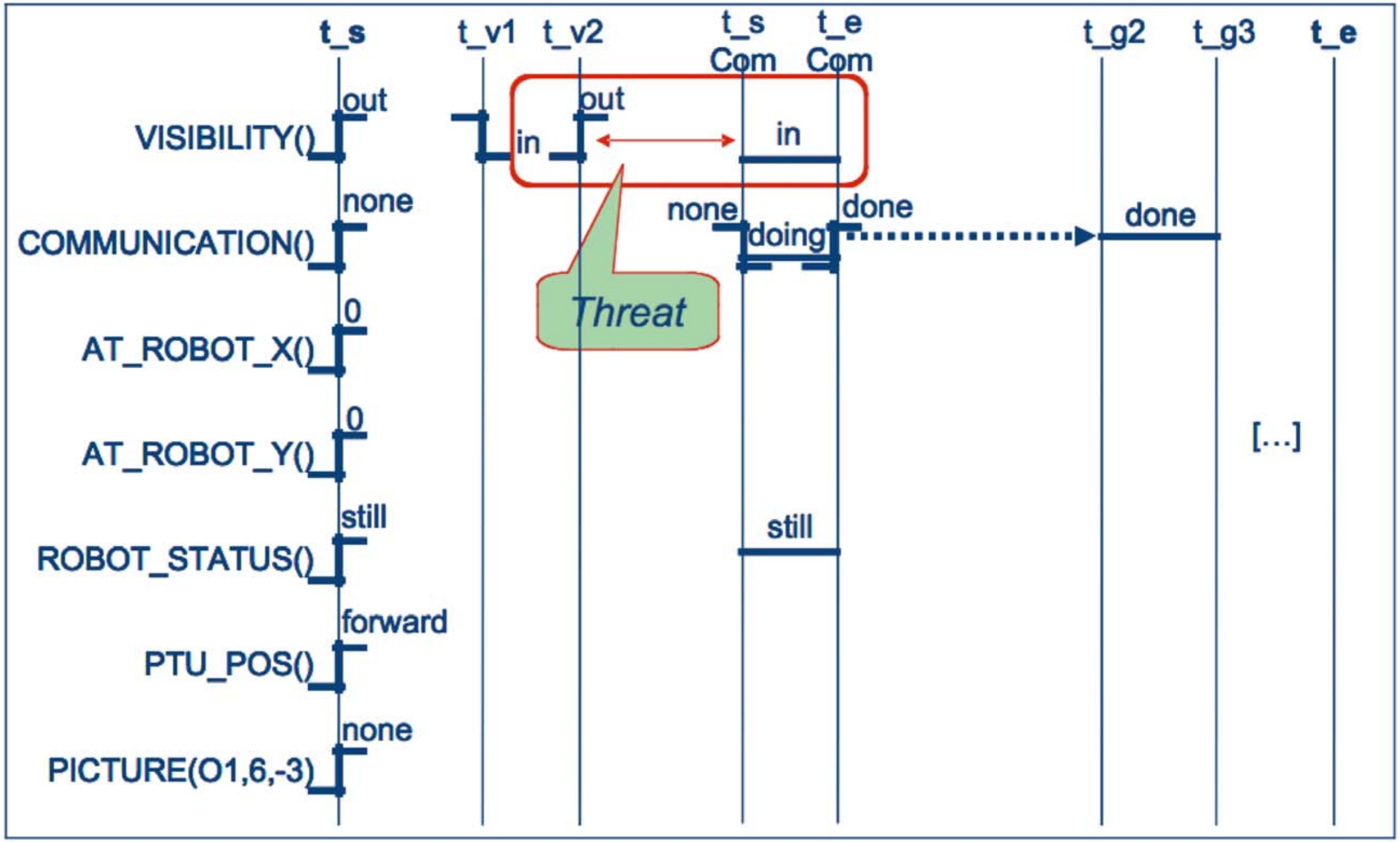
The IxTeT Temporal Planner

POCL (Partial Order Causal Link) planning



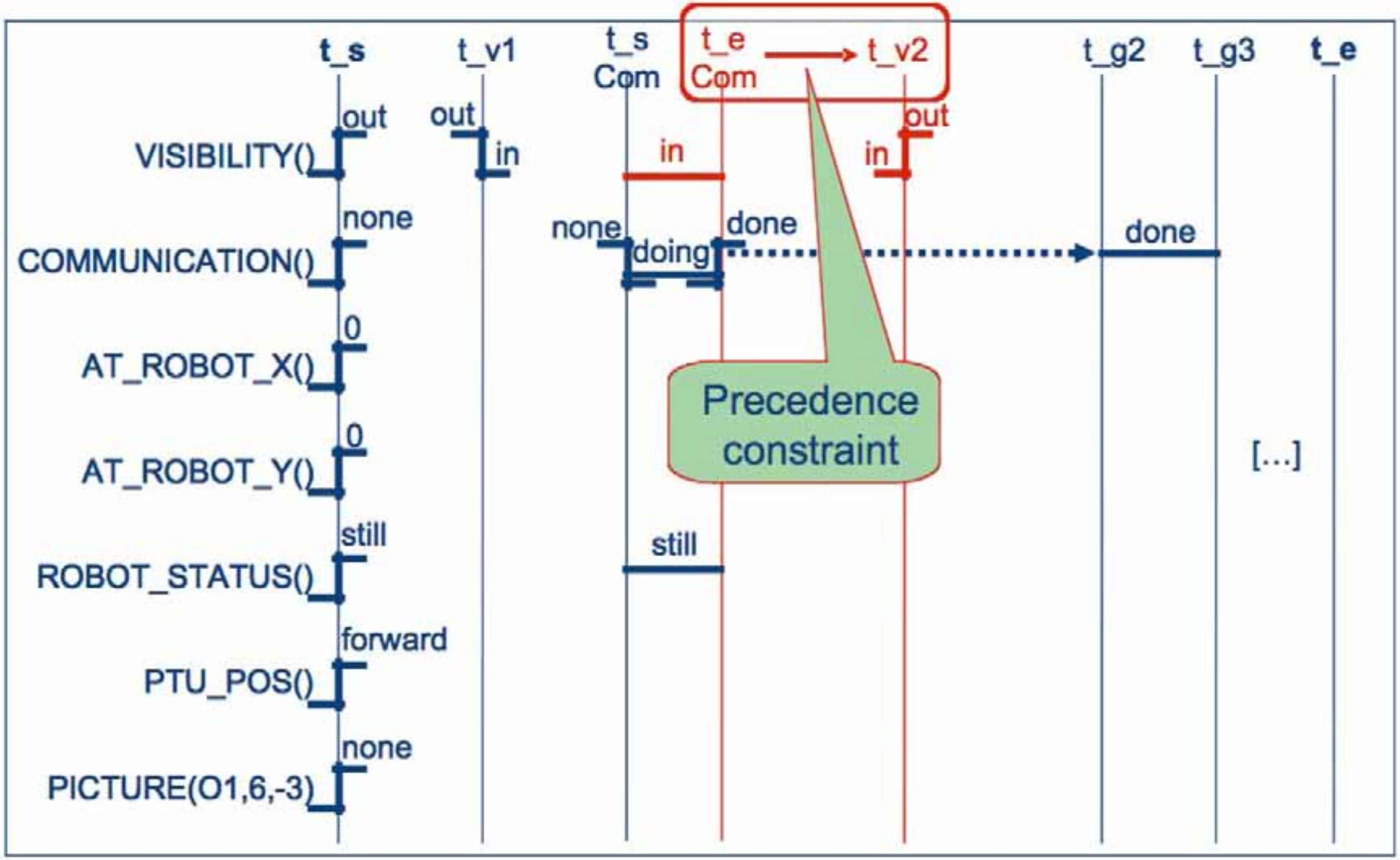
The IxTeT Temporal Planner

POCL (Partial Order Causal Link) planning



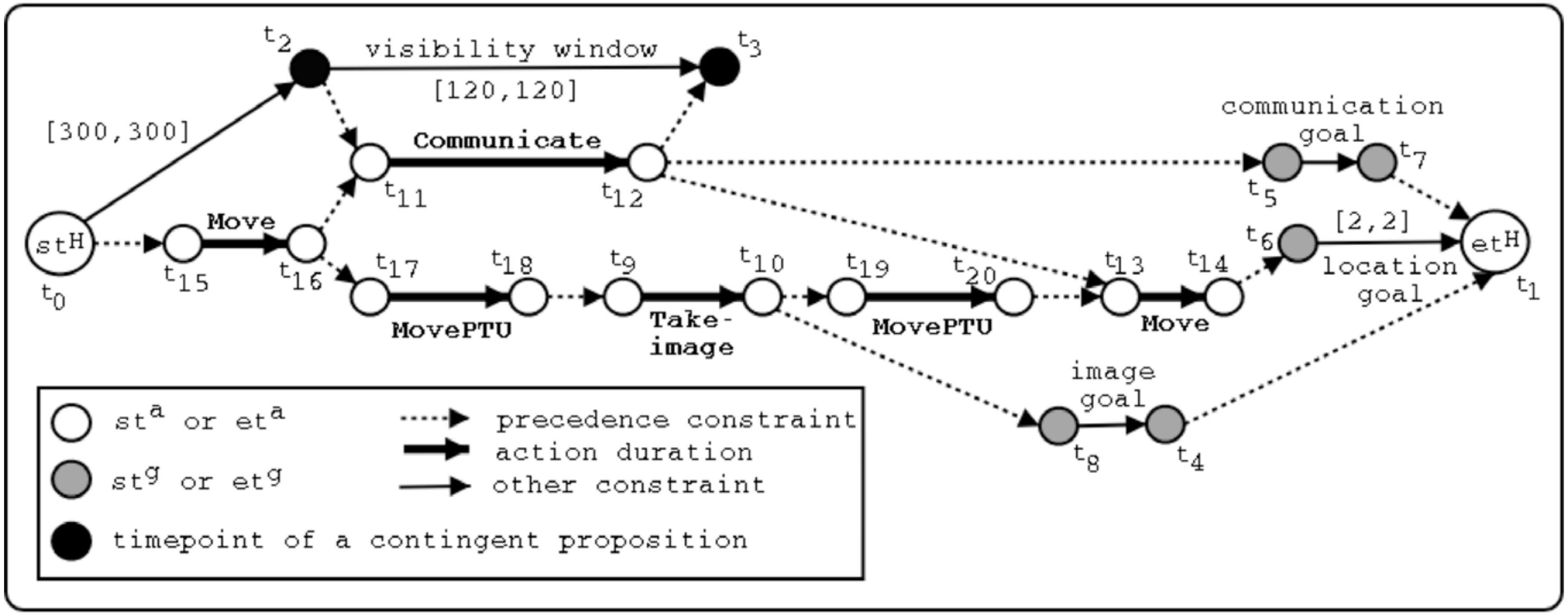
The IxTeT Temporal Planner

POCL (Partial Order Causal Link) planning



The IxTeT Temporal Planner

Plan database: temporal constraint network



- Execution interval of timepoint T: (T-st^H) in [T_{lb},T_{ub}]
- Start / stop / monitor action execution according to type
- Actual occurrence times propagated to update network

