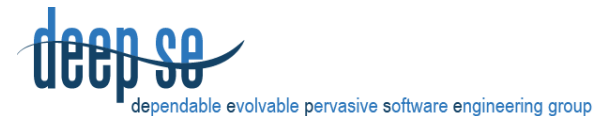




Dynamic Updates and Self-Adaptation Can we fill the gap?

Valerio Panzica La Manna – Politecnico di Milano, Italy
panzica@elet.polimi.it





Dynamic Updates...



Introduction

- Modern software systems are subject to continuous and **unanticipated changes**:
 - In the surrounding environment.
 - In the requirements.



Introduction

- Modern software systems are subject to continuous and **unanticipated changes**:
 - In the surrounding environment.
 - In the requirements.
- To incorporate these changes, systems are typically **updated offline**:
 - Shutdown
 - Update
 - Restart



Introduction

- Modern software systems are subject to continuous and **unanticipated changes**:
 - In the surrounding environment.
 - In the requirements.
- To incorporate these changes, systems are typically **updated offline**:
 - Shutdown
 - Update
 - Restart
- Many of these systems **must operate continuously**



Introduction

- Modern software systems are subject to continuous and **unanticipated changes**:
 - In the surrounding environment.
 - In the requirements.
- To incorporate these changes, systems are typically **updated offline**:
 - Shutdown
 - Update
 - Restart
- Many of these systems **must operate continuously**





Introduction

- Modern software systems are subject to continuous and **unanticipated changes**:
 - In the surrounding environment.
 - In the requirements.
- To incorporate these changes, systems are typically **updated offline**:
 - Shutdown
 - Update
 - Restart
- Many of these systems **must operate continuously**





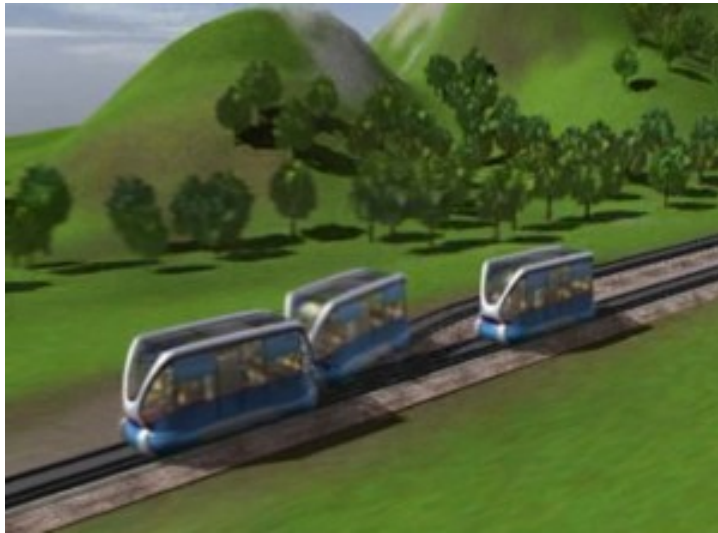
Dynamic Software Update

Engineering software systems able to evolve at run-time

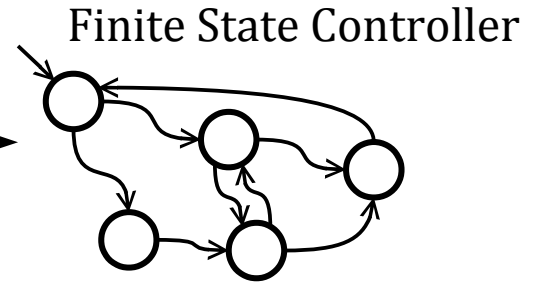
- Dynamic updates must be **safe**
 - The update process must not lead to erroneous behavior
 - Crucial for safety-critical applications.
- Systems must be updated **as soon as possible**
 - To adapt to unpredicted changes in the environment
 - To incorporate new critical requirements



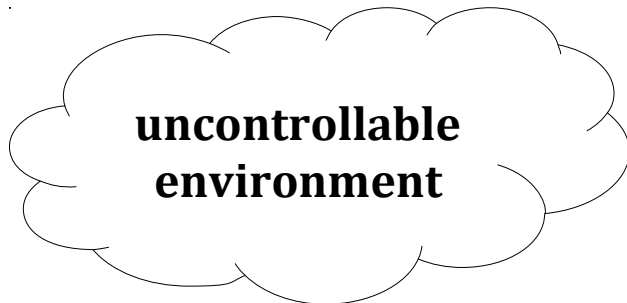
Open Reactive Systems



controlled by



operate



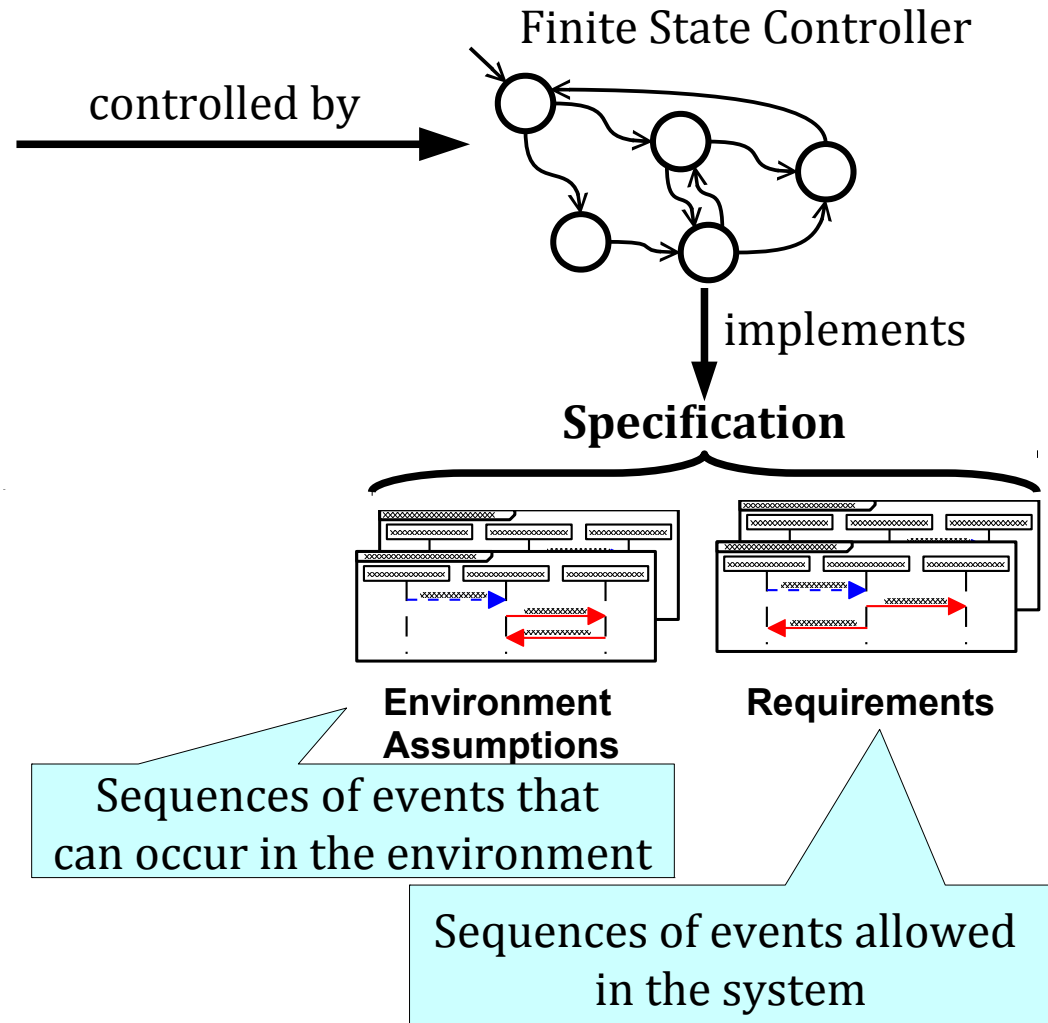


Open Reactive Systems

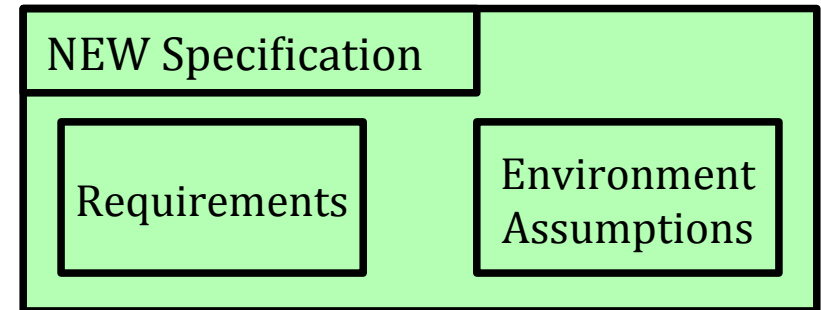
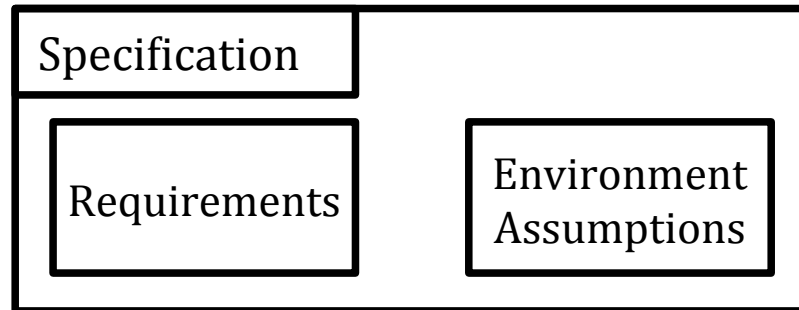


↑ operate

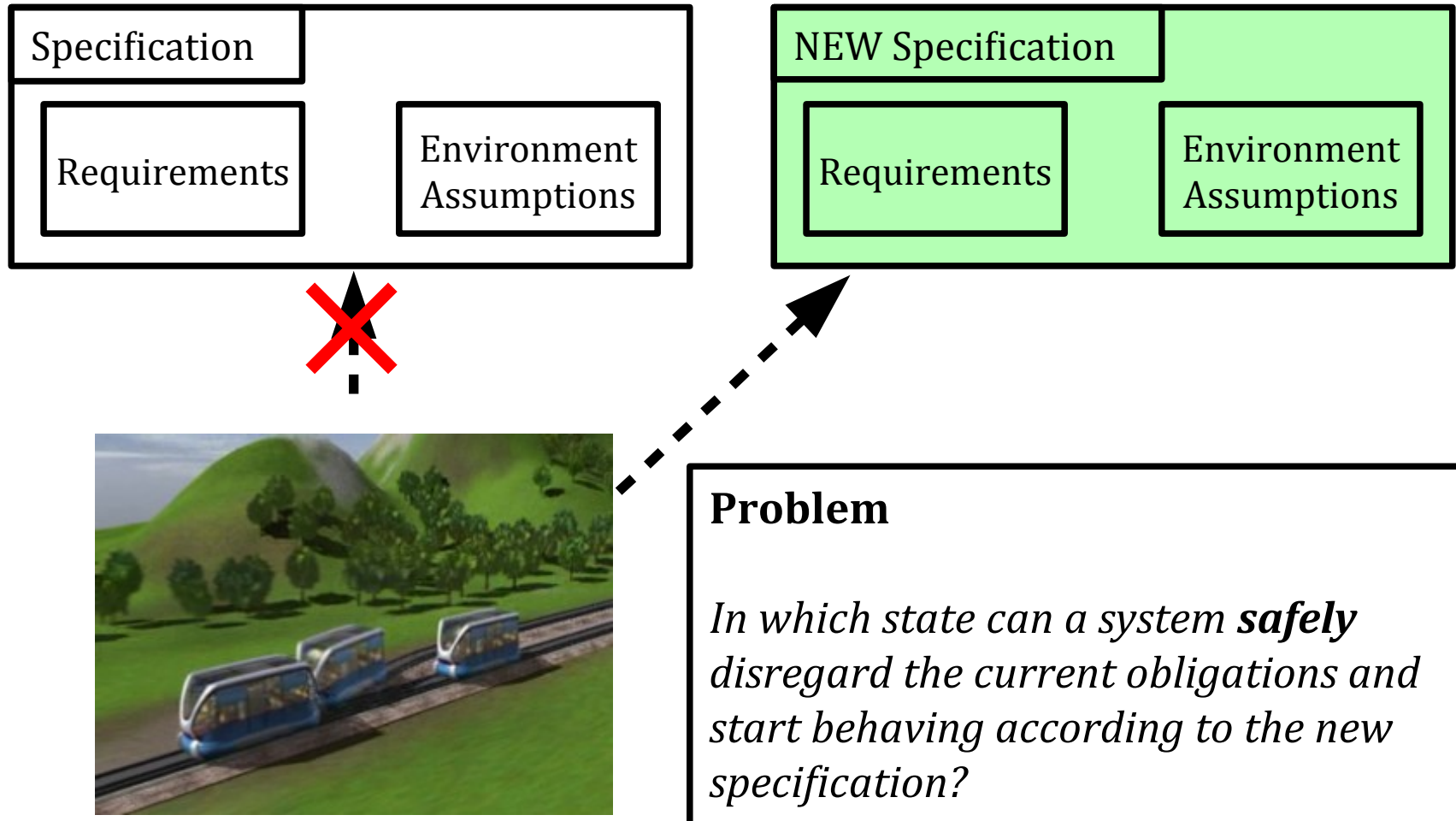
**uncontrollable
environment**



A specification oriented perspective



A specification oriented perspective





Our contribution

- Formalizing criteria for safe dynamic updates.
 - Definition of updatable states [1].
 - Additional criteria for more timely updates [2].
- Approach to automatically construct dynamically updating systems from changes in MSD specification [1, 2].
- Tool realization as part of ScenarioTools [2].

