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

Valerio Panzica La Manna – Politecnico di Milano, Italy

panzica@elet.polimi.it

EASSy 2013 – Shonan Meeting – 12 Sep 2013

Dynamic Updates…

deep se
dependable evolvable pervasive software engineering group

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

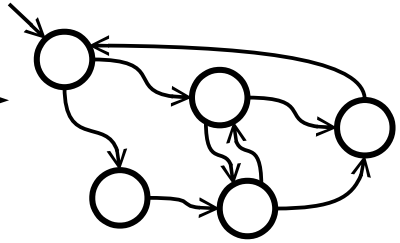
Engineering software systems able to evolve at run-time

- 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

Finite State Controller



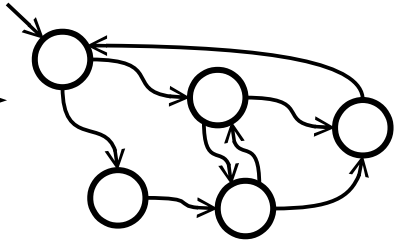
operate

uncontrollable environment

Open Reactive Systems

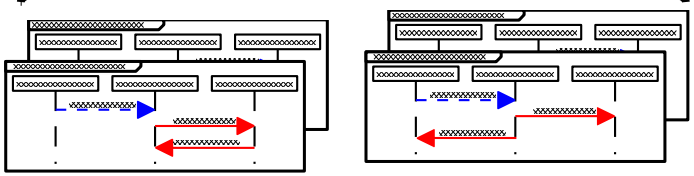
controlled by

Finite State Controller



implements

Specification



Environment Assumptions

Requirements

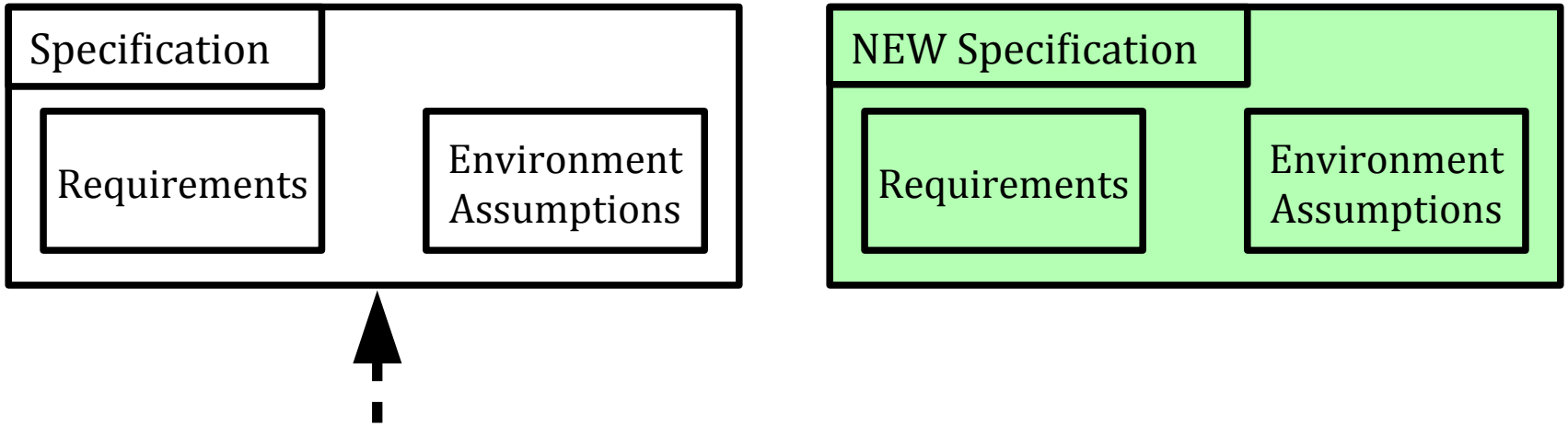
Sequences of events that can occur in the environment

Sequences of events allowed in the system

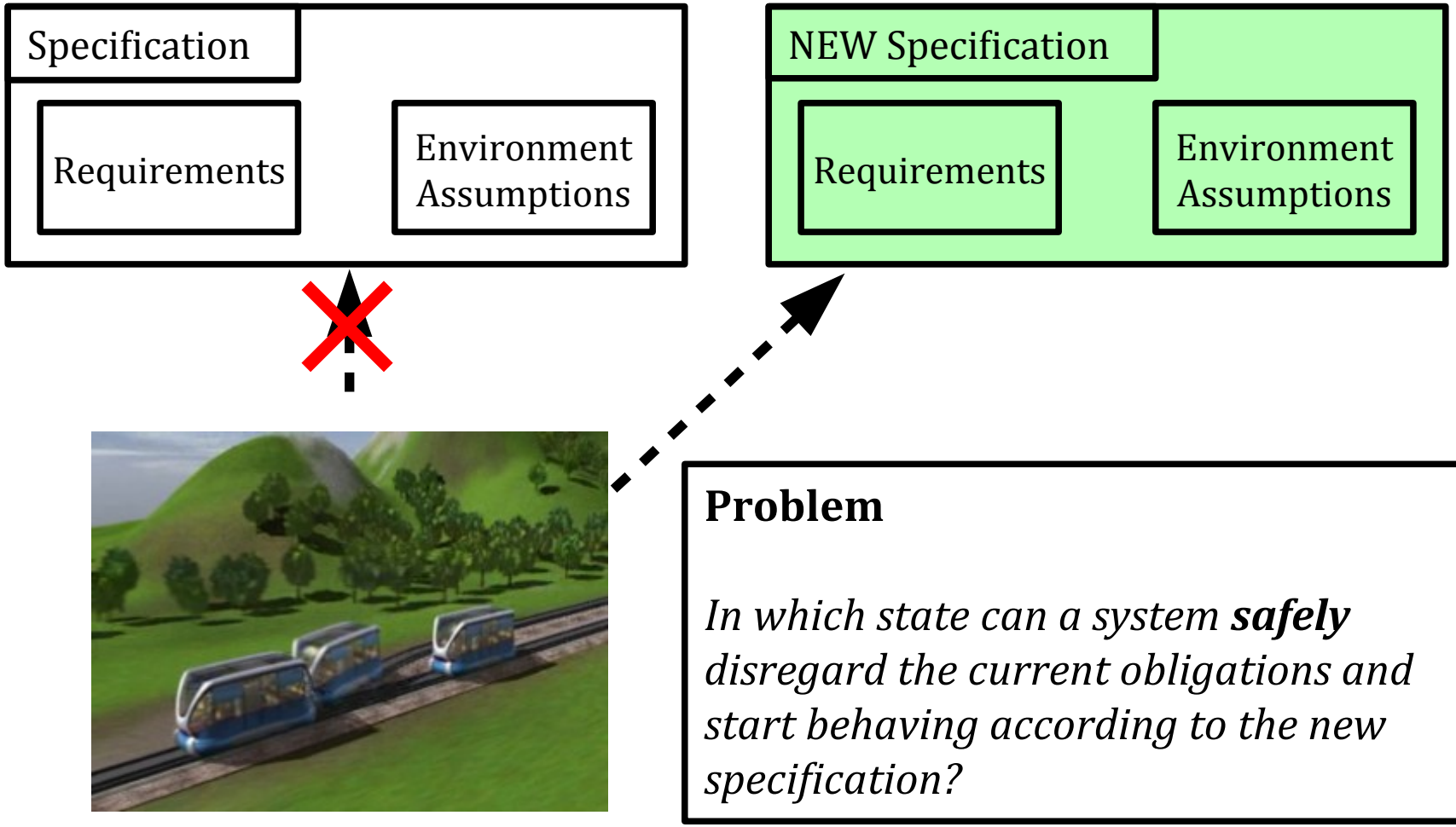
operate

uncontrollable environment

A specification oriented perspective



A specification oriented perspective



Problem

*In which state can a system **safely** disregard the current obligations and start behaving according to the new specification?*