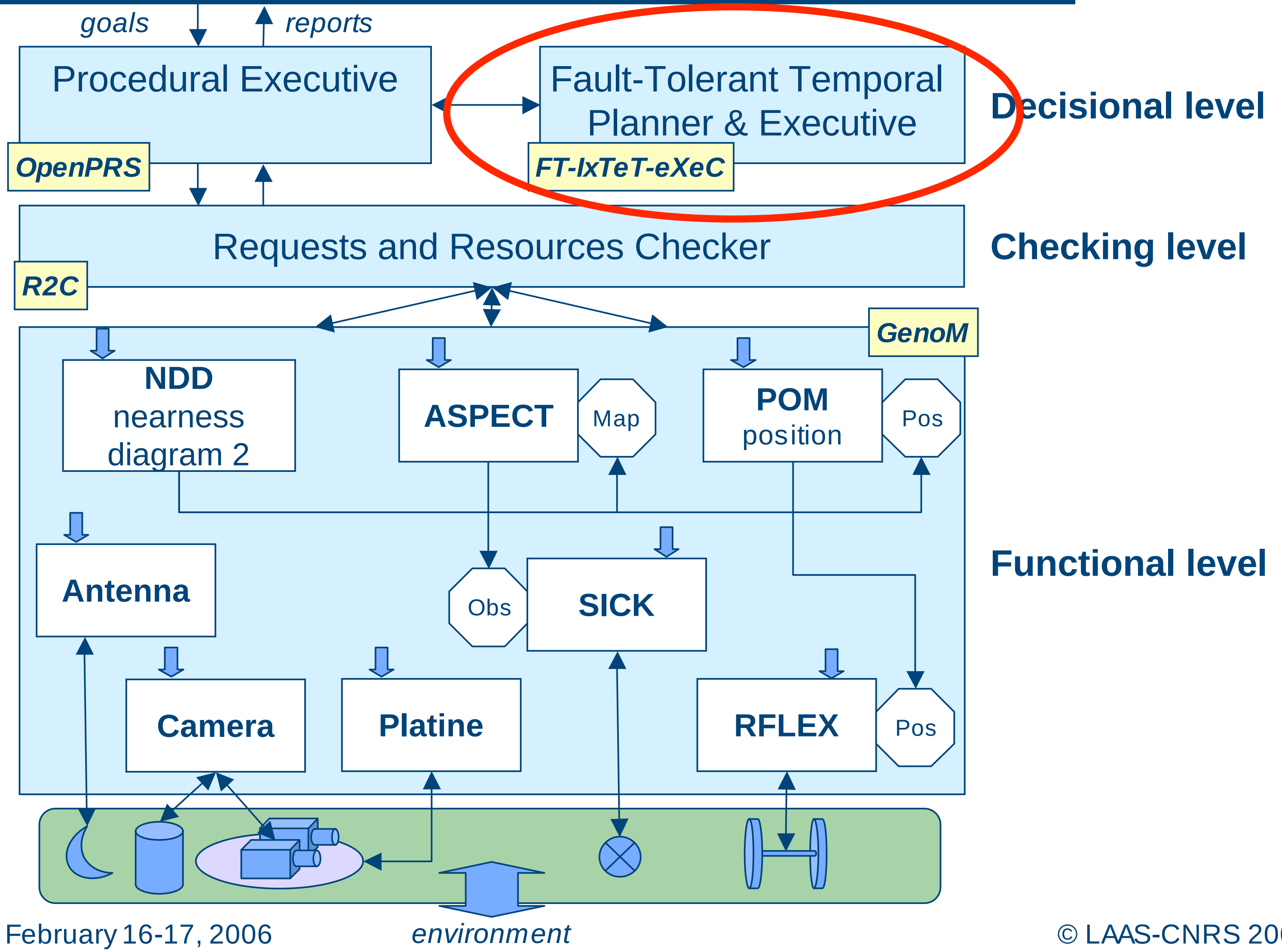
The IxTeT Temporal Planner

Non-nominal situations and new goals

- Adaptation of current plan is required in the following cases:
  - Early or late timing failure of a timepoint occurrence
  - Resource conflict occurs (under/over consumption/production, or detected device failure)
  - Action failure is reported from controlled system
  - New goal is requested
- Adaptation can be of two types:
  - **Plan repair**: planner applies flaw analysis / resolver insertion process to partially invalidated plan, while execution of valid part of plan continues in parallel
  - **Replanning**: start all over from current system state and remaining goals

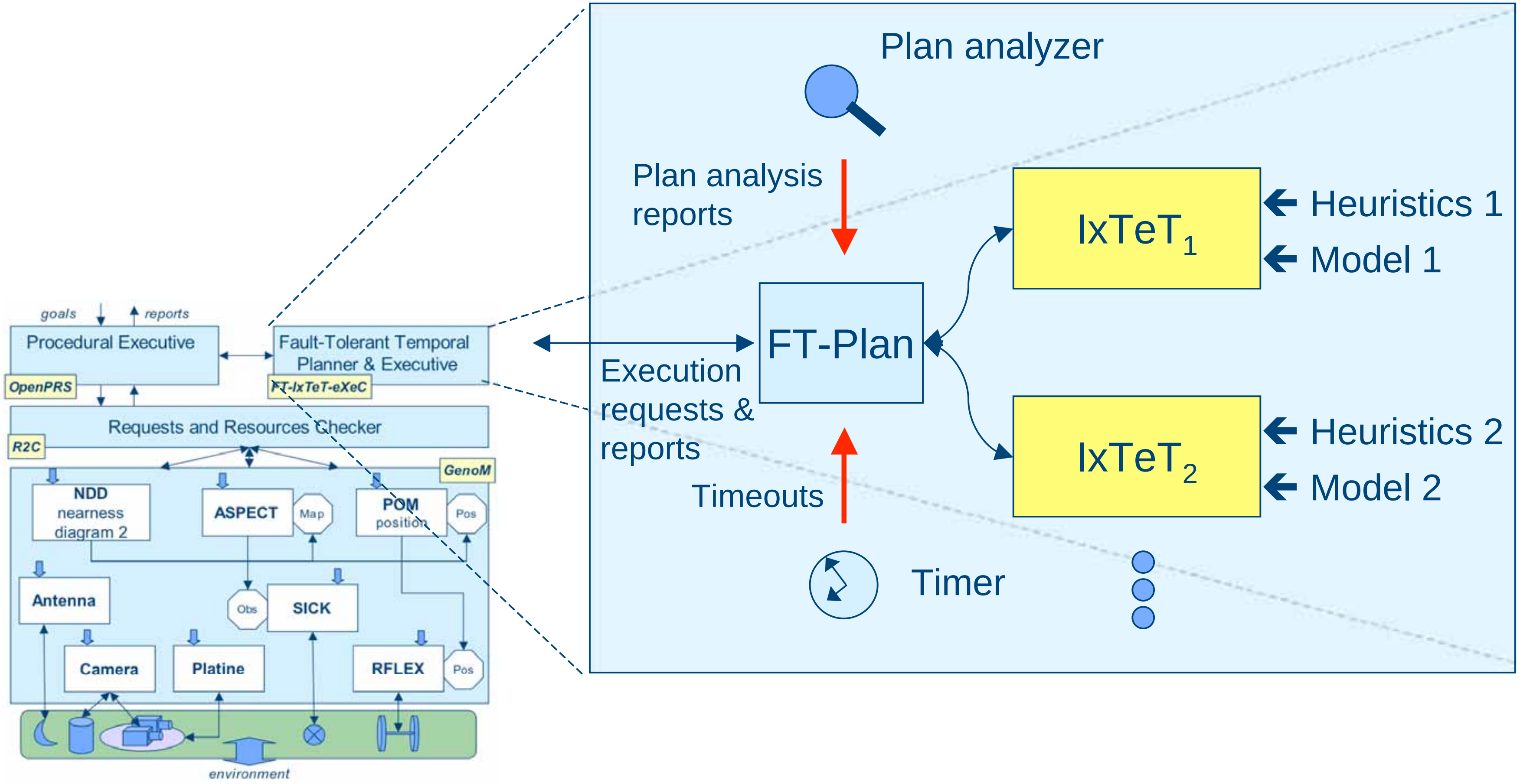
Tolerating Planner Faults

A fault-tolerant IxTeT temporal planner?