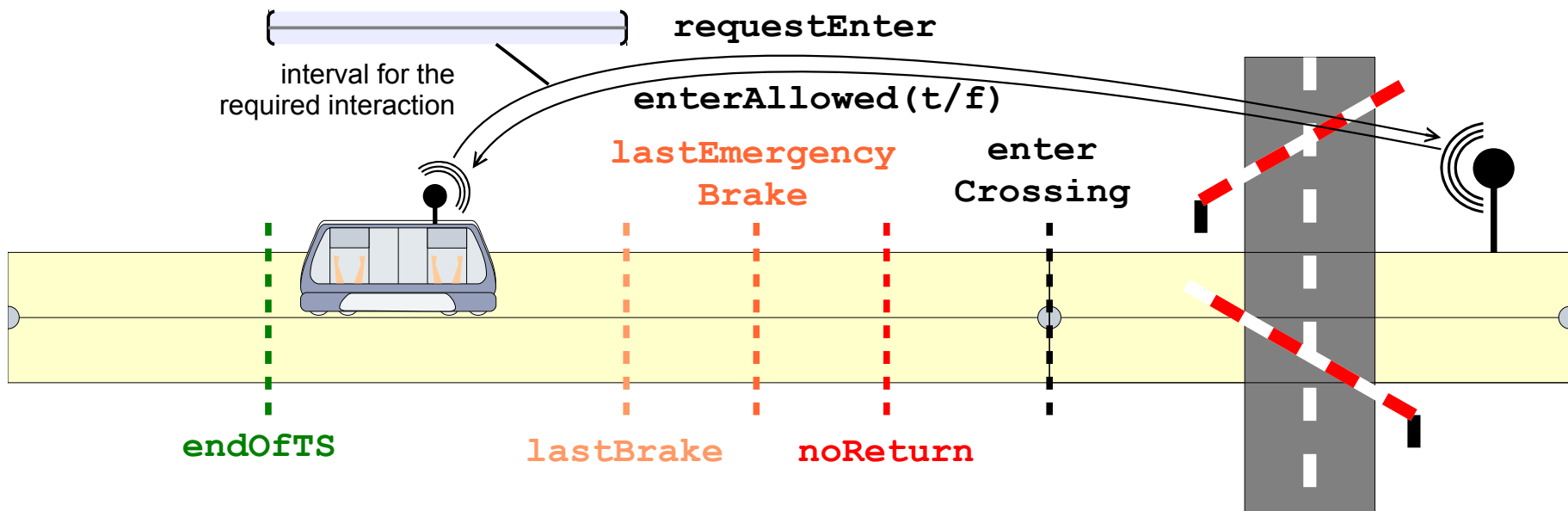
[1] C.Ghezzi et al. “Synthesizing dynamically updating controllers from changes in scenario based specification”, SEAMS 2012.

[2] V. Panzica La Manna et al. “Formalizing correctness criteria of dynamic updates derived from specification changes”, SEAMS 2013.

Example: Current Specification



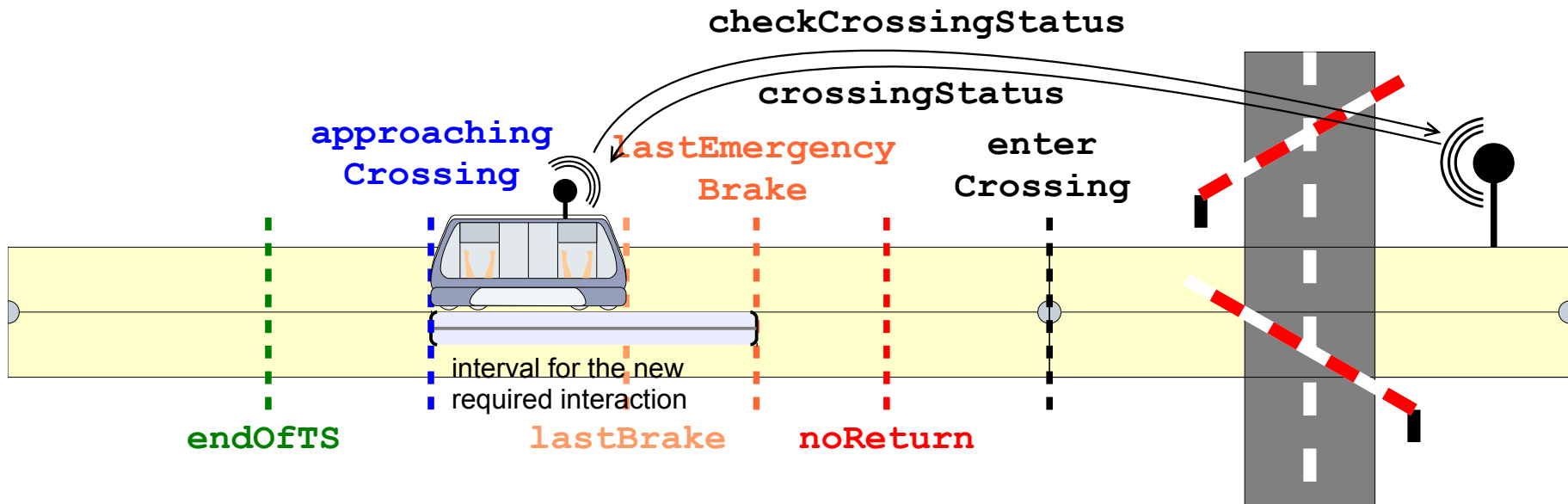
- **Environment Assumptions:** The points on the track section occur in the shown order.
- **Requirements:** After `endOfTS` and before `lastBrake`, the RC requests the crossing control permission to enter the crossing (...)



Example: Specification Change

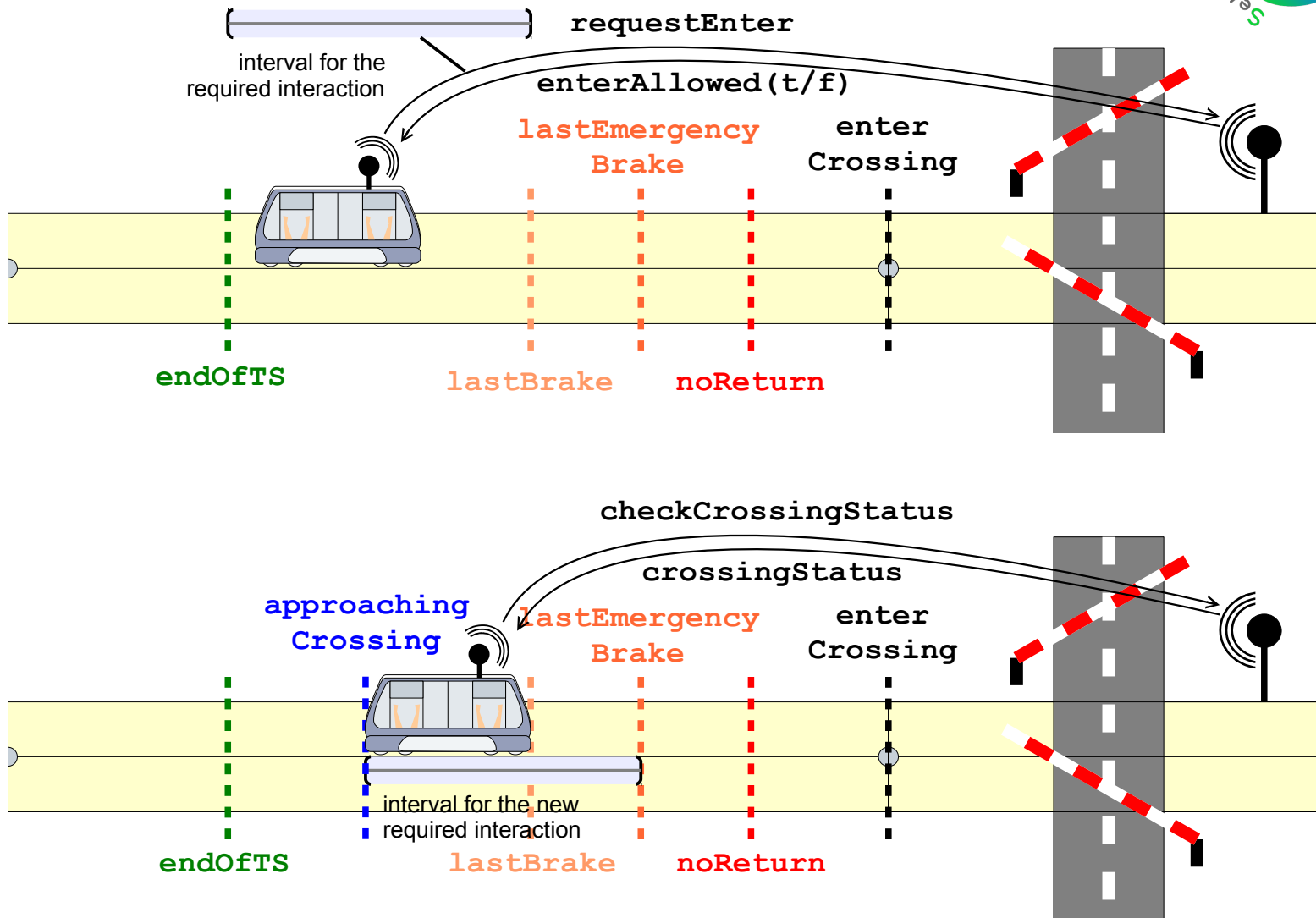


- **Changed Assumptions:** The event `approachingCrossing` occurs in the sequence of environment events as shown
- **New Added Requirement:** After `approachingCrossing` and before `lastEmergencyBrake`, the RC must check the status of the crossing.