Our contribution

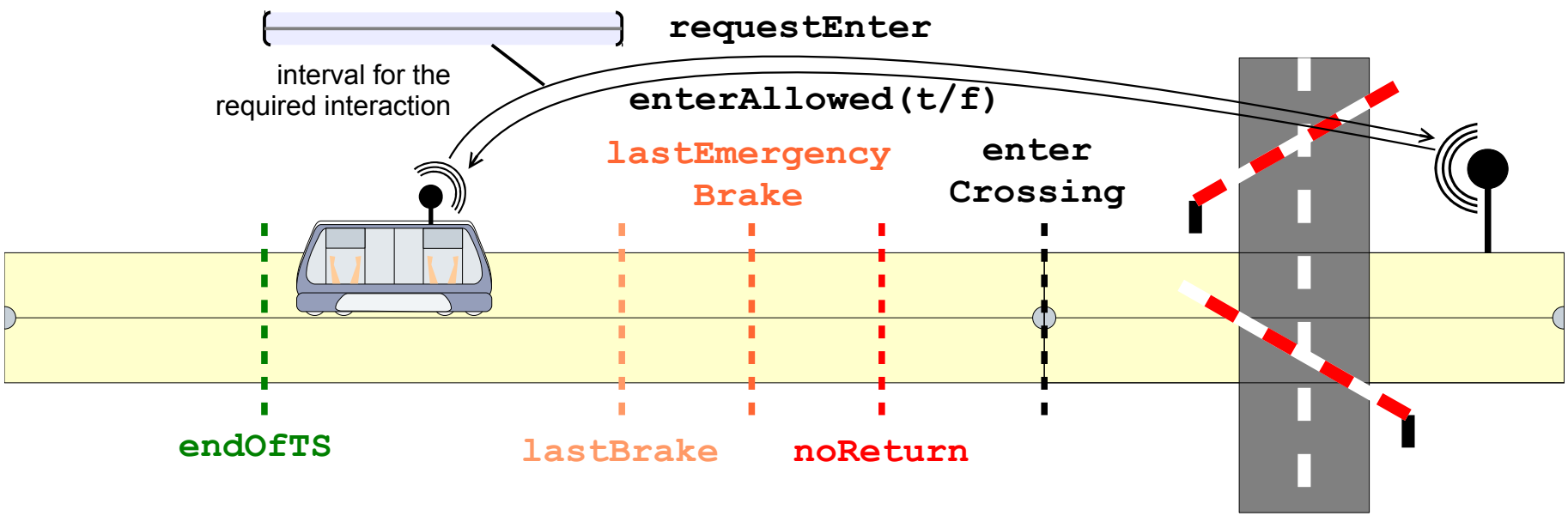
- Formalizing criteria for safe dynamic updates.
  - Definition of updatable states [1].
  - Additional criteria for more timely updates [2].
- Approach to automatically construct dynamically updating systems from changes in MSD specification [1, 2].
- Tool realization as part of ScenarioTools [2].

[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.

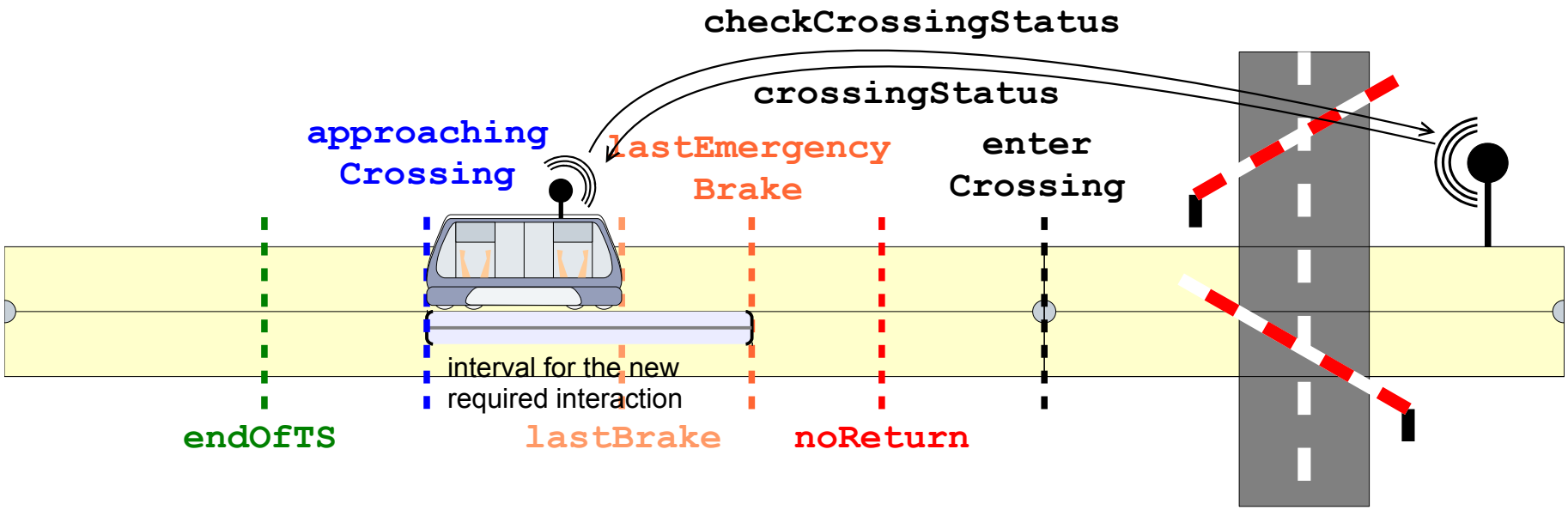
Example: Current Specification

- **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 (...)