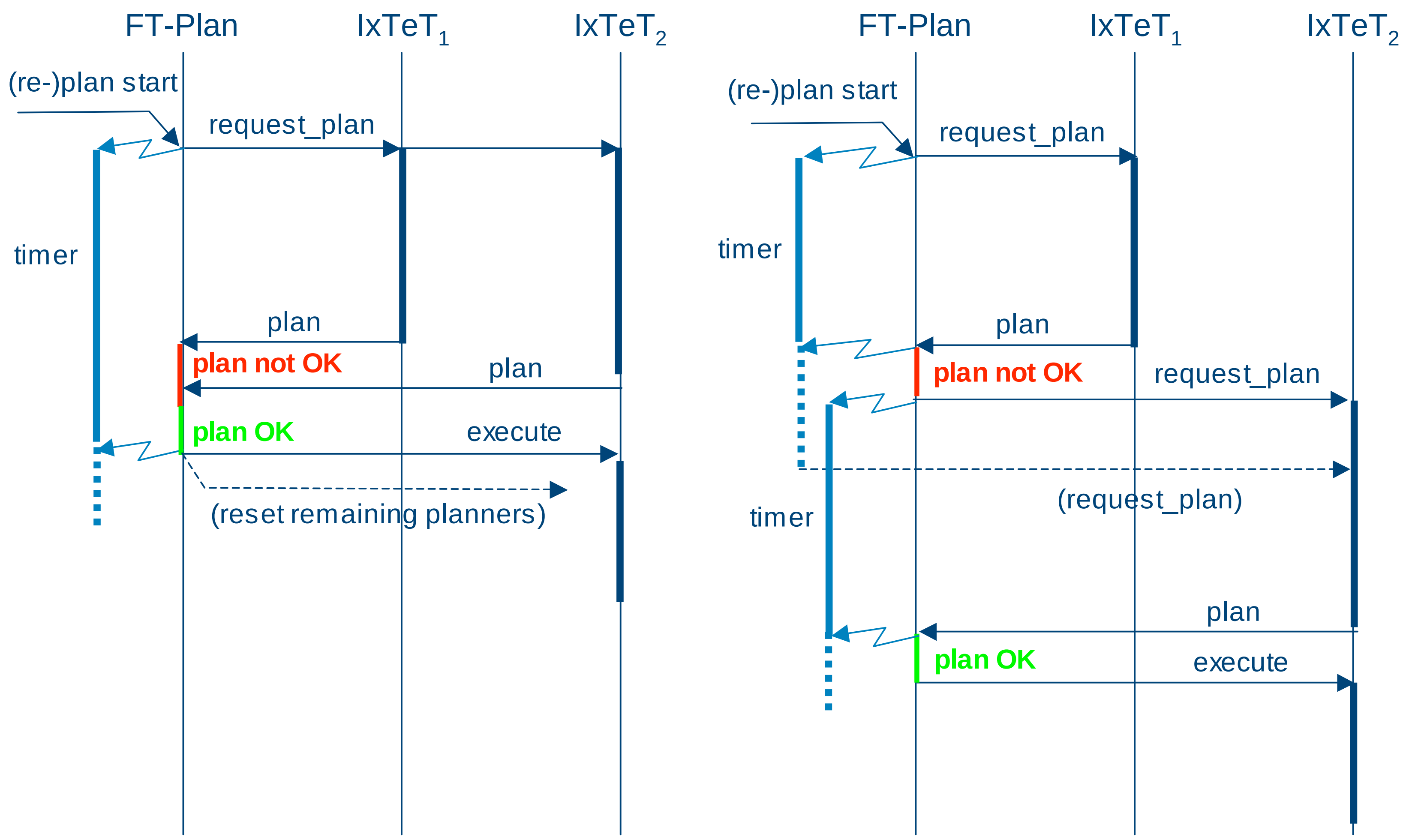
Tolerating Planner Faults

A fault-tolerant IxTeT temporal planner ➔ diversity



Tolerating Planner Faults

Parallel or sequential redundant planners



Tolerating Planner Faults

Parallel redundant planner coordination

```
begin mission
    while(goals≠∅)
        candidates ← planners ;
        send(request_plan) to candidates ; set_timer(max_planning_time) ;
        while(candidates ≠∅)
            wait
             □ plan from any k ∈ candidates
                candidates ← candidates \ k ;
                if analysis(plan)=OK then do
                    send(reset) to candidates ; candidates ← ∅ ;
                    reset_timer(max_planning_time) ;
                    send(execute(plan)) to k ; enddo ;
                    % abnormal termination implies goals≠∅ %
                else do report(k,"invalid plan") ;
                    if candidates = ∅ exception("failure: no valid plan found") ; enddo
             □ timeout(max_planning_time)
                exception("failure: timeout")
        endwhile
    endwhile
end mission
```

Tolerating Planner Faults

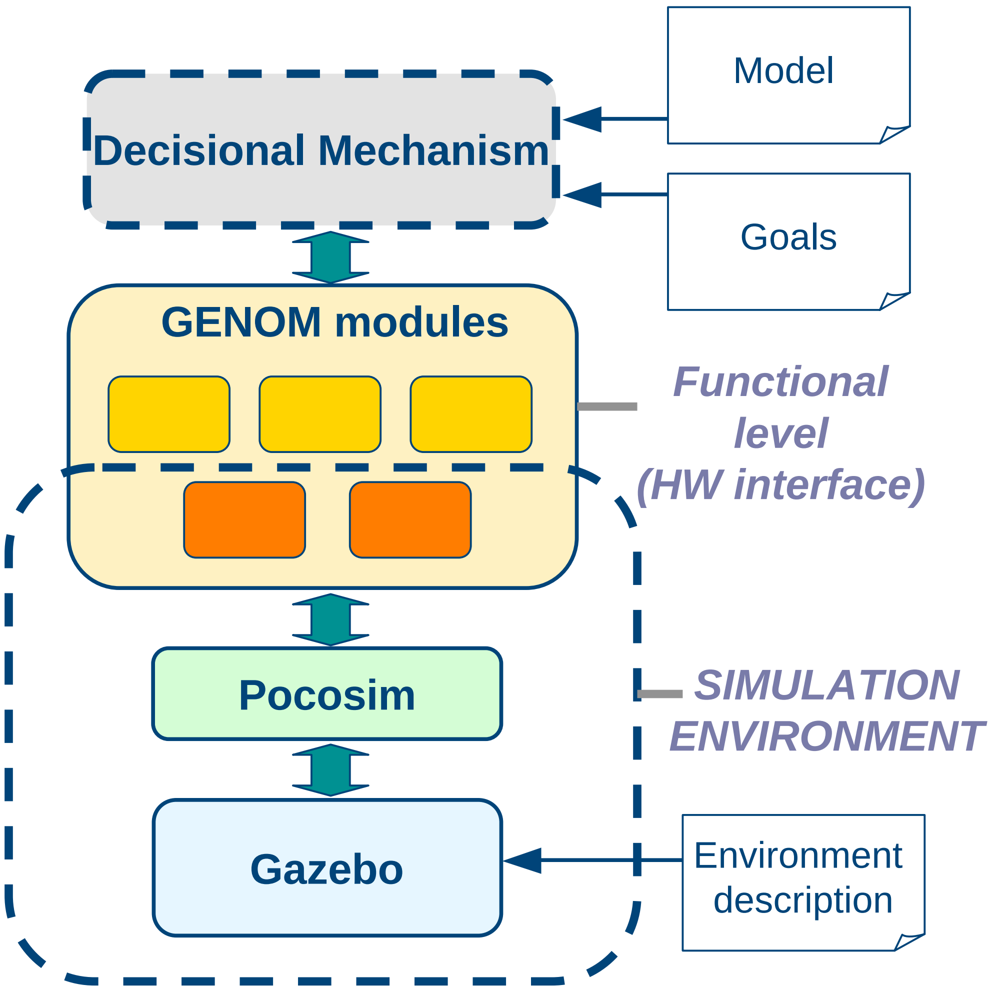
Sequential redundant planner coordination

```
begin mission
    while(goals≠∅)
        candidates ← planners ;
        while(candidates ≠∅)
            choose k ∈ candidates ; % optionally take account of recent failure history %
            candidates ← candidates \ k ;
            send(request_plan) to k ; set_timer(max_planning_time) ;
            wait
            ☐ plan from k
                reset_timer(max_planning_time) ;
                if analysis(plan)=OK then do send(execute(plan)) to k ; enddo
                    % abnormal termination implies goals≠∅ %
                else do report(k,"invalid plan") ;
                    if candidates = ∅ exception("failure: no valid plan found") ;
                    enddo
            ☐ timeout(max_planning_time)
                report(k,"timeout") ;
                if candidates = ∅ exception("failure: no valid plan found") ;
        endwhile
    endwhile
end mission
```