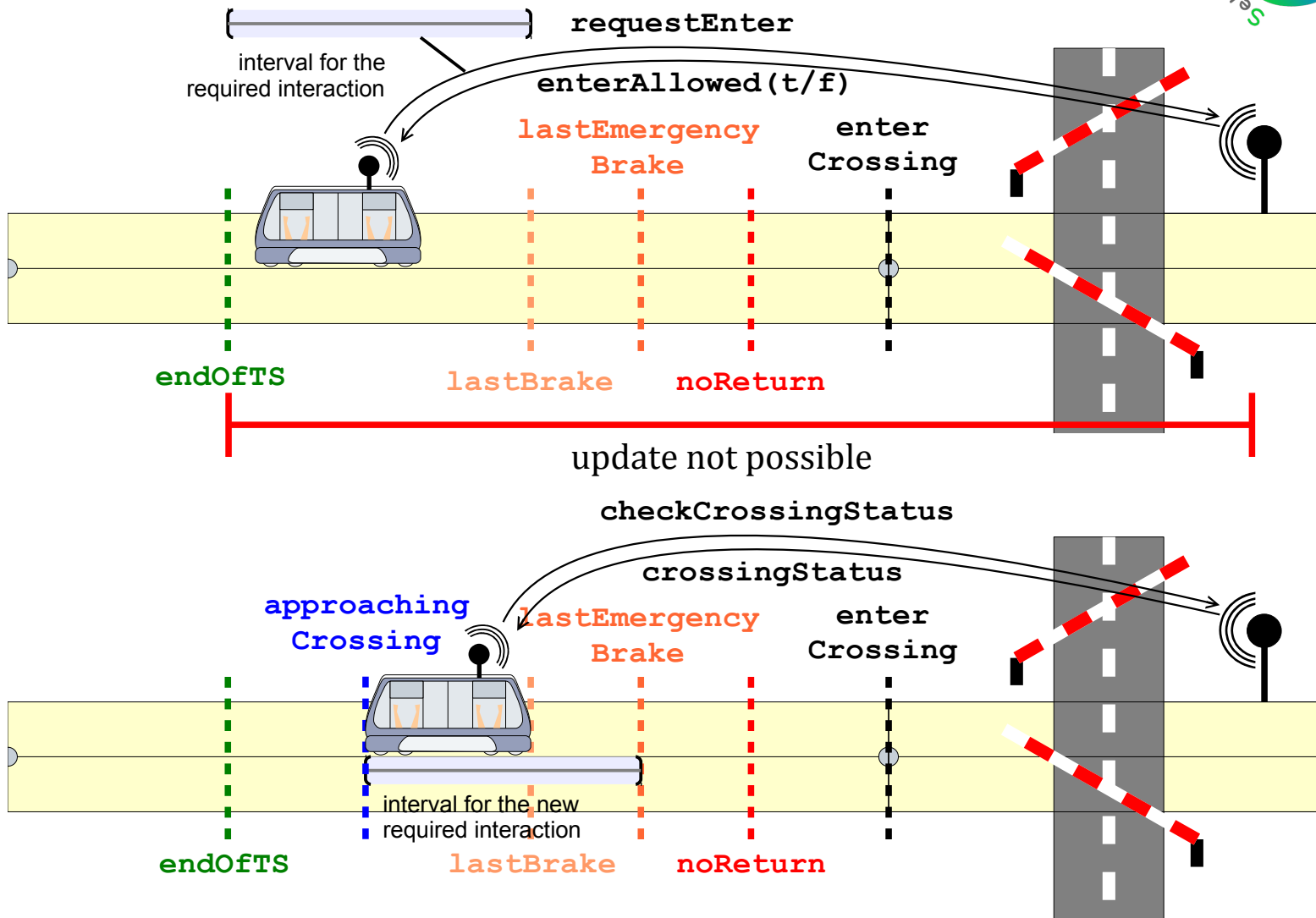




Example: Offline Update

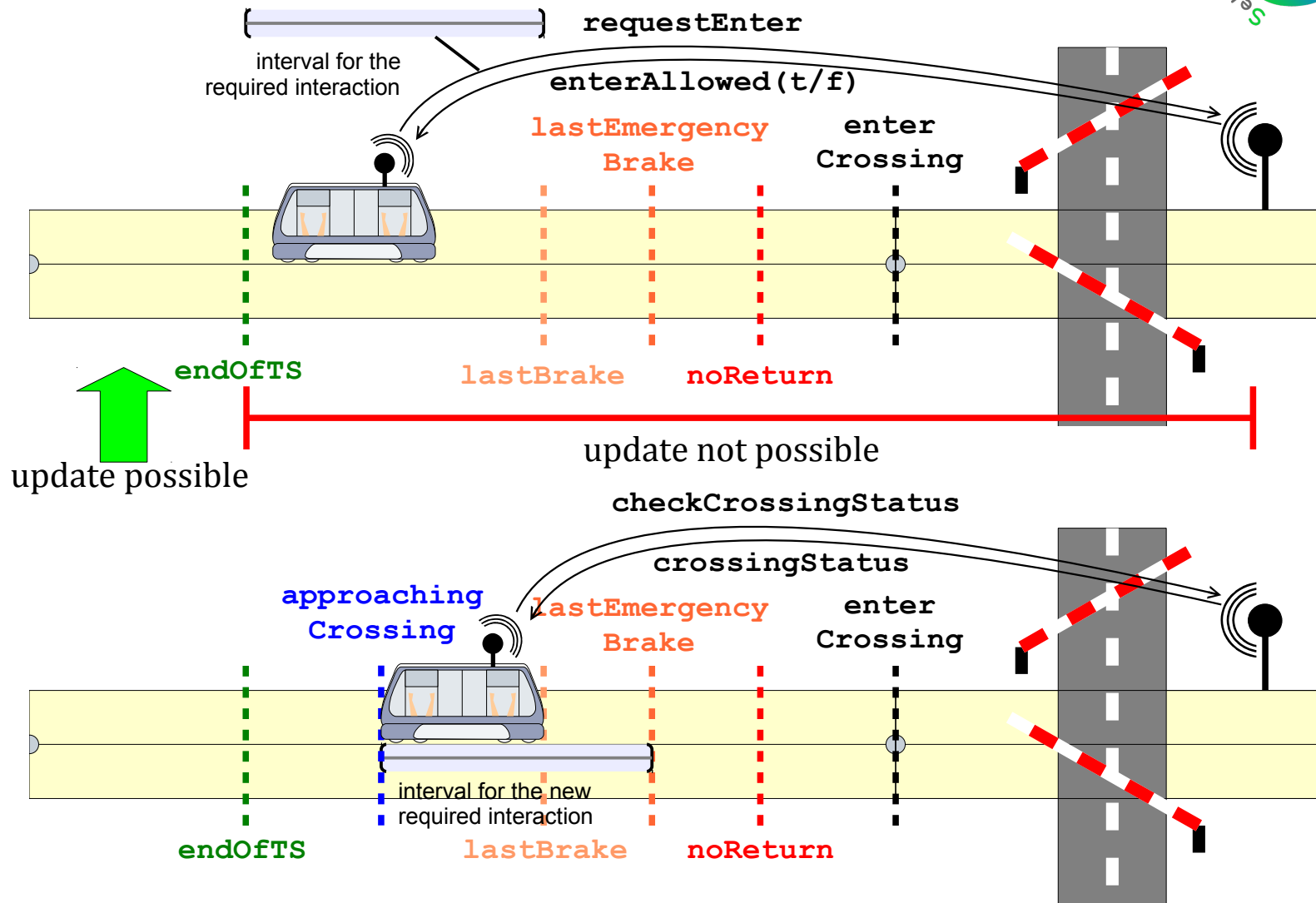


Example: Offline Update



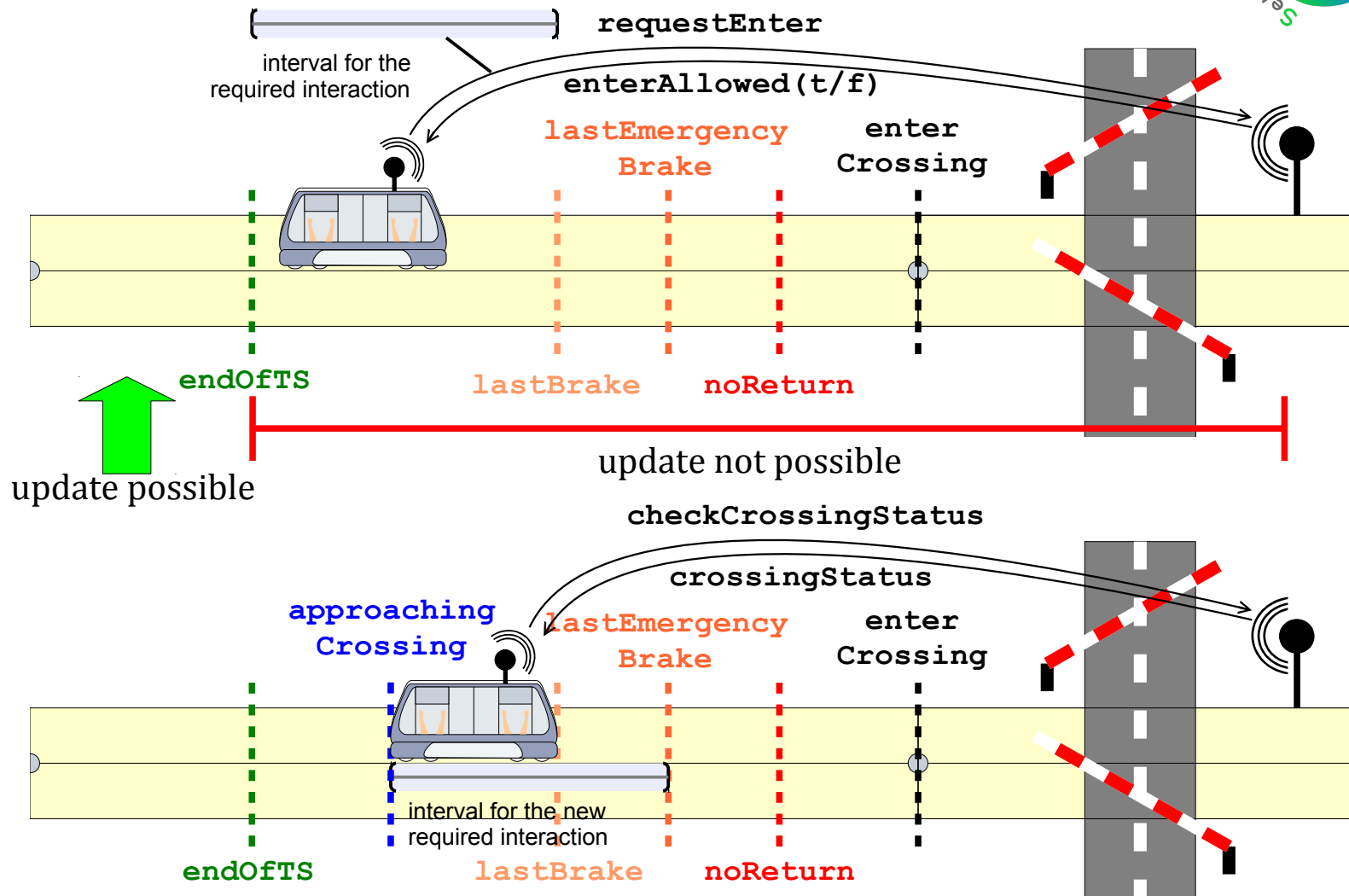


Example: Offline Update





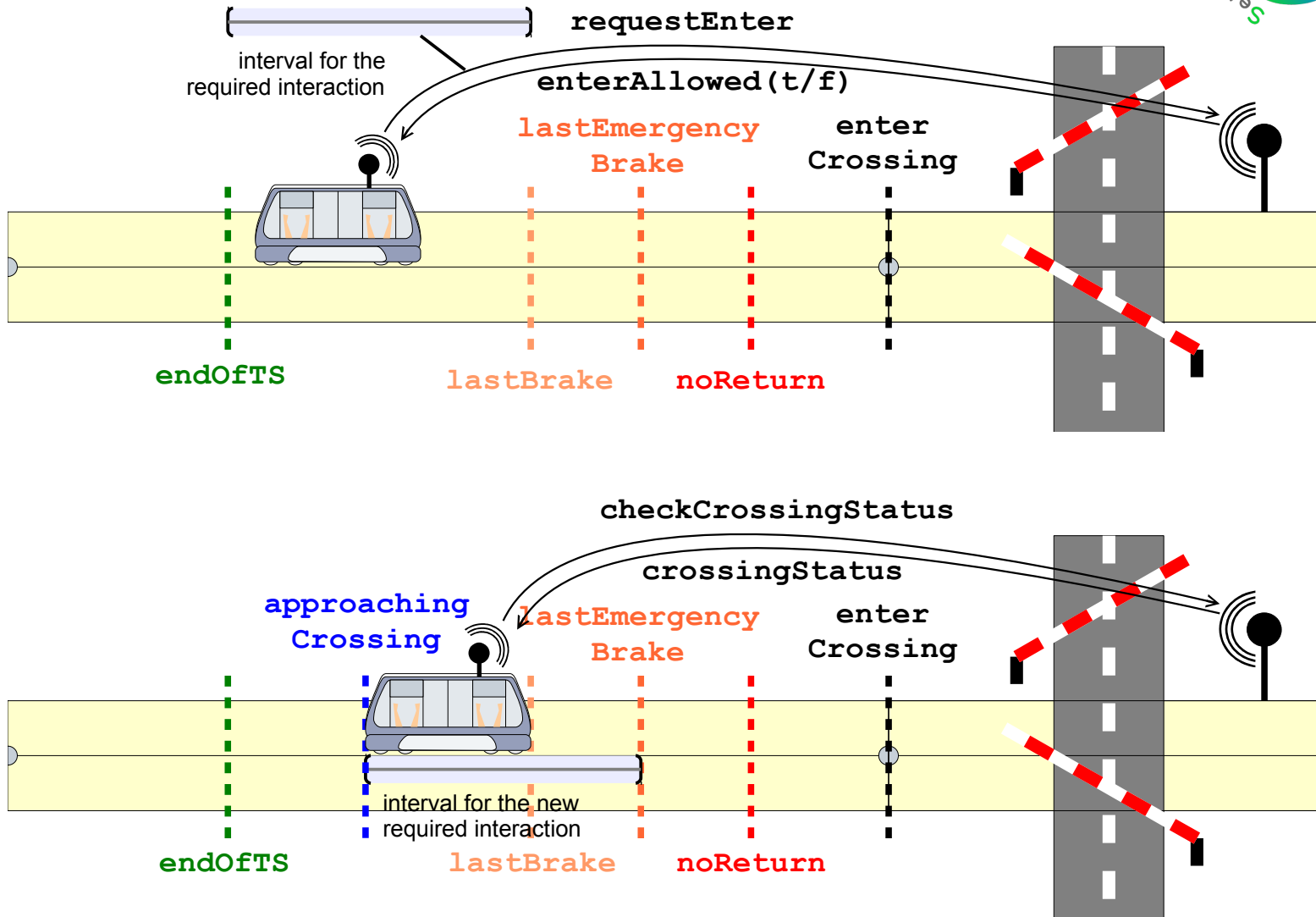
Example: Offline Update



How to dynamically update the RailCab to the new behavior at run-time?