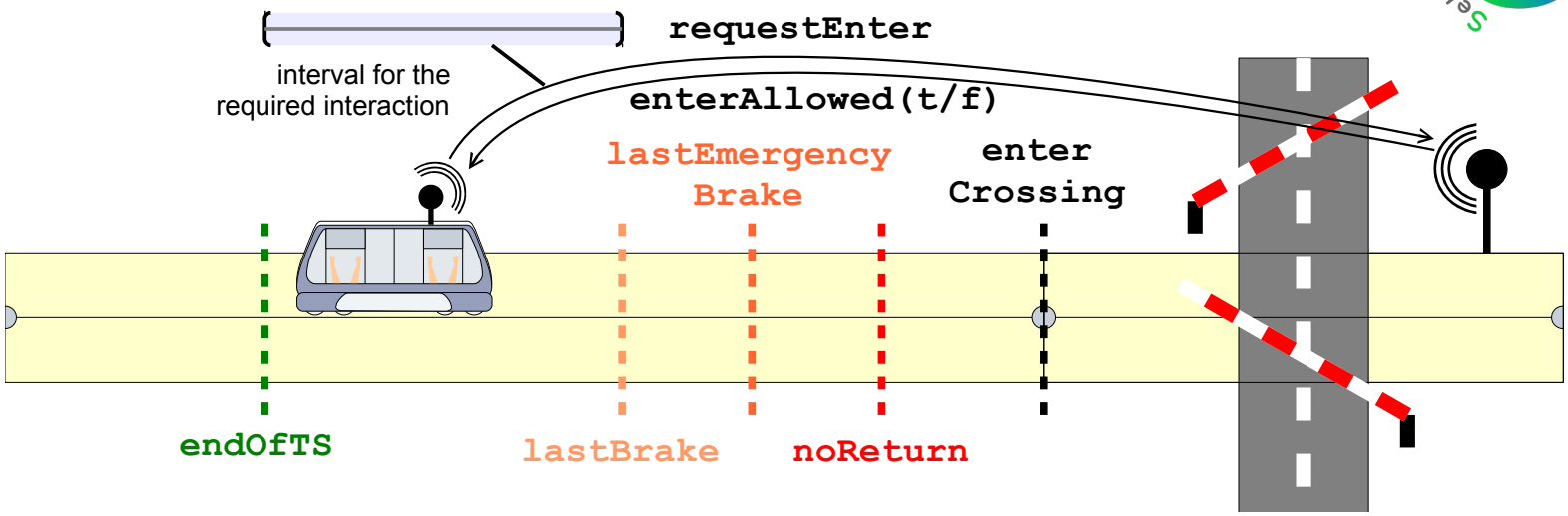


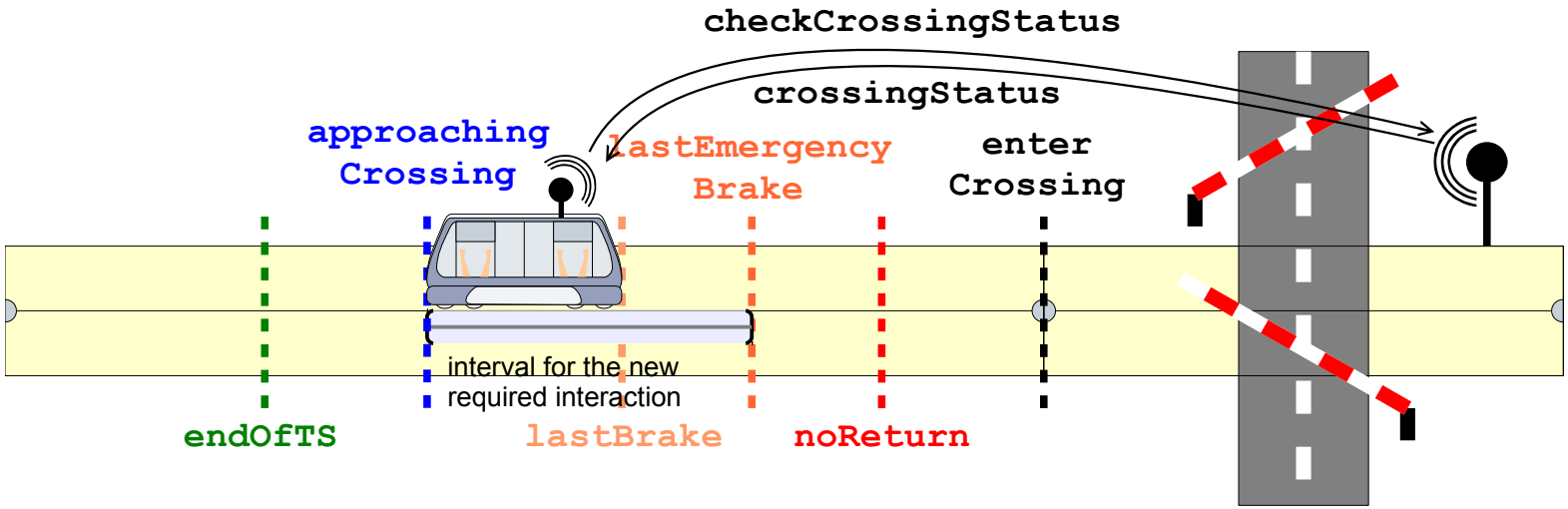
Example: Specification Change

- **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.