Experimentation Framework

Fault injection and simulation environment

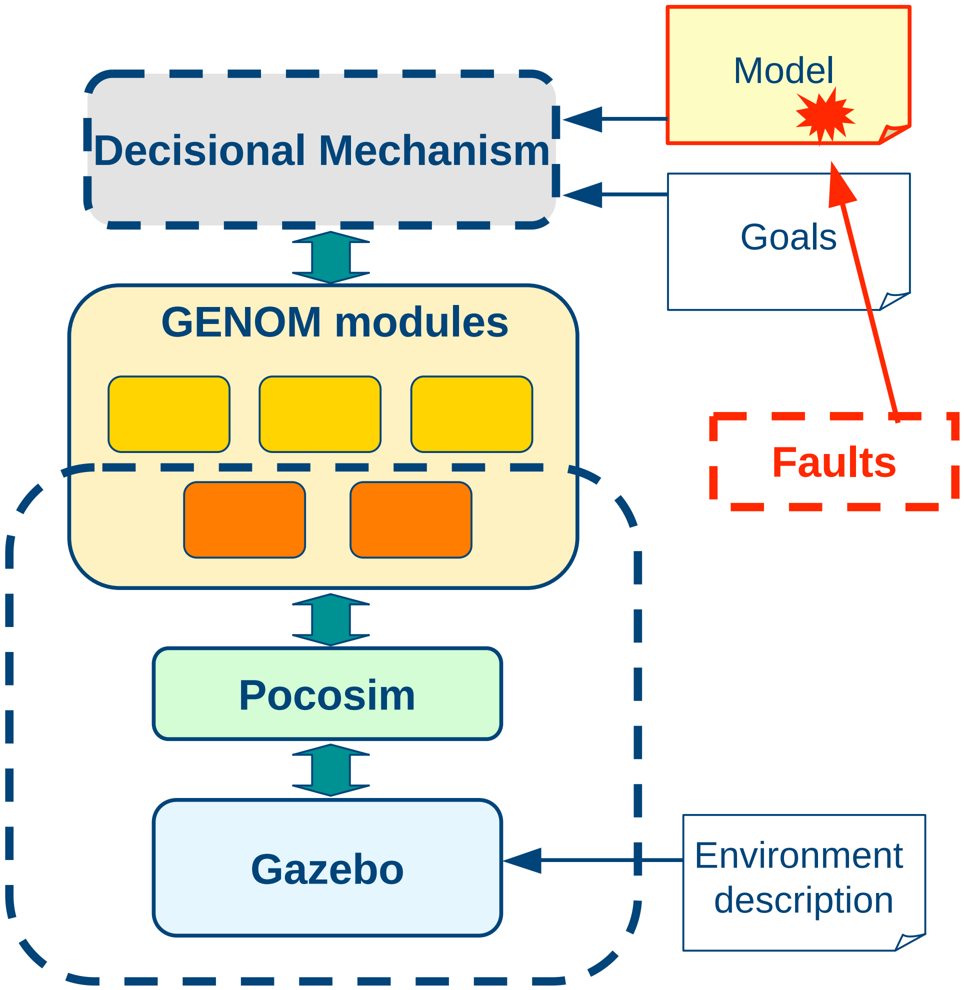
- Why fault injection?
  - To evaluate fault tolerance, we need faults
  - Fault injection simulates efficiently real software faults
- Why simulation?
  - Large number of experiments required for significant evaluation
  - Hazardous behavior of the system during experiments



Experimentation Framework

FARM attributes of fault injection campaign

- Faultload
  - **The faults to be injected**
  - **In our case:** *development faults affecting IxTeT models*
- Activity
- Readouts
- Measures