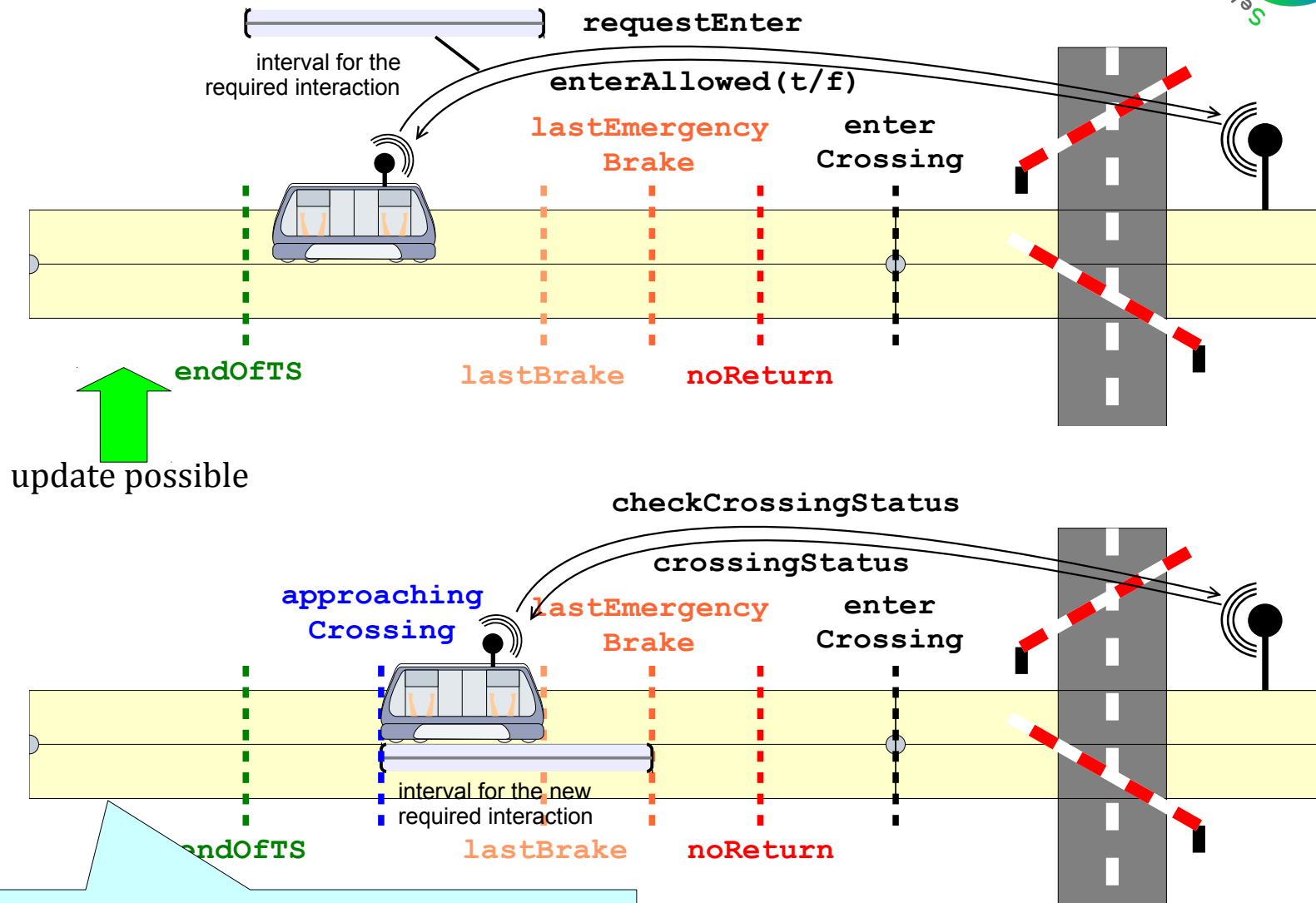


Fundamental criterion





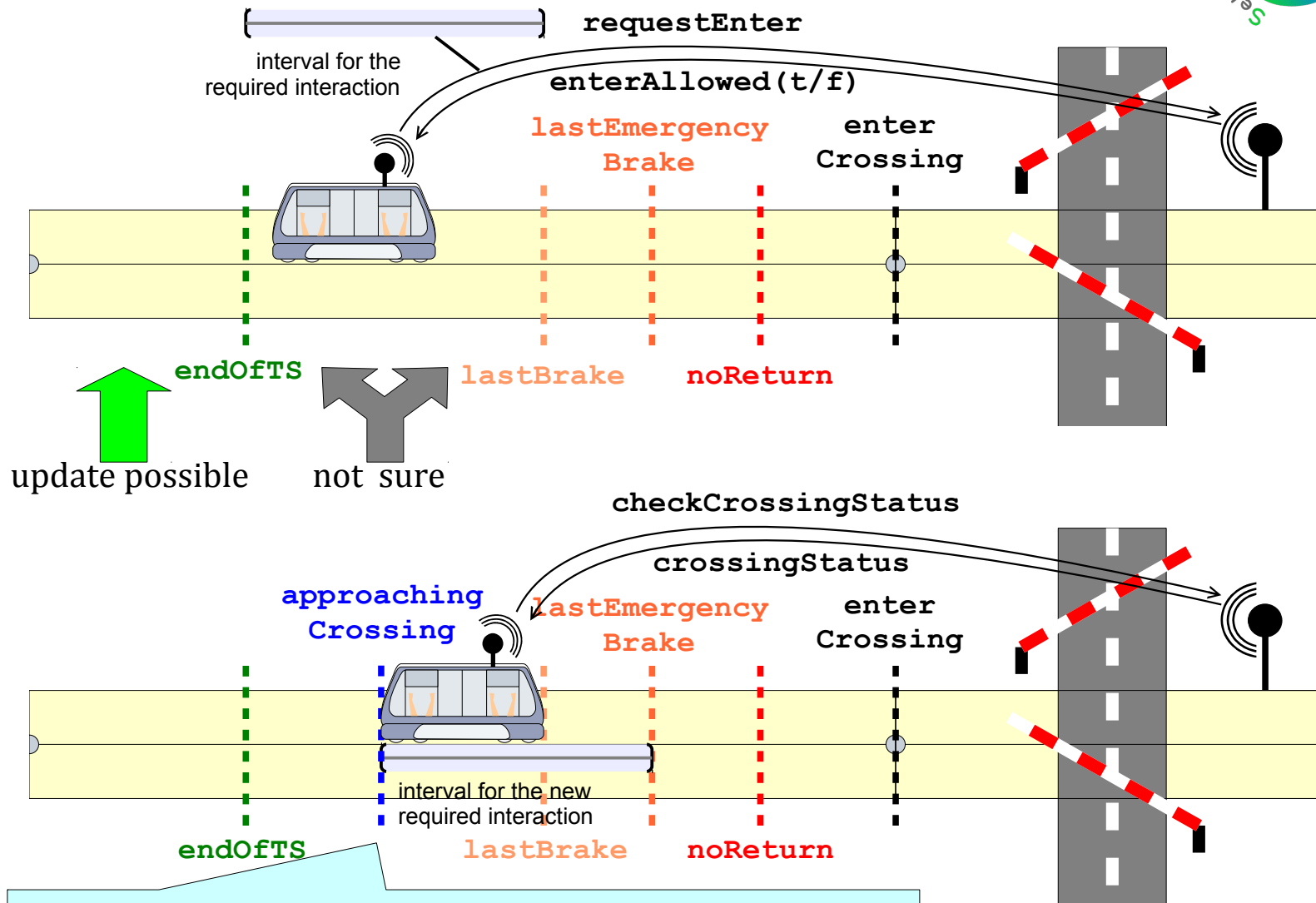
Fundamental criterion



initial state: it can always be updated (same behavior of off-line updates).



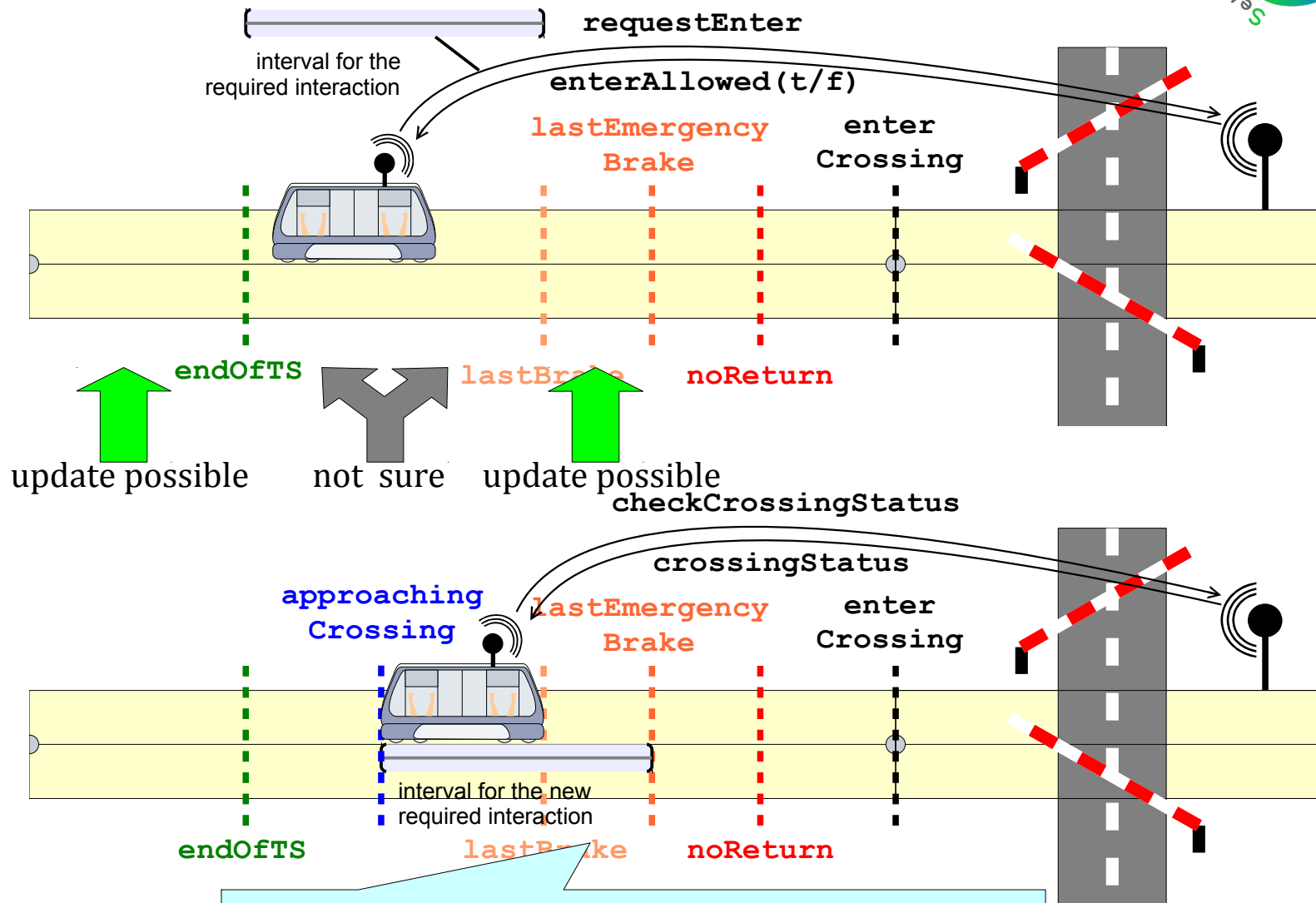
Fundamental criterion



the running system does not remember this event since it is not present in the old assumptions