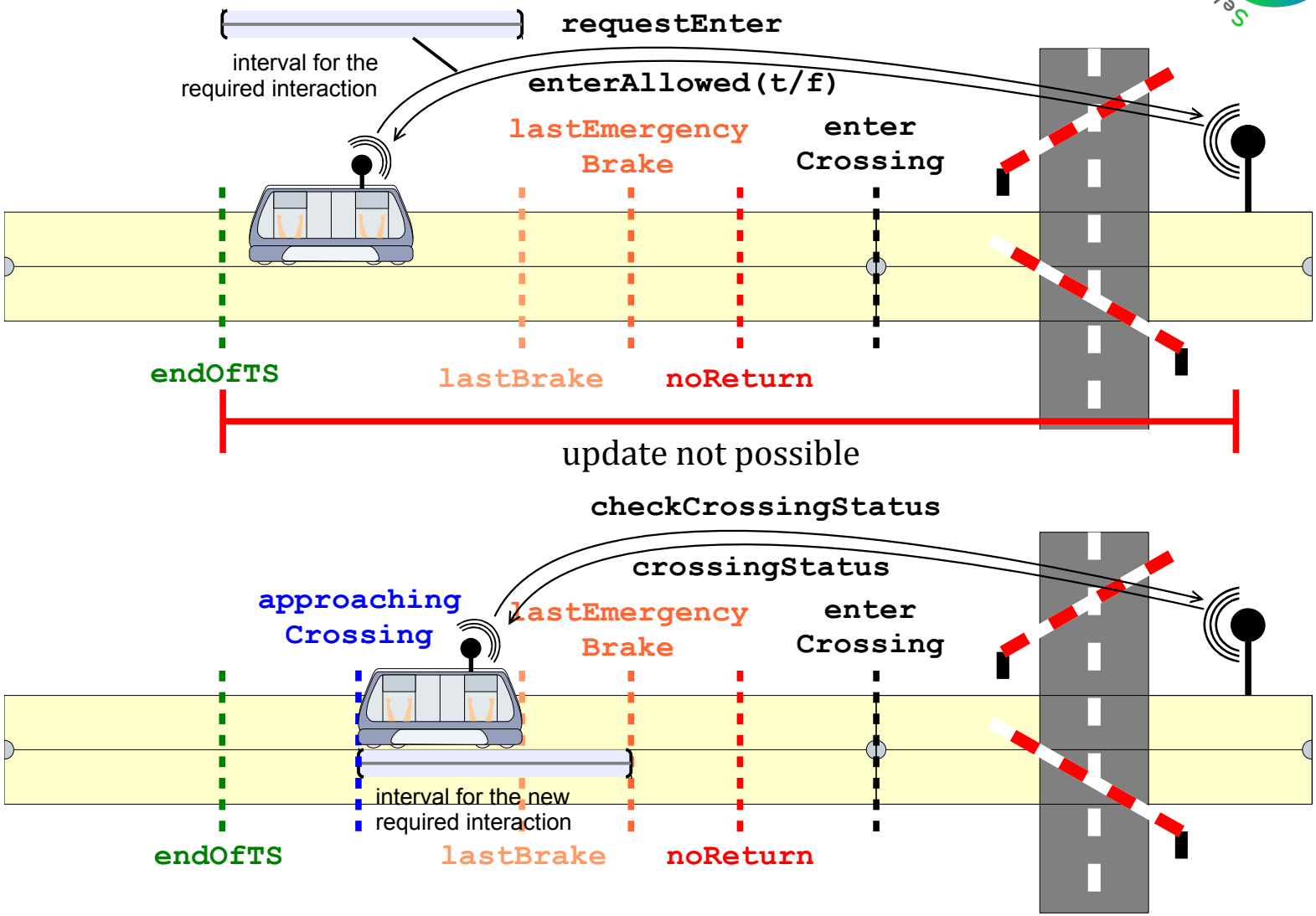


Example: Offline Update

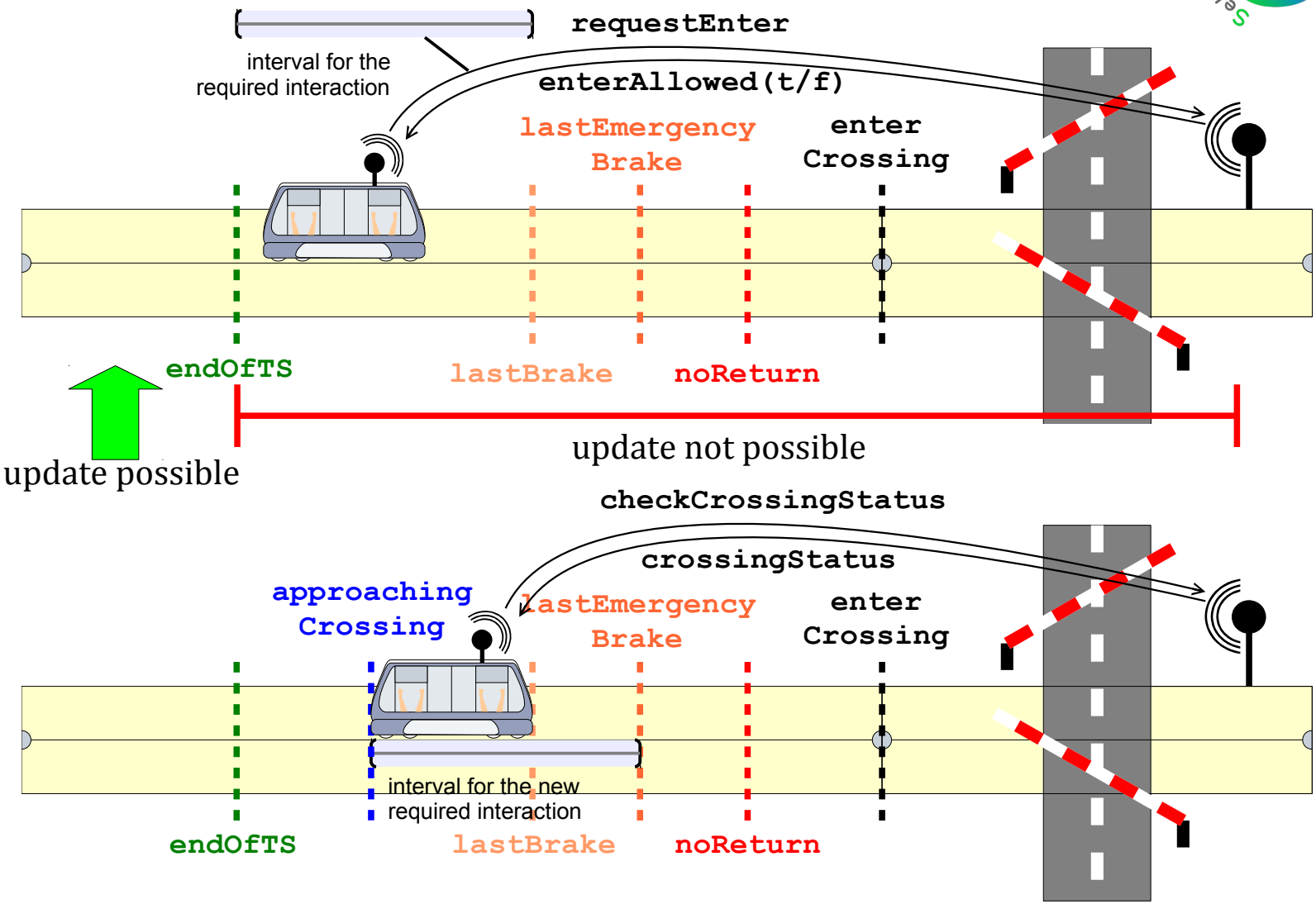




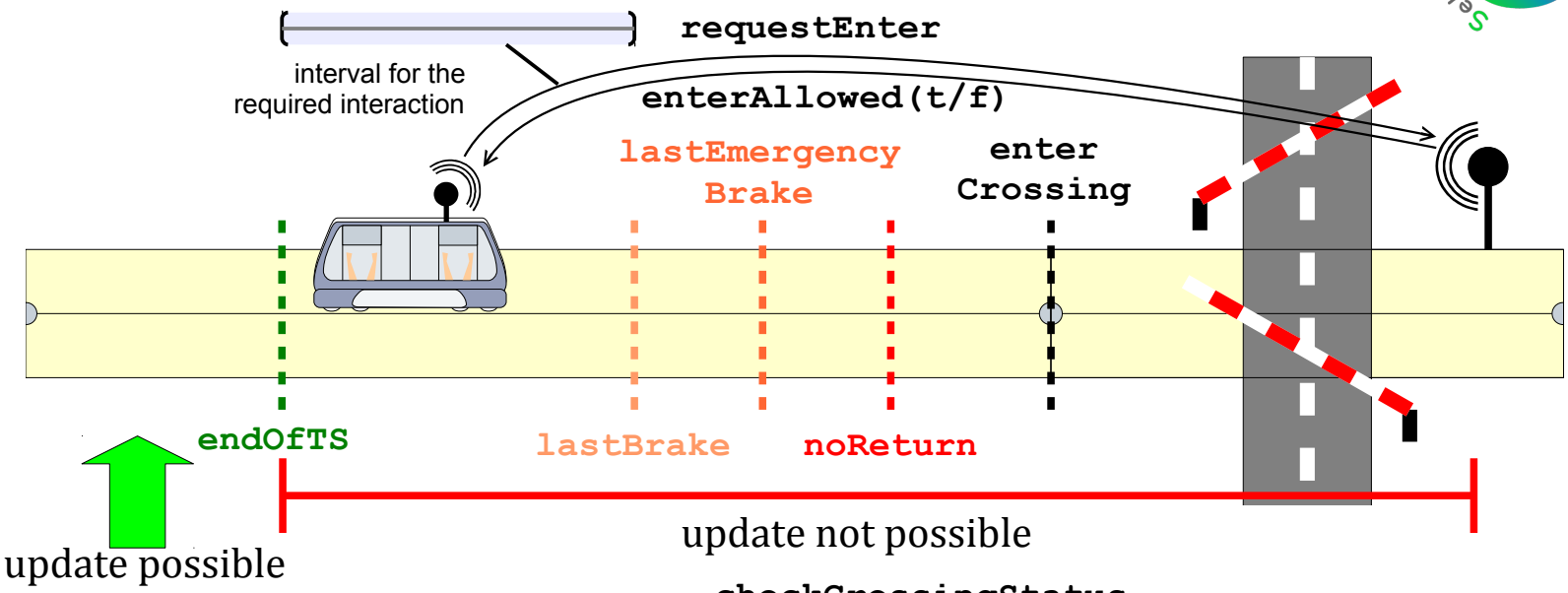
Example: Offline Update

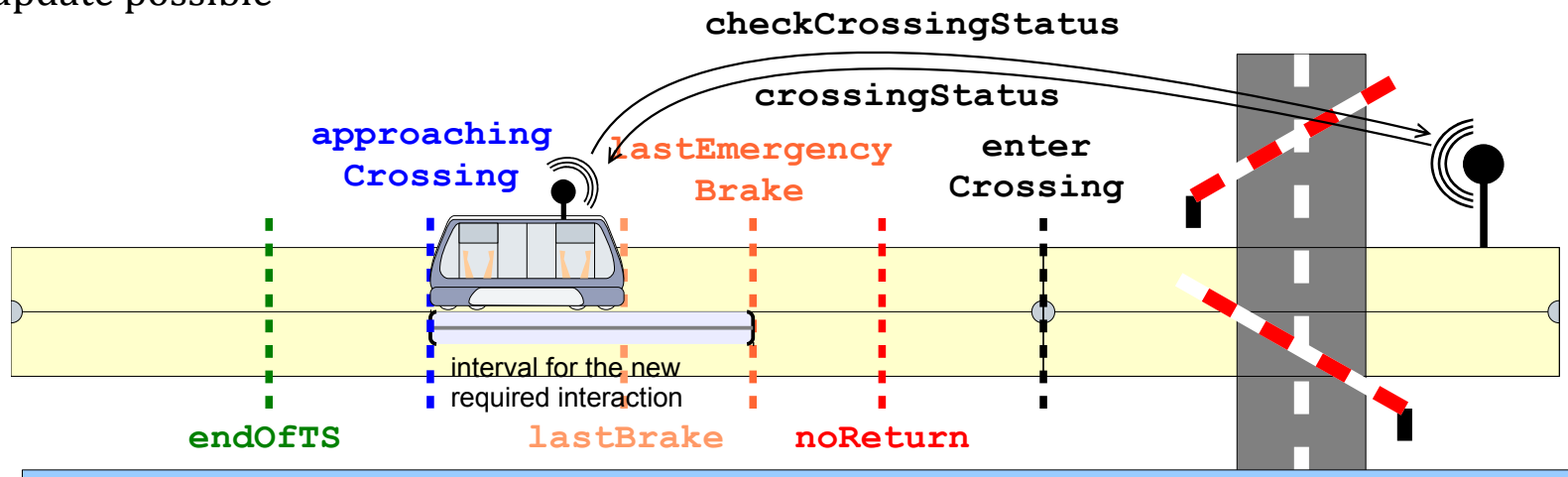


Example: Offline Update