Experimentation Framework

FARM attributes of fault injection campaign

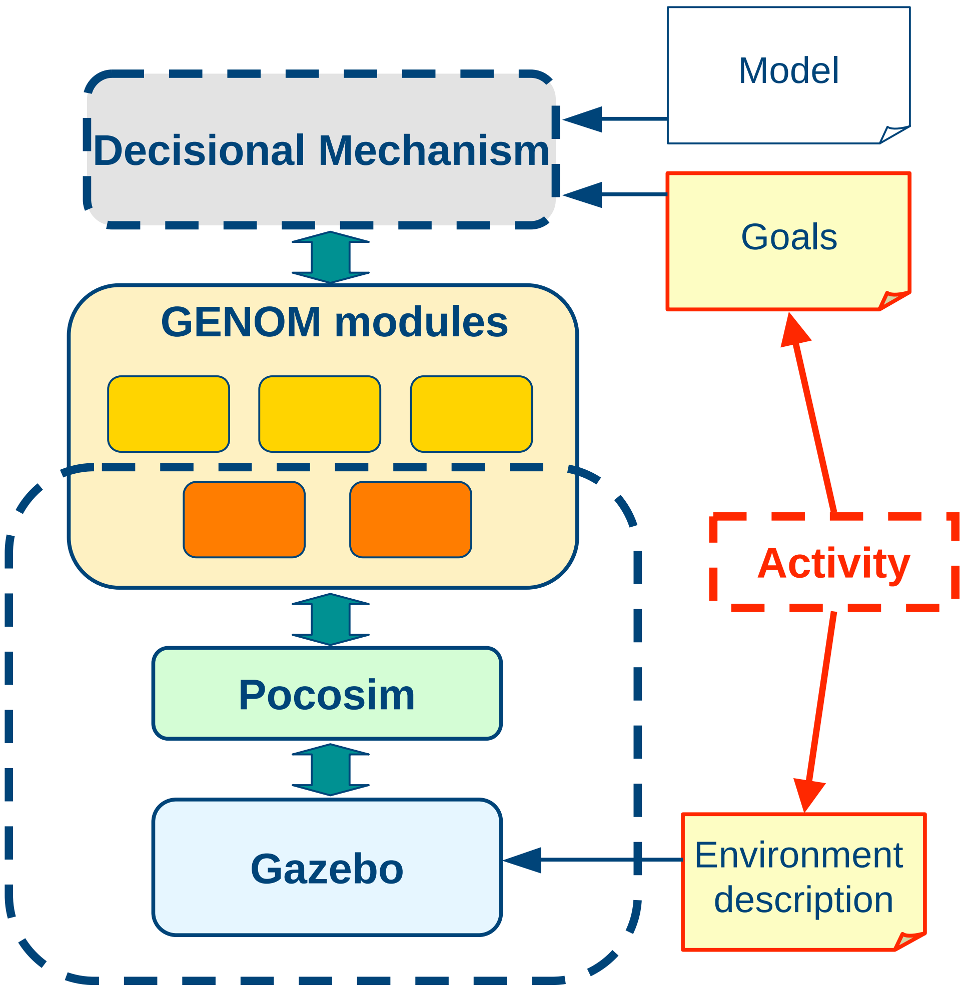
- Syntactic mutations using SESAME tool
  - constant and range substitutions, e.g.
    - {"-oo", "+oo", "0.0", "0.5", "1", "1.4", "1.5", "2", "4", "6", "10", "15", "18", "25", "35", "40", "100", "1000", "0.0-10.0", "0.0-4.0", "9", "100"}
    - {"PICTURE_IDLE", "NONE", "DONE", "COMMUNICATION_IDLE"}
  - variable substitutions, e.g.
    - {"?initpos", "?finpos}"
    - {"?x," "?y", "?obj"}
  - operator substitutions, e.g.
    - {"explained", "contingent")
    - ("nonPreemptive", "latePreemptive", "earlyPreemptive"}
- Addition / removal of a model statement, i.e., a constraint

➔ Database of compilable mutations

Experimentation Framework

FARM attributes of fault injection campaign

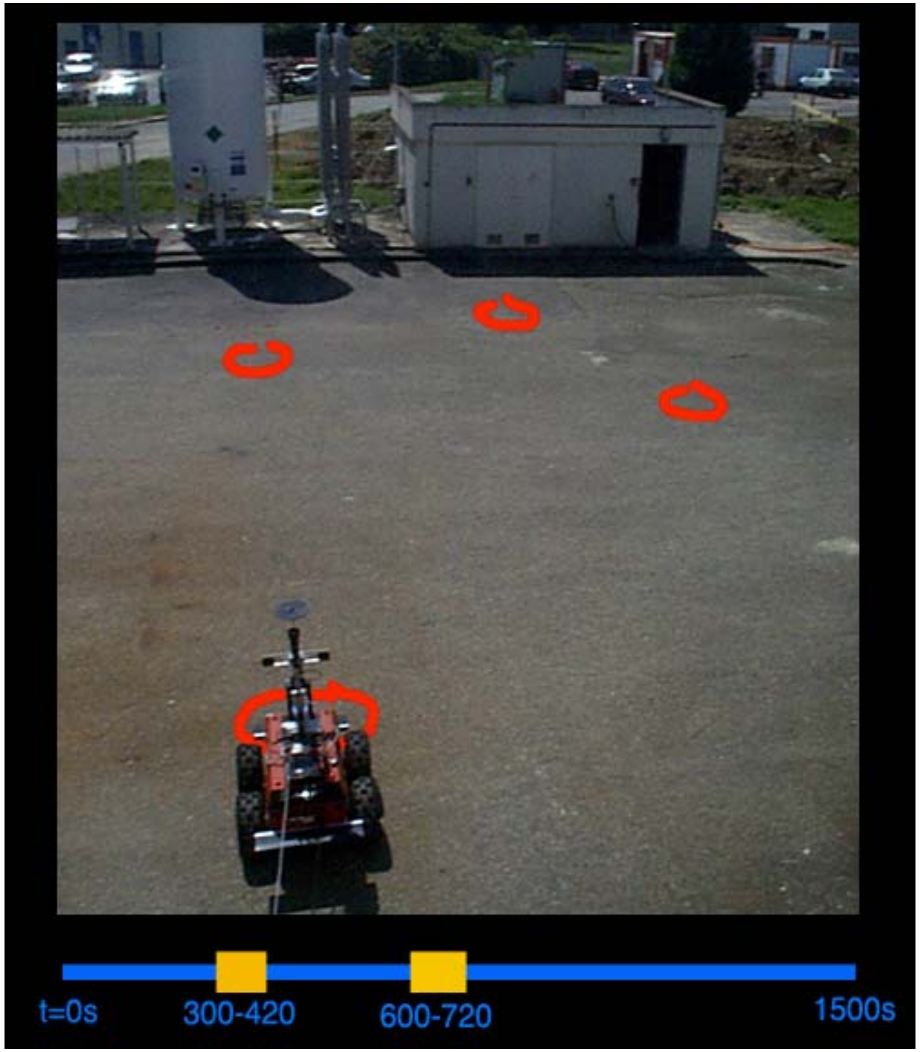
- Faultload
- Activity
  - The workload executed by the robot during an experiment
    - Goals
    - Environment
- Readouts
- Measures