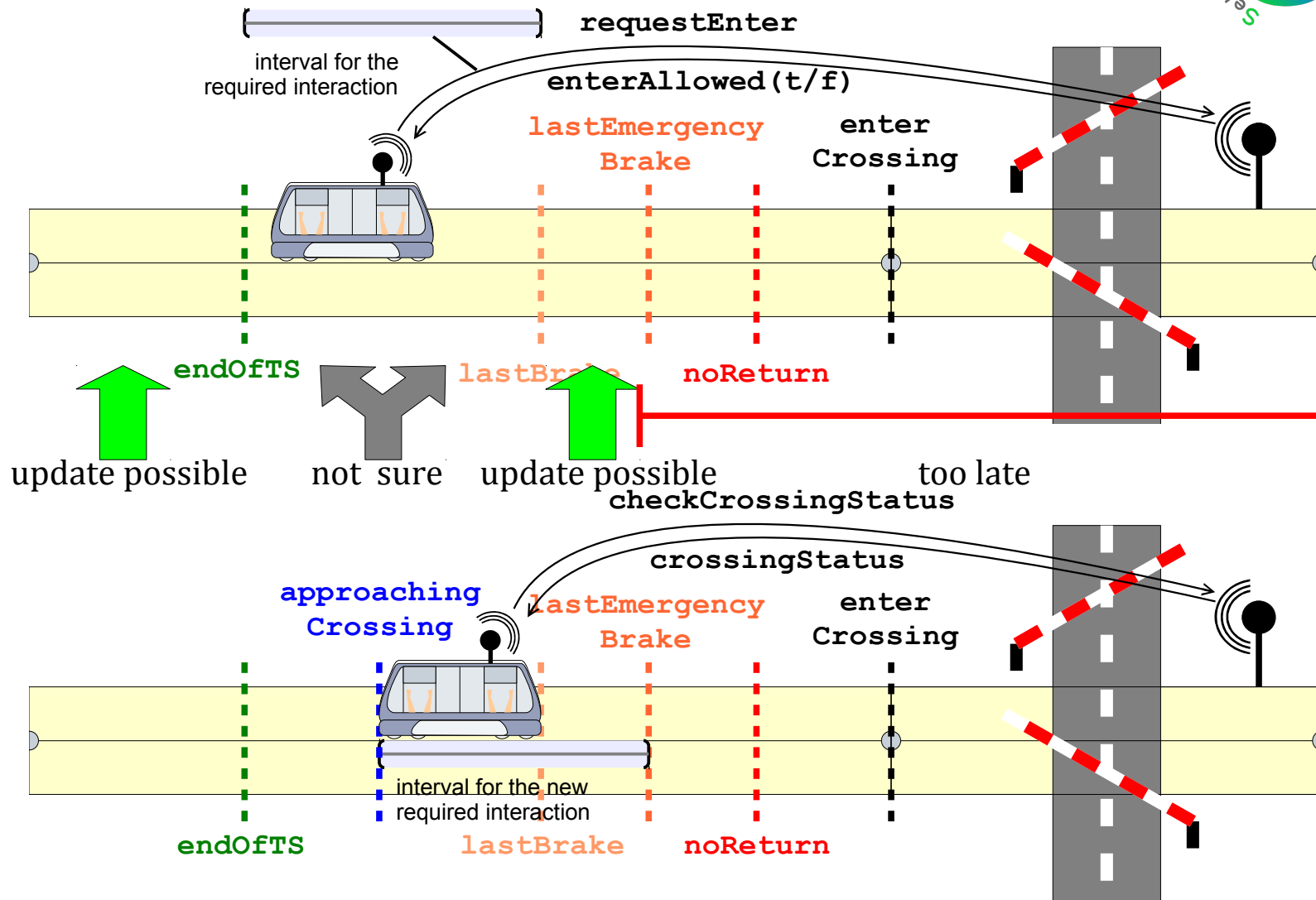
Fundamental criterion



here we can be sure that approachingCrossing has occurred (due to the new assumption)