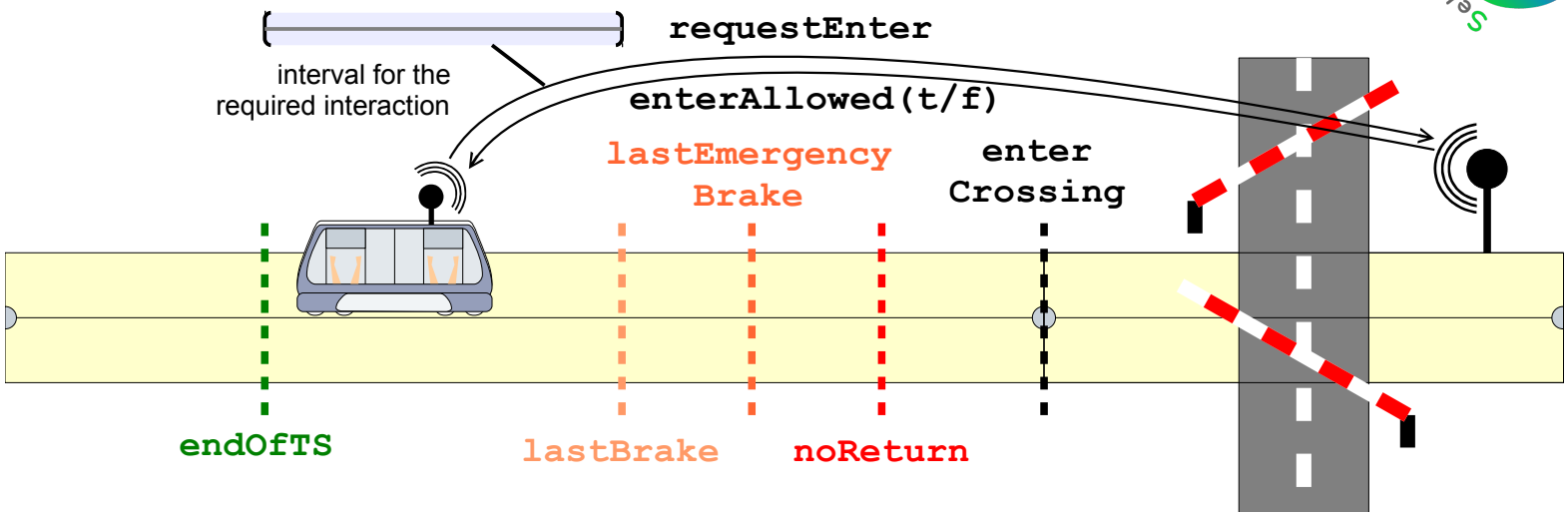
Example: Offline Update

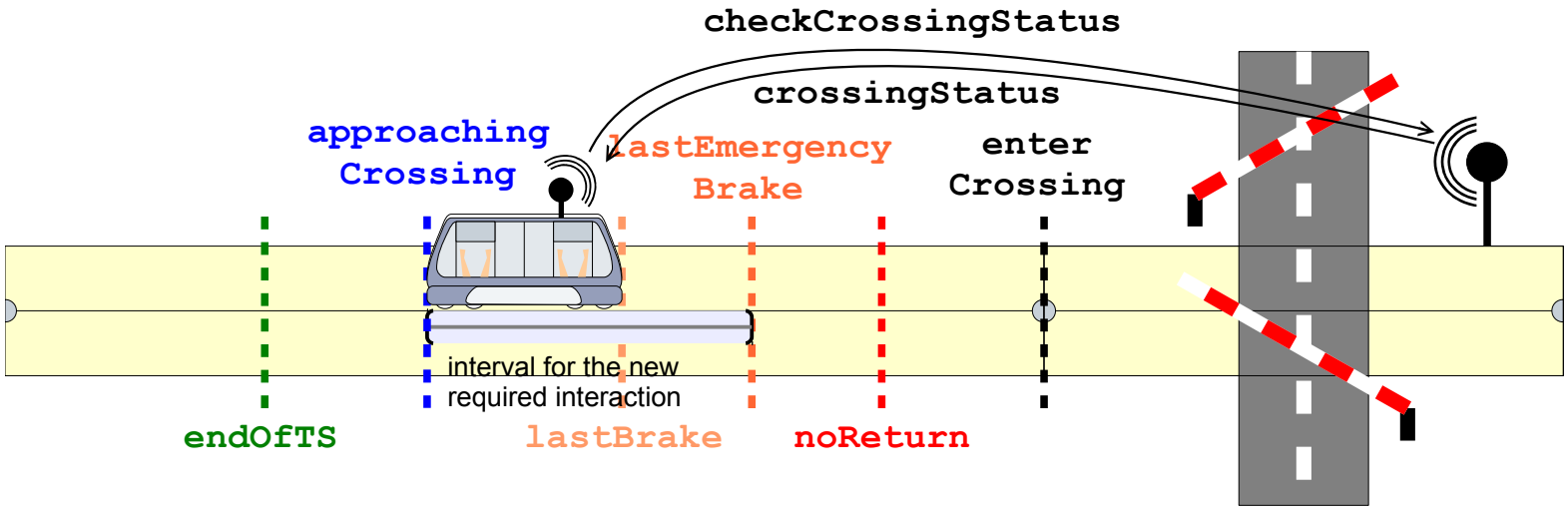




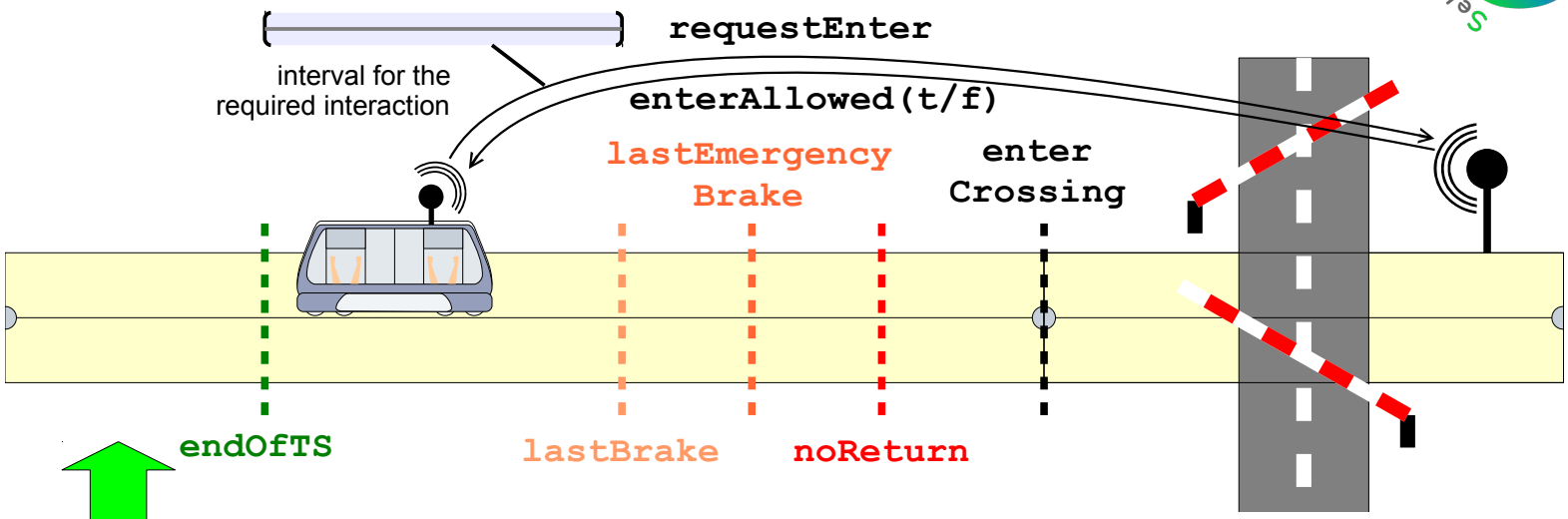
How to dynamically update the RailCab to the new behavior at run-time?