Experimentation Framework

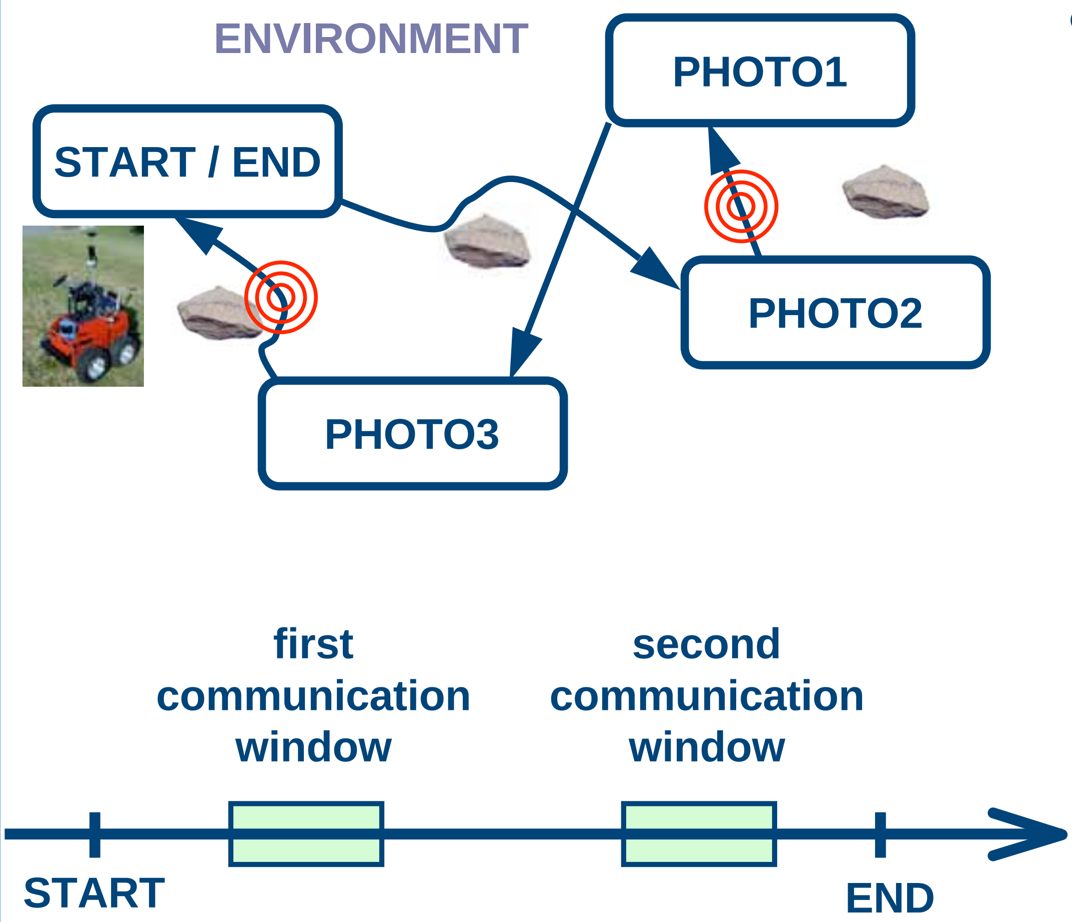
FARM attributes of fault injection campaign

- Faultload
- Activity
  - The workload executed by the robot during an experiment
    - Goals
    - Environment
- Readouts
- Measures



Experimentation Framework

FARM attributes of fault injection campaign

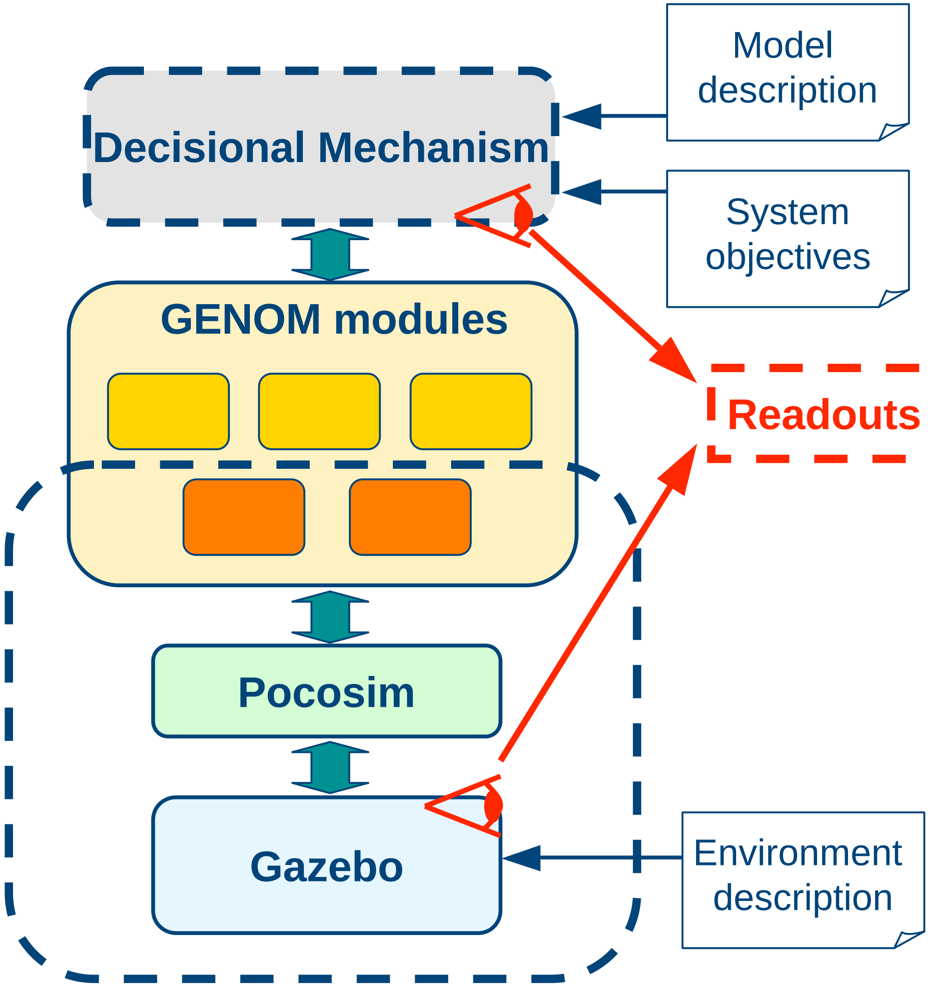


- Activity definition variables
  - Physical dimension
    - Number and location of photos to be taken
    - Topology of the environment
  - Temporal dimension
    - Number and occurrence times of communication windows