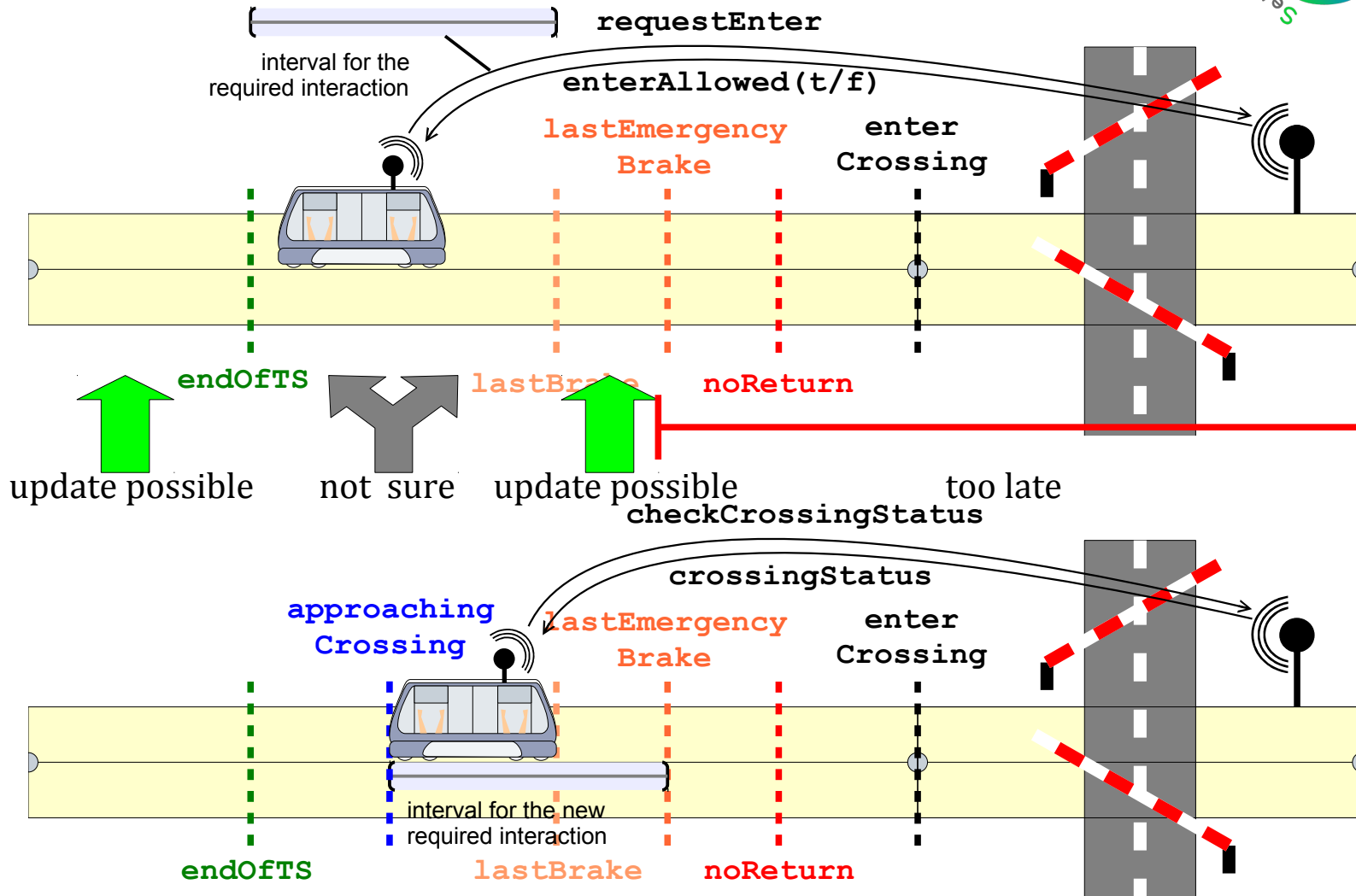


Fundamental criterion



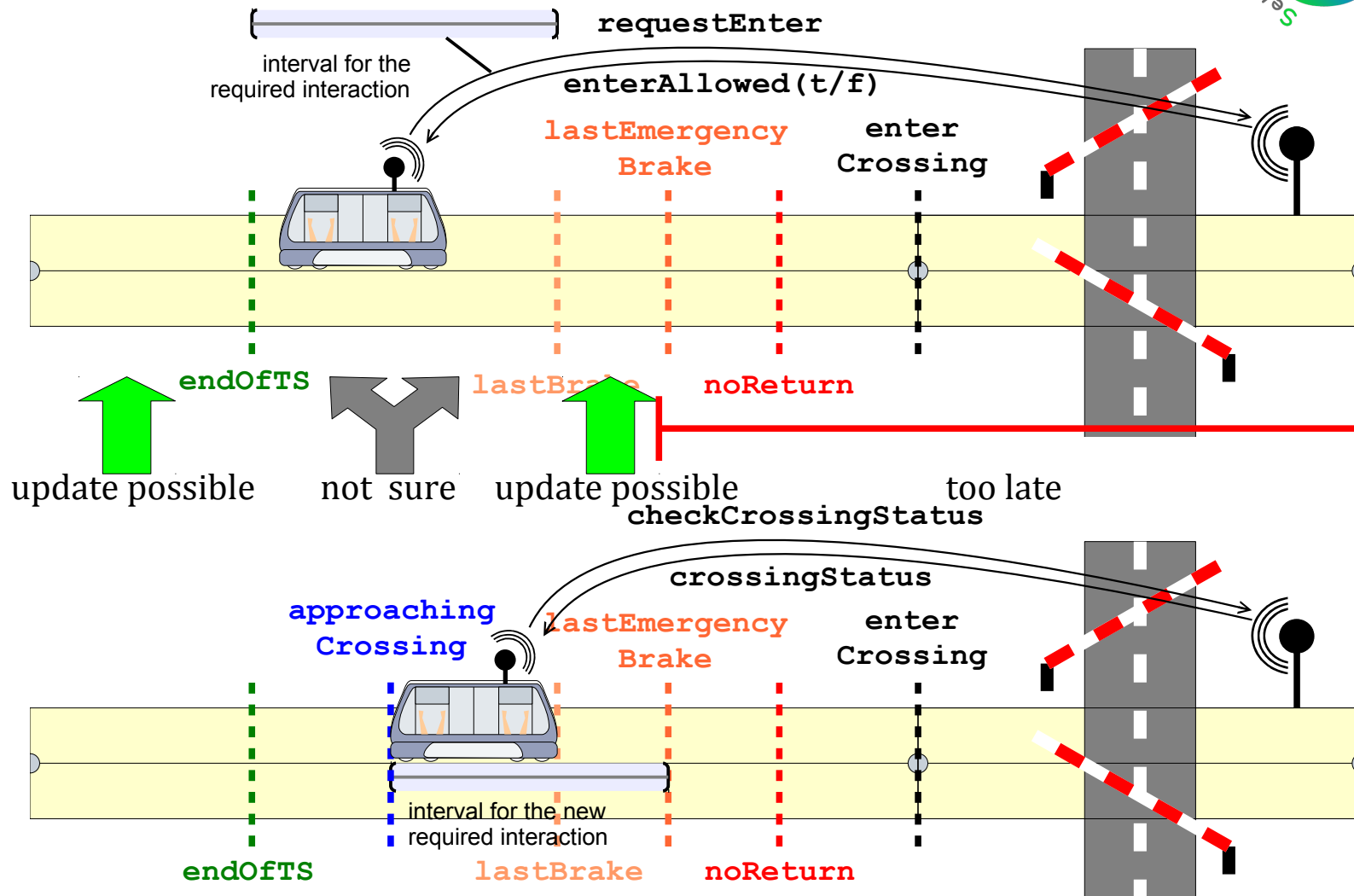


Fundamental criterion



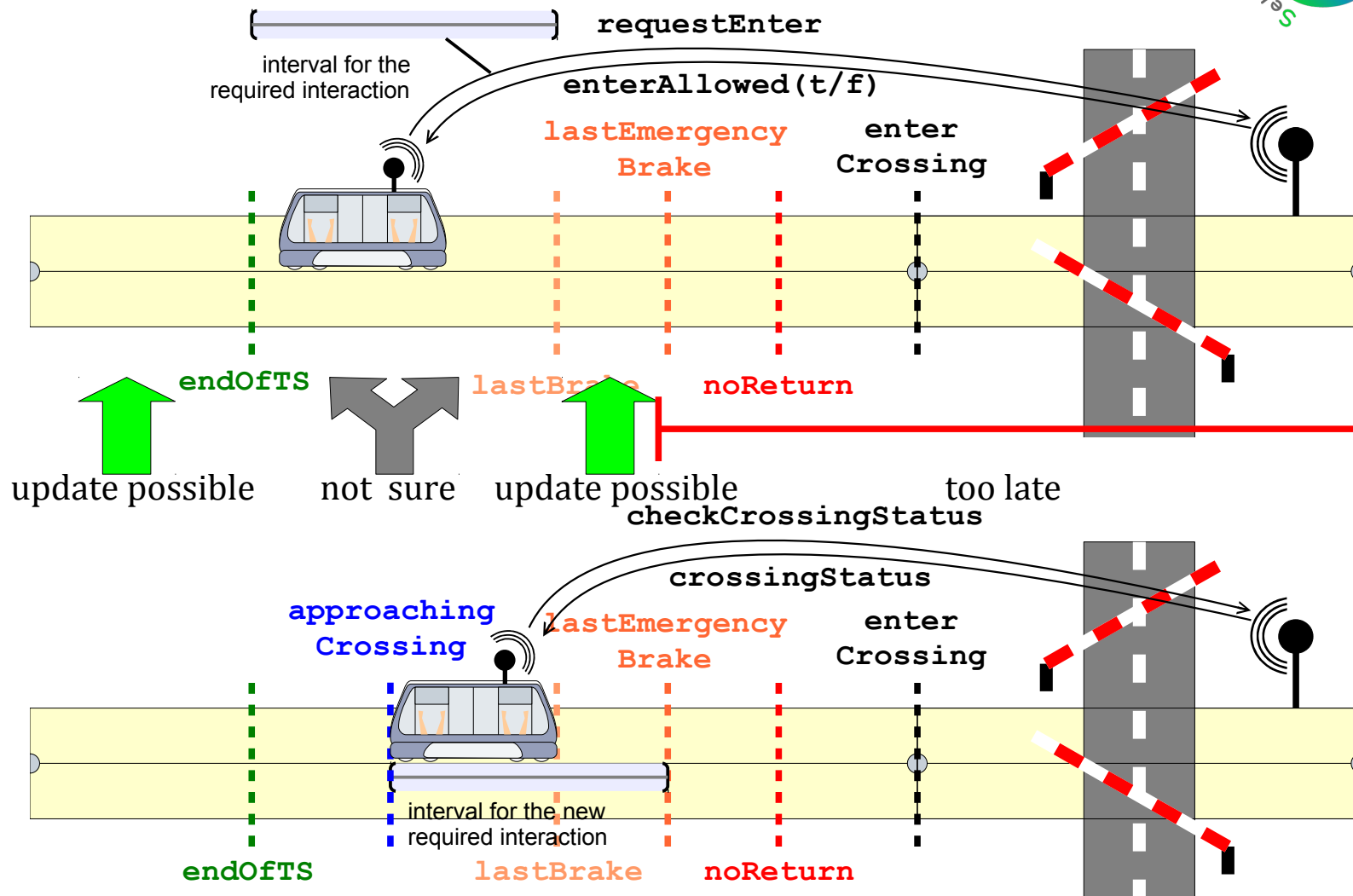
When is a system updatable?

Fundamental criterion



Intuition 1: The system must continue its past execution to satisfy S' .

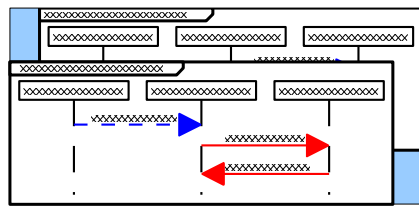
Fundamental criterion



Intuition 2: The considered past execution starts from the last time the initial state is visited.



Synthesizing Dynamically Updating Controller

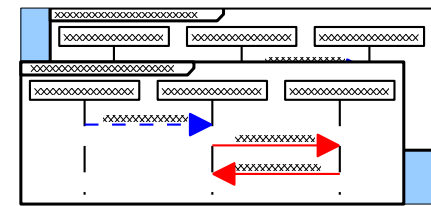


Specification S

change in requirements or
environment assumptions

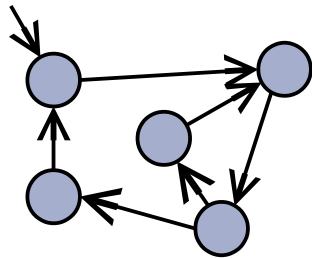


(assumption or requirement
MSDs added or removed)



Specification S'

is implemented by

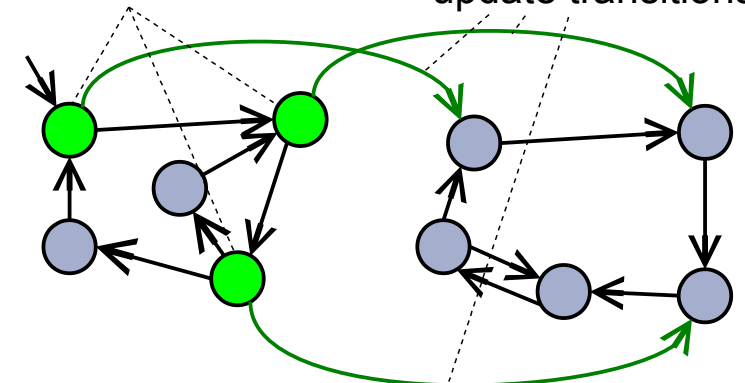


current controller

automated
synthesis

updatable states

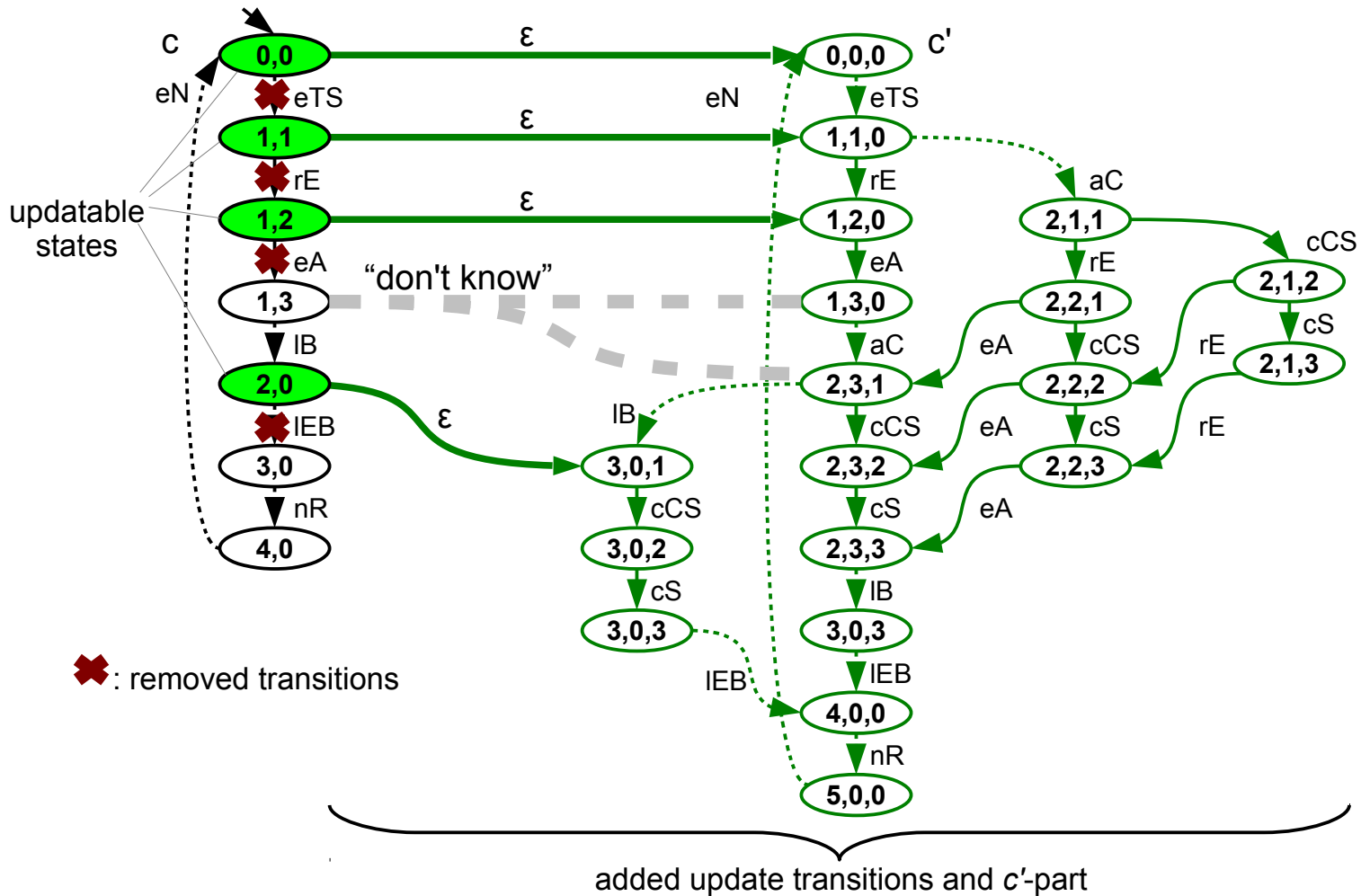
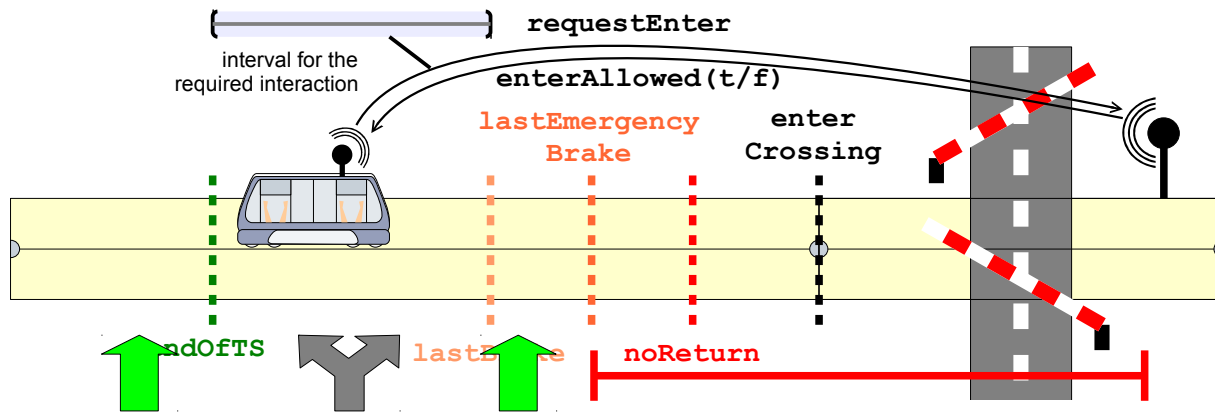
update transitions



copy of the
current controller

controller for
implementing S'

dynamically updating controller





Modeling MSD Specifications

Screenshot of an Eclipse IDE showing the modeling of MSD specifications. The main editor displays a sequence diagram titled "sd: RequestEnterAtEndOfTrackSection" with participants: env : Environment, rc : RailCab, and next : CrossingControl.

```

sequenceDiagram
    participant env as env : Environment
    participant rc as rc : RailCab
    participant next as next : CrossingControl

    env-->>rc: endOfTSS()
    rc->>next: requestEnter()
    next-->>rc: enterAllowed( isAllowed: Boolean)
    env-->>rc: lastBreak()
  
```

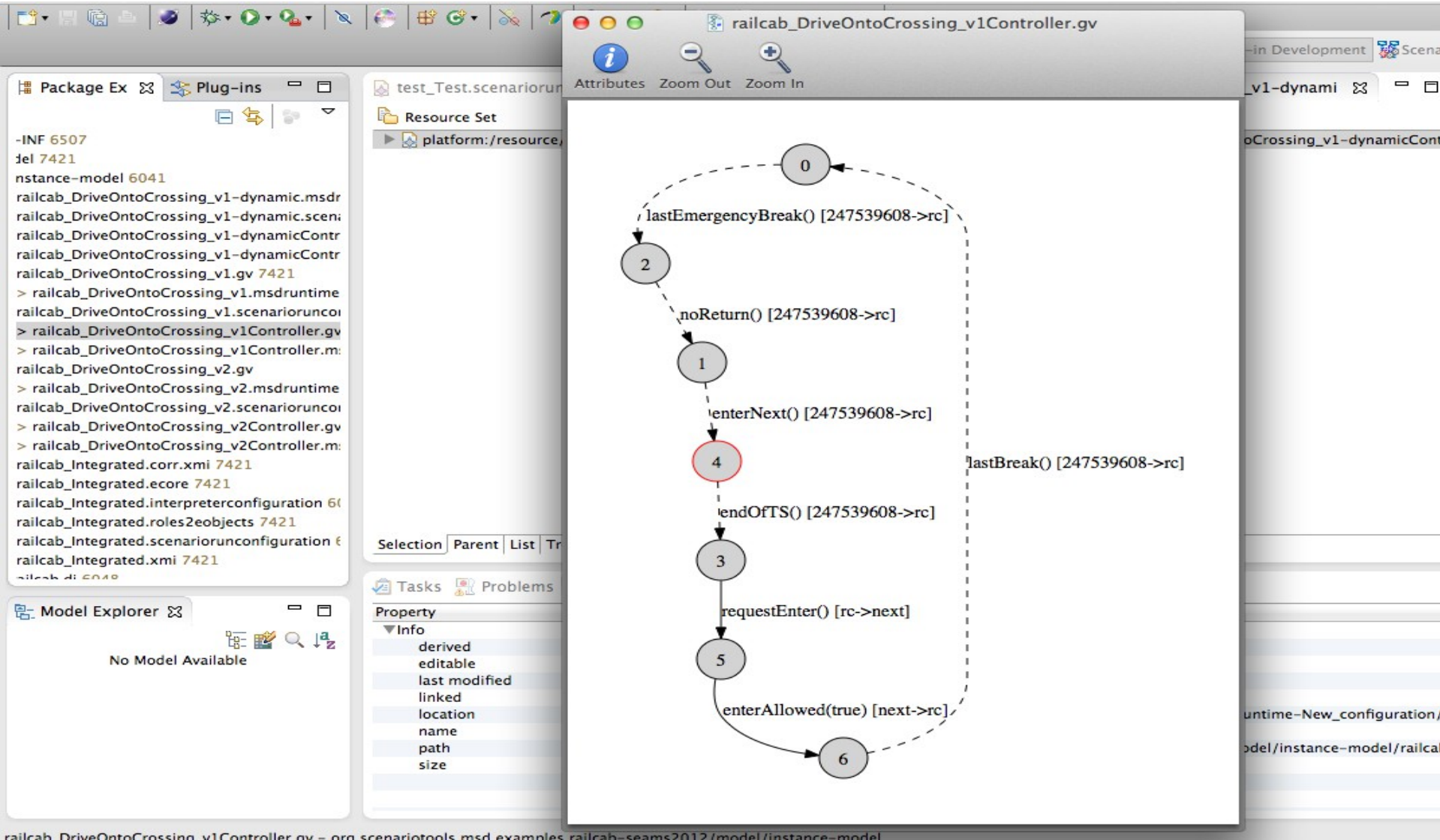
The diagram shows the following interactions:

- env : Environment** sends a dashed blue message `endOfTSS()` to **rc : RailCab**.
- rc : RailCab** sends a solid red message `requestEnter()` to **next : CrossingControl**.
- next : CrossingControl** returns a solid red message `enterAllowed(isAllowed: Boolean)` to **rc : RailCab**.
- env : Environment** sends a dashed blue message `lastBreak()` to **rc : RailCab**.

The Properties view for the selected element "RequestEnterAtEndOfTrackSection" is shown below:

UML	Name	RequestEnterAtEndOfTrackSection	
Profile	Is abstract	<input type="radio"/> true <input checked="" type="radio"/> false	Is active <input type="radio"/> true <input checked="" type="radio"/> false
Appearance	Is leaf	<input type="radio"/> true <input checked="" type="radio"/> false	Is reentrant <input checked="" type="radio"/> true <input type="radio"/> false
Advanced	Visibility	public	Specification <Undefined>
	Precondition	Postcondition	

Controller Synthesis



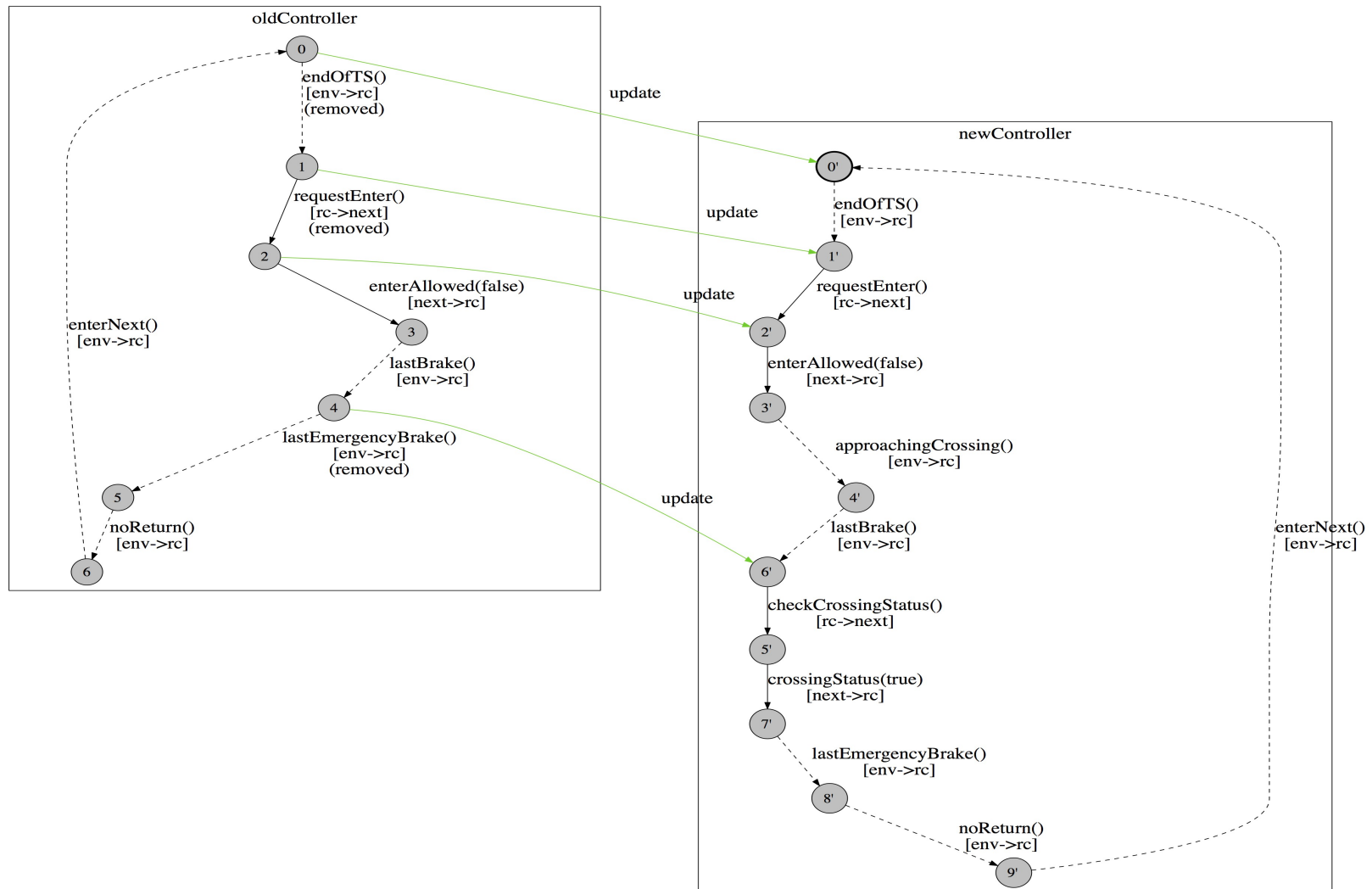
The screenshot displays a software development environment with a state machine diagram for a controller synthesis project. The diagram is titled "railcab_DriveOntoCrossing_v1Controller.gv" and shows a sequence of states (0 through 6) connected by transitions. State 4 is highlighted with a red border.

The transitions and their associated actions are:

- State 0 to State 2: `lastEmergencyBreak() [247539608->rc]`
- State 2 to State 1: `noReturn() [247539608->rc]`
- State 1 to State 4: `enterNext() [247539608->rc]`
- State 4 to State 3: `endOfTS() [247539608->rc]`
- State 3 to State 5: `requestEnter() [rc->next]`
- State 5 to State 6: `enterAllowed(true) [next->rc]`
- State 6 to State 0: `lastBreak() [247539608->rc]`

The diagram also shows a "Model Explorer" panel on the left with the text "No Model Available" and a "Property" panel on the right listing attributes such as "derived", "editable", "last modified", "linked", "location", "name", "path", and "size".

Dynamically Updating Controller





Dynamic Updates and Self-Adaptation

Can we fill the gap?



1) Self-Adaptation & Safe Updates

- Self-adaptation **must be safe**:
 - Parameter self-tuning is safe:
 - No changes in the implementation
 - Self-adaptation via composition:
 - Needs to rely on stateless components or services
 - Self-adaptation of stateful applications:
 - Requires quiescence

Can we do better?

- Our criterion of updatable states can help:
 - **Automatic identification** of safe updatable states
 - **More timely adaptation**
 - No need to wait for quiescence

2) Automatic generation of self-adaptive systems

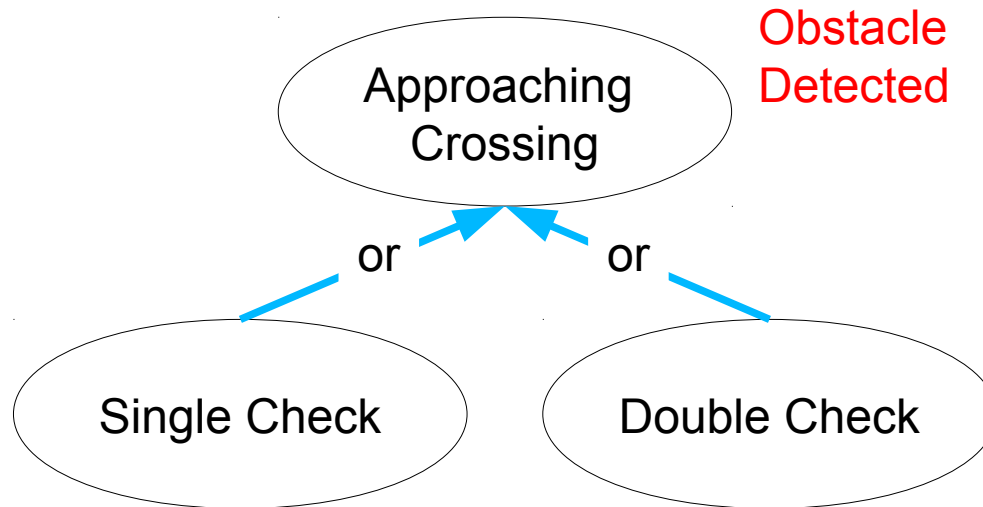


Integrating goal models and scenario-based specifications:

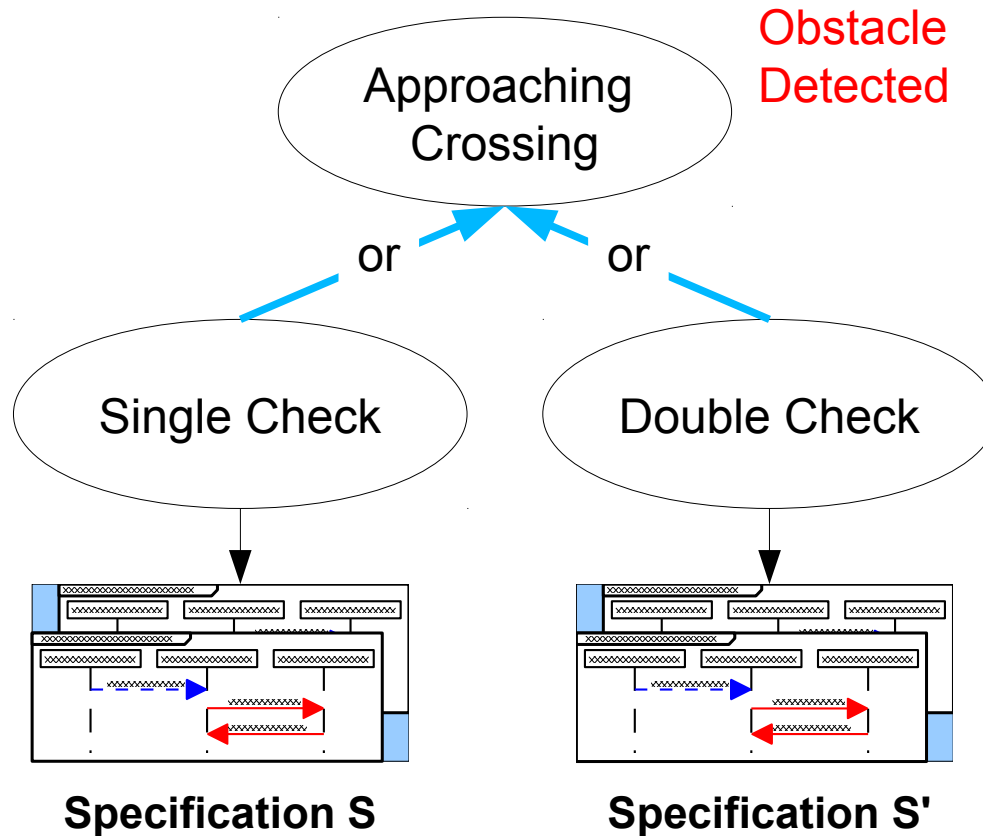
- Goal models defines the adaptation
 - The alternative goals and tasks
 - The context triggering the adaptation
- MSD specification can define the behavior of goals and tasks
- The synthesis approach can automatically generate
 - the controller of each adaptive behavior
 - and the update transitions between them



RailCab Example

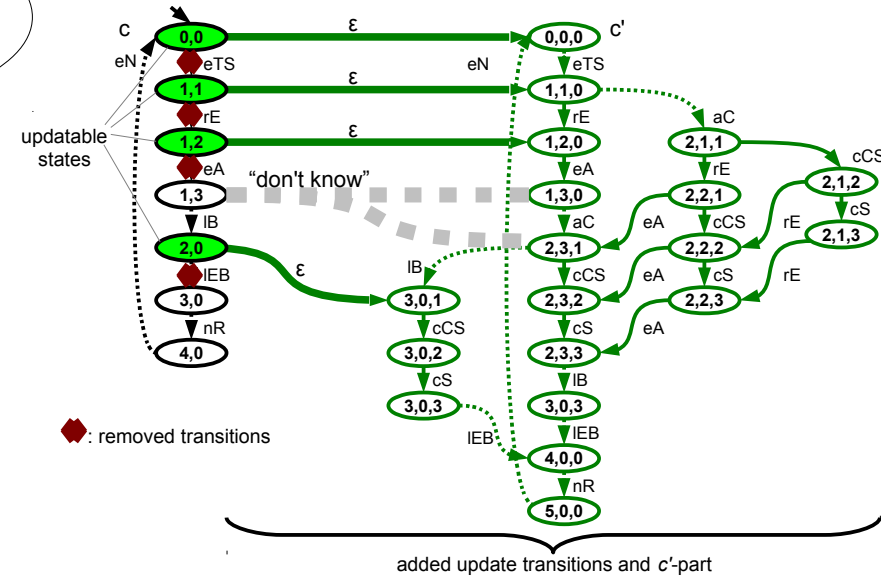
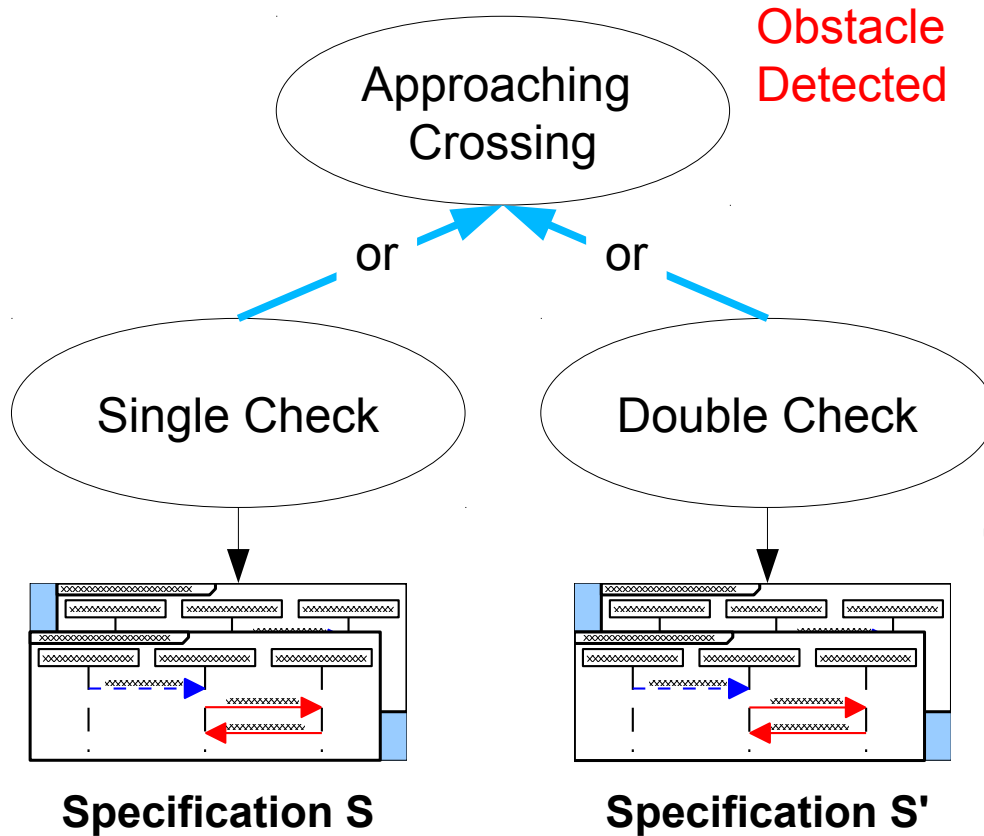


RailCab Example



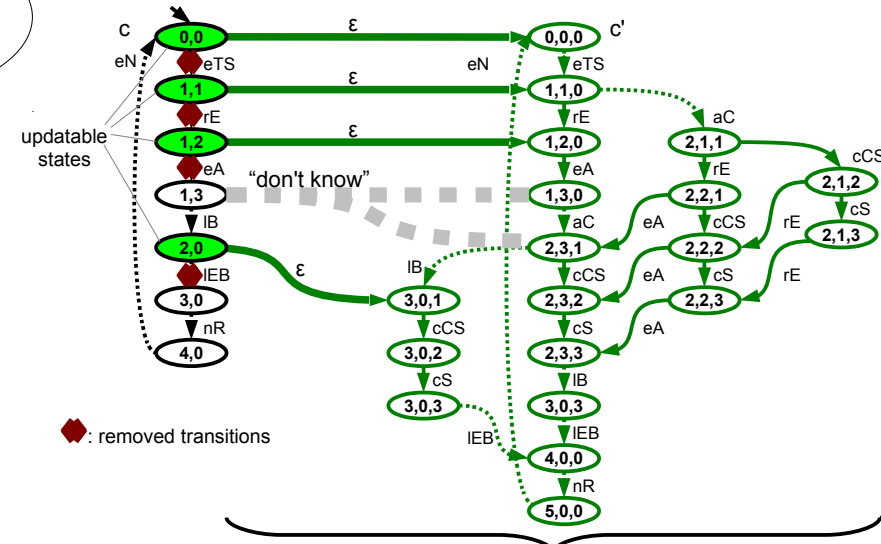
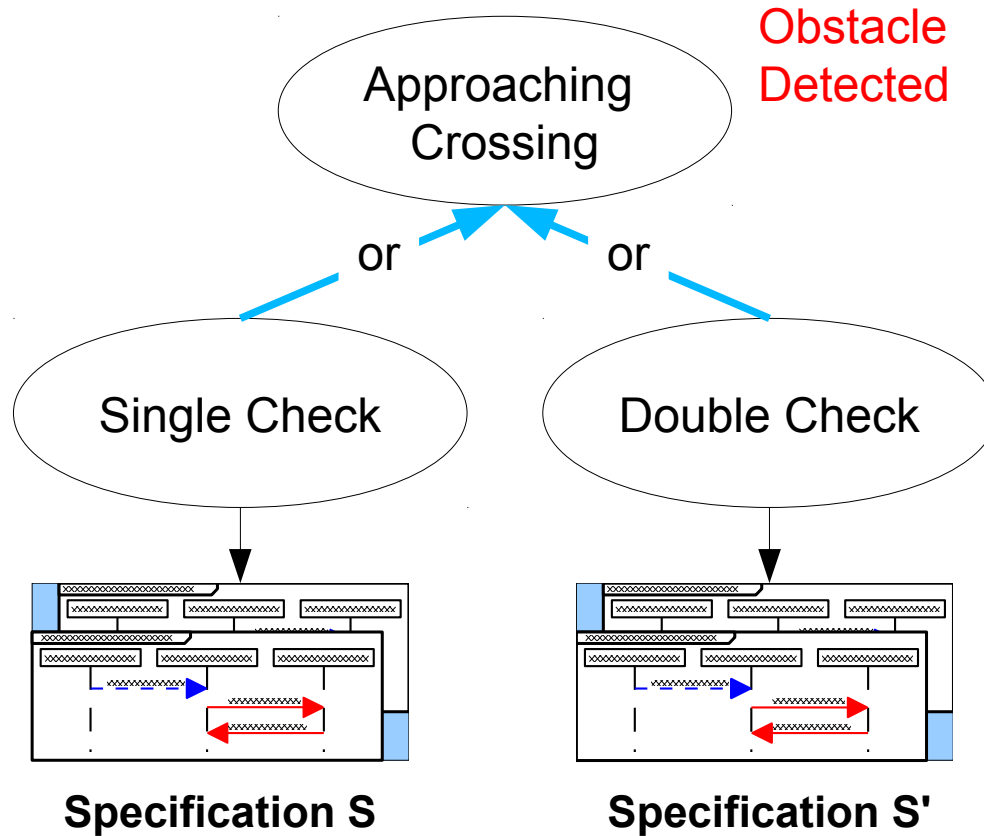


RailCab Example





RailCab Example



Under which condition the update transitions are reversible?

3) Dynamic Updates of Self-Adaptive Systems



Applying the approach to the MAPE-K

- Adding new goals and associated behavior
 - No need to manually define the K
 - It can be derived from the specification
 - No need to manually identify updatable states
 - Automatic synthesis of unanticipated adaptive behavior



3) Dynamic Updates of Self-Adaptive Systems

Applying the approach to the MAPE-K

- Adding new goals and associated behavior
 - No need to manually define the K
 - It can be derived from the specification
 - No need to manually identify updatable states
 - Automatic synthesis of unanticipated adaptive behavior
- Dynamic Updates of Monitoring capability?



ありがとう