Fundamental criterion

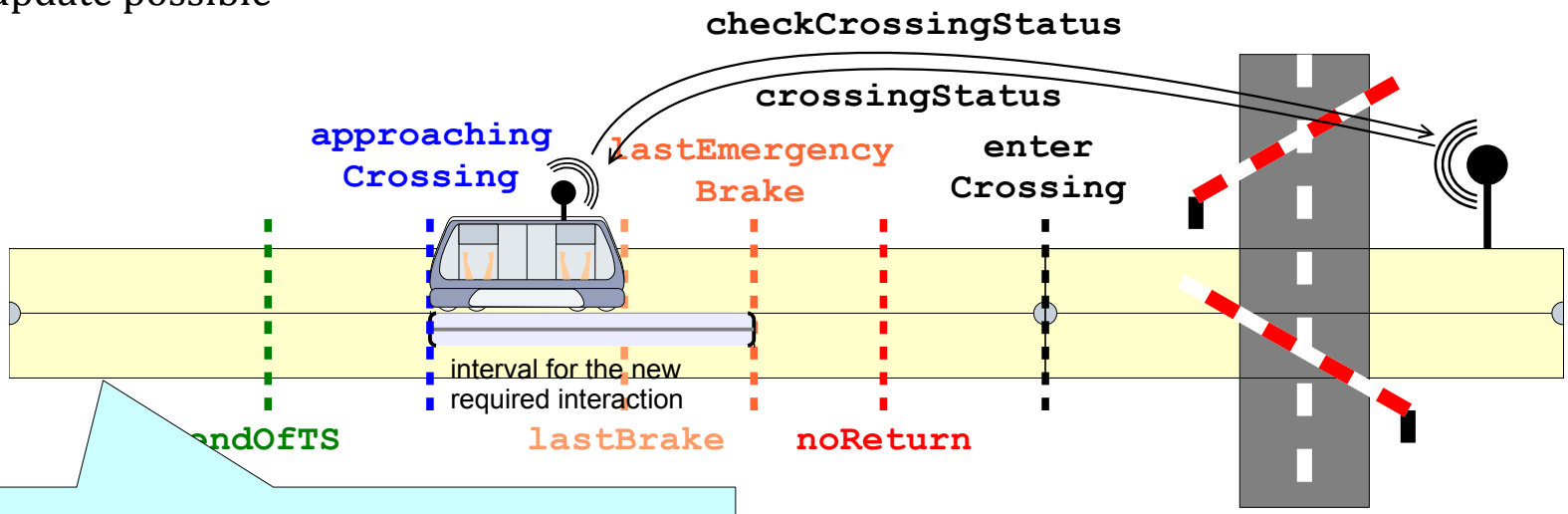




Fundamental criterion

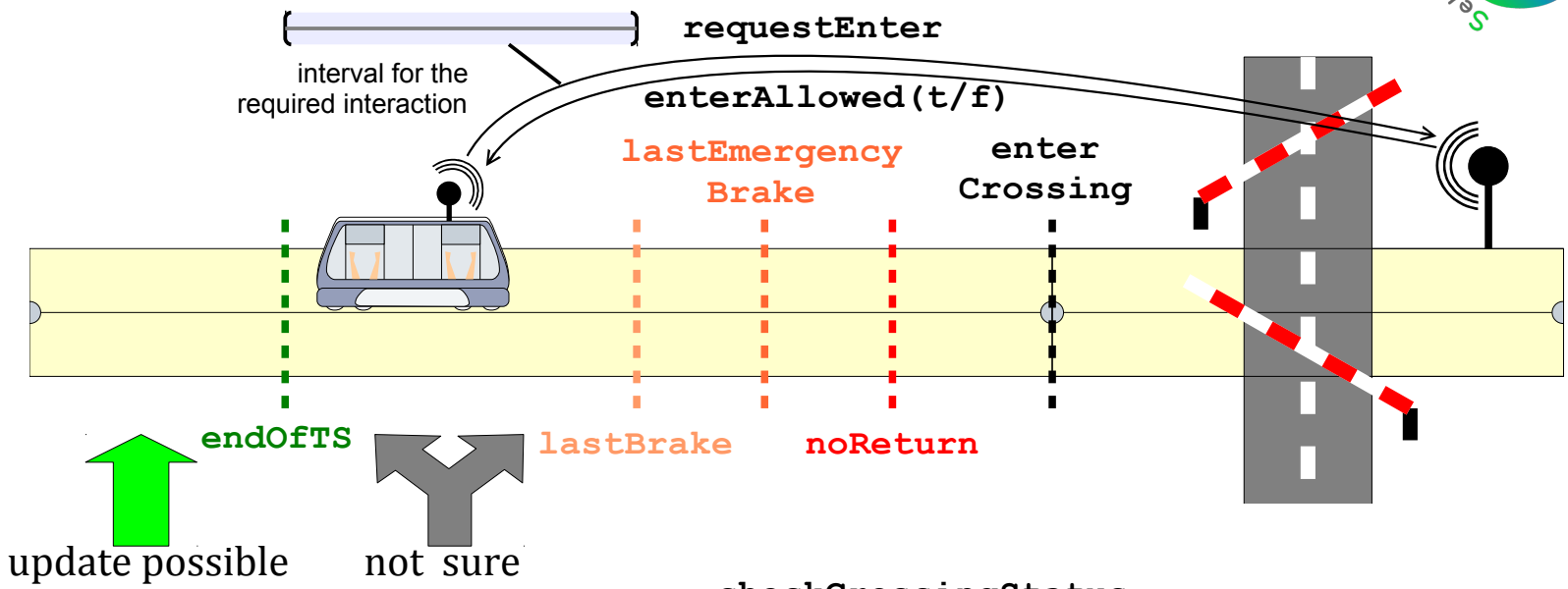


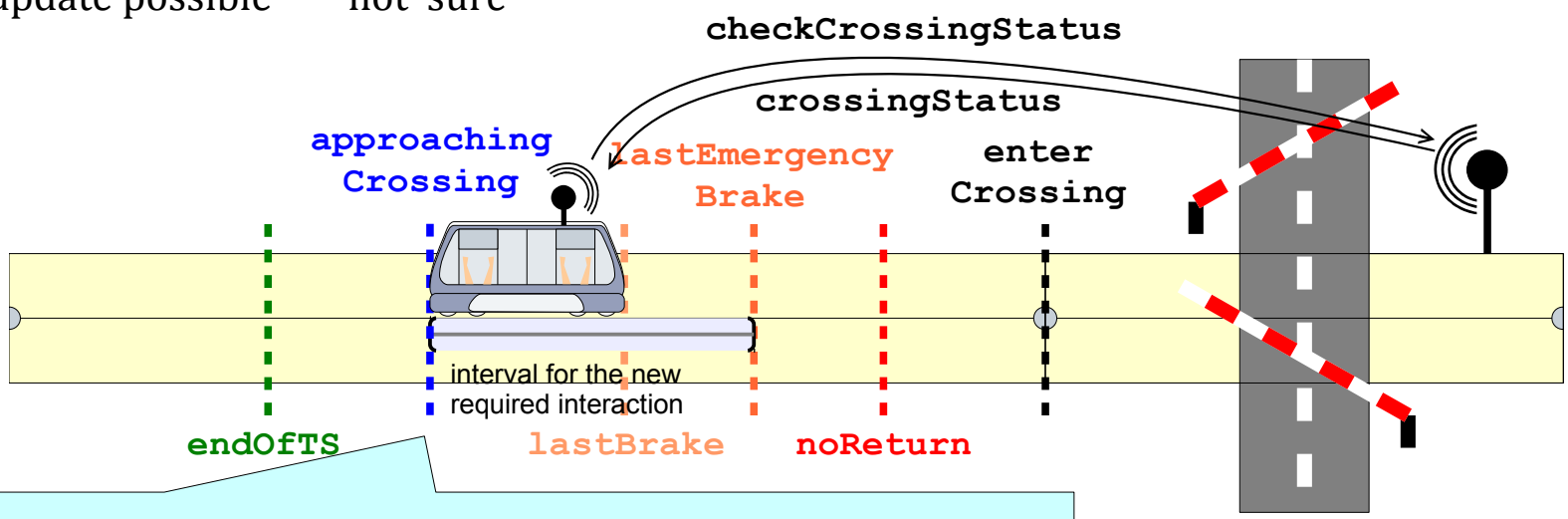
update possible



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

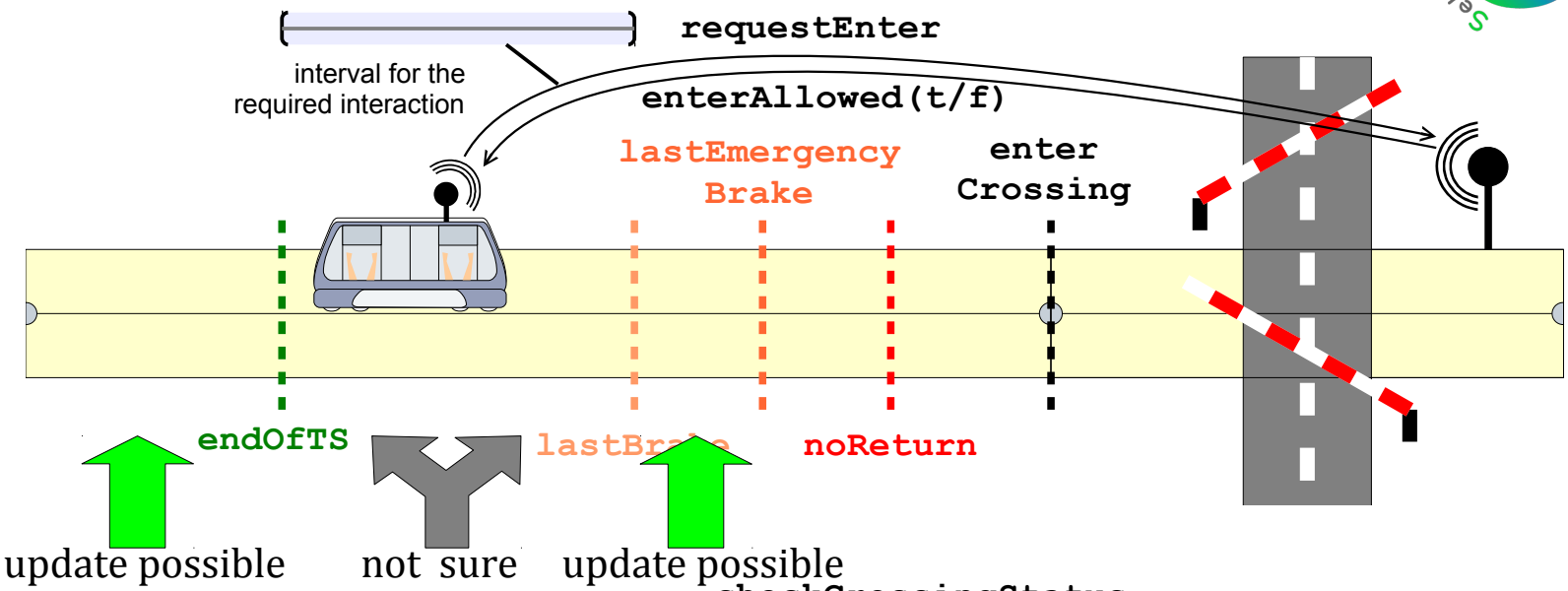
Fundamental criterion

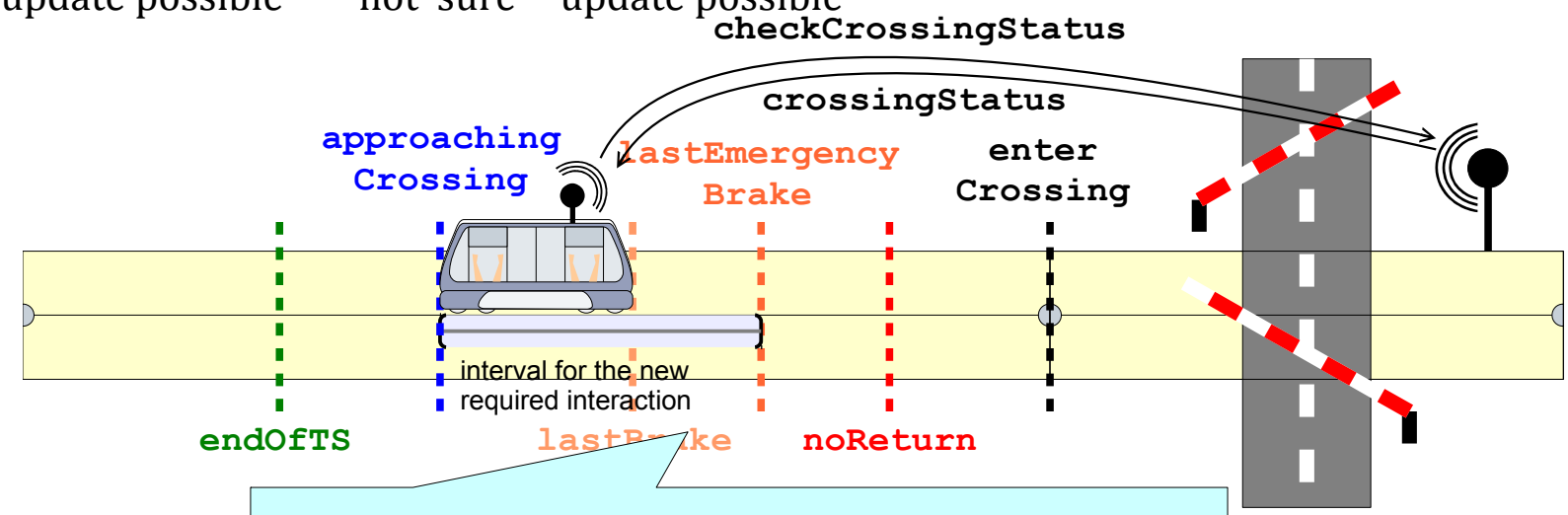




the running system does not remember this event since it is not present in the old assumptions

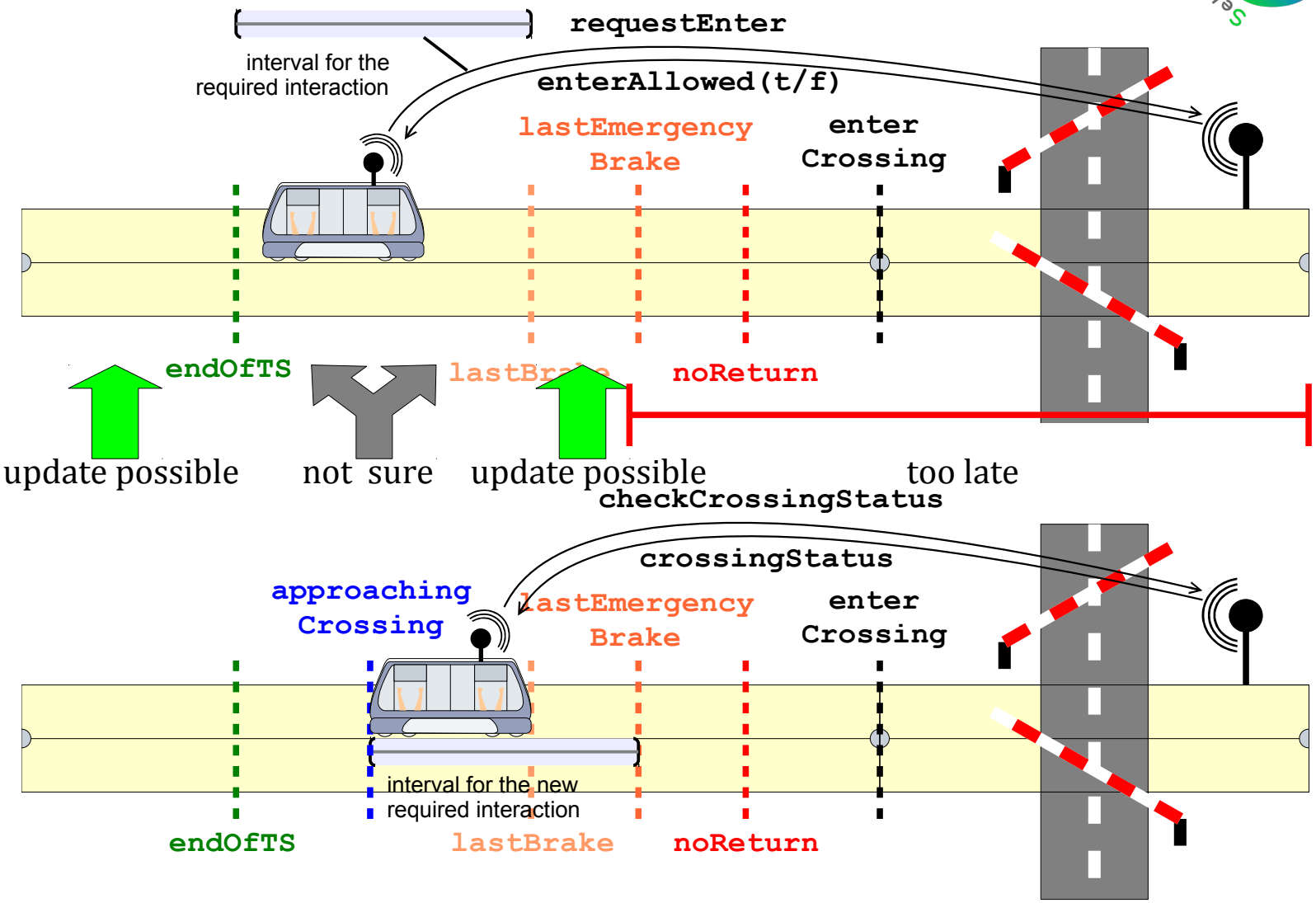
Fundamental criterion



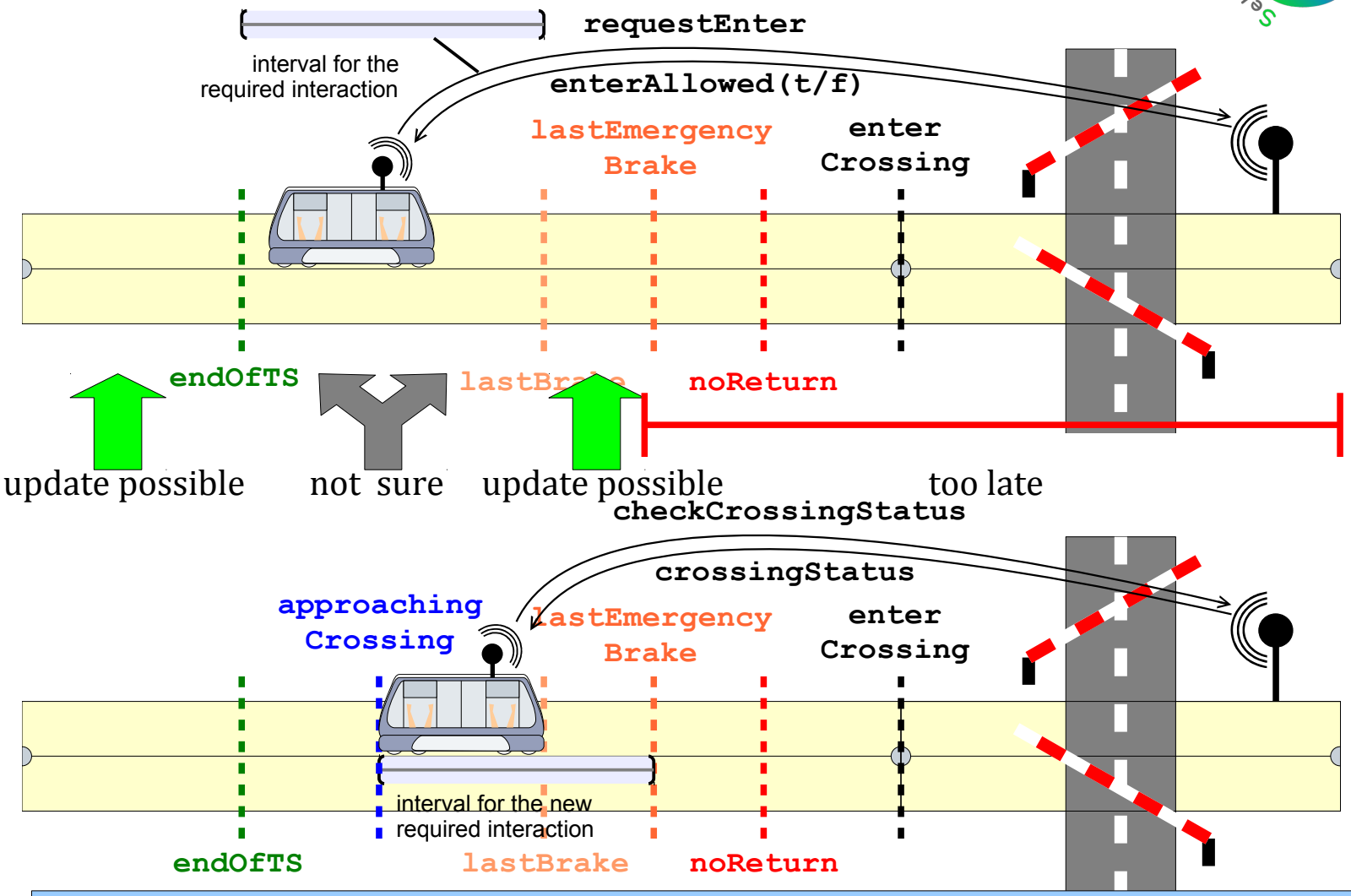


here we can be sure that approachingCrossing has occurred (due to the new assumption)

Fundamental criterion

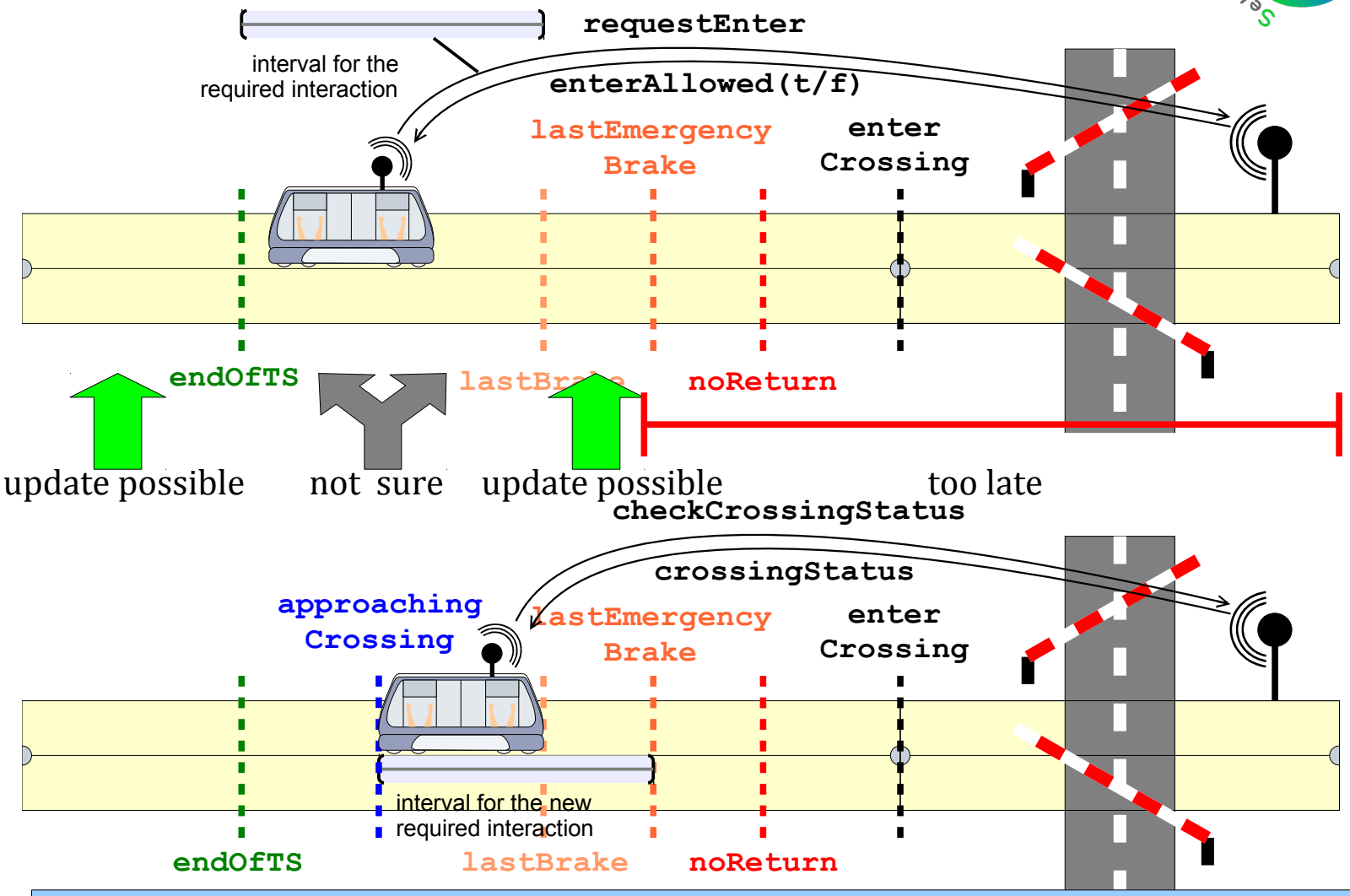


Fundamental criterion