Experimentation Framework

FARM attributes of fault injection campaign

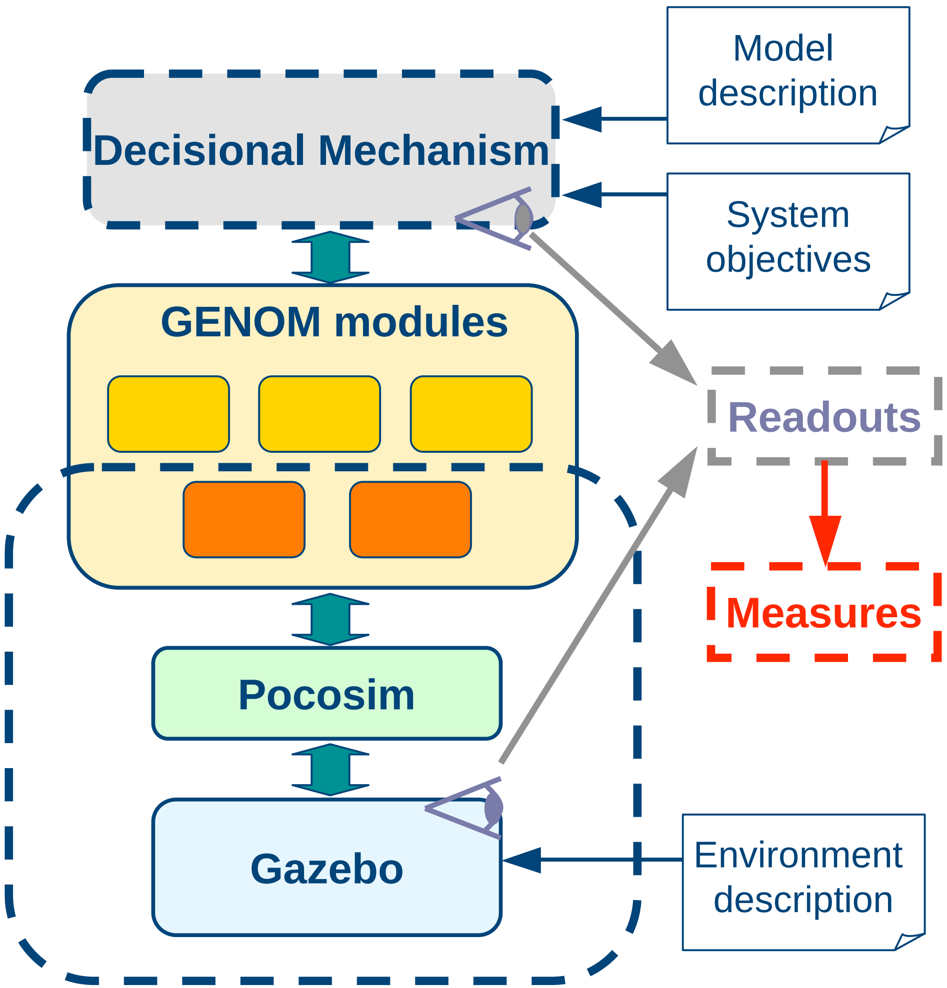
- Faultload
- Activity
- Readouts
  - Observations during one experiment
  - Fault outcome
    - Activated?
    - Error detected?
  - System behavior
    - Goals achieved?
    - Performance?
- Measures



Experimentation Framework

FARM attributes of fault injection campaign

- Faultload
- Activity
- Readouts
- Measures
  - Statistics on set of readouts
  - Dependability-specific
    - Coverage
    - Detection latency
  - Performance
    - Goals achieved
    - Distance and time required



Preliminary results

Baseline configuration: no redundancy

ID	mutation		readouts	possible explanation
	before	after		
3-041	?x in]**-oo**,+oo[;	?x in]**9**,+oo[;	ixtet crash	conflicting constraints for ?x
3-416	?dist =. ?di *. **?du**;	?dist =. ?di *. **?y2**;	ixtet crash	conflicting constraints for ?dist
3-472	**?duration** in]1.5, +oo[;	**?y2** in]1.5, +oo[;	ixtet crash	conflicting constraints for ?y2
1-296	**?Xc** =. _xi -. _xf;\	**_d** =. _xi -. _xf;\	ixtet crash	conflicting constraints for _d
3-525	event(COMMUNICATION(?w):(COMMUNICATION_IDLE,**DONE**),t_end);	event(COMMUNICATION(?w):(COMMUNICATION_IDLE,**COMMUNICATION_IDLE**),t_end);	ixtet hang timeout	ixtet compilation bug
2-162	explained event(AT_ROBOT_X() : (1.0, **0.0**), t_start);	explained event(AT_ROBOT_X() : (1.0, **10.0**), t_start);	ixtet hang timeout	ixtet execution bug
3-110	?mindist =. **0.5**;	?mindist =. **6**;	ixtet search timeout	model overconstrained, thus no solutions found
3-128	?minduration =. **1.4**;	?minduration =. **15**;	ixtet search timeout	model overconstrained, thus no solutions found
3-540	hold(PTU_DRIVER_INITIALIZED():**TRUE**,(t_start,t_end));	hold(PTU_DRIVER_INITIALIZED():**PTU_DRIVER_INITIALIZED_IDLE**,(t_start,t_end));	ixtet search timeout	a necessary condition becomes impossible to achieve as no other action can act as a resolver

Preliminary results

Baseline configuration: no redundancy

ID	mutation		readouts	possible explanation
	before	after		
3-139	?duration in]**1.5**, +oo[;	?duration in]**-oo**, +oo[;	(3, 2, 1, 14.0, 254)	one constraint was relaxed, but no incidence on plan
1-344	_u in [**0.7**,1]	_u in [**0.17**,1]	(3, 2, 1, 14.0, 254)	one constraint was relaxed, but no incidence on plan
3-418	?dist =. ?di *. **?du**;	?dist =. ?di *. **?di**;	(3, 2, 1, 19.0, 320)	wrong distance leads to bad performance, but all goals fulfilled
3-277	**?dist =. ?di *.** ?du;	**?dist =. ?di +.** ?du;	(2, 1, 1, 19.2, 353)	wrong distance leads to missed goals
2-050	explained event (COMMUNICATION(W1) : (COMMUNICATION_IDLE, **NONE**), t_start);	explained event (COMMUNICATION(W1) : (COMMUNICATION_IDLE, **DONE**), t_start);	(3, 1, 1, 14.1, 255)	one communication explained as DONE in the initial situation, thus goal not fulfilled
3-559	}**latePreemptive**	}**nonPreemptive**	(2, 0, 1, 19.1, 155)	action MOVE can no longer be interrupted, thus execution is quicker but some goals are missed

(image, communicate, location, distance, duration)

goals performance

Conclusion

Very preliminary results

- First mutation experiments on declarative models
- Tuning such models is a “black art”
  - easy to get wrong in subtle ways
  - tolerance of subtle faults might indeed be useful
- 15 mutations to date
  - 13% of mutations had no effect (all goals met)
  - 27% of mutations resulted in sub-optimal mission
    - goals missed
    - degraded performance
  - 60% of mutations resulted in crashes/hangs/timeouts
    - easy to detect

Conclusion

Current and future work

- Fault injection process
  - currently being automated
  - random goal selection
  - “difficult” environments
  - multiple parallel simulations (~10 minutes / experiment)
- Implementation of FT-Plan with dual IxTeTs
  - diverse search heuristics
    - parameters of depth-first search cost functions
    - …
  - diverse models
    - cellular model
    - state mapping between models based on different abstractions?
    - …
  - definition of a plan analyzer (on-line test oracle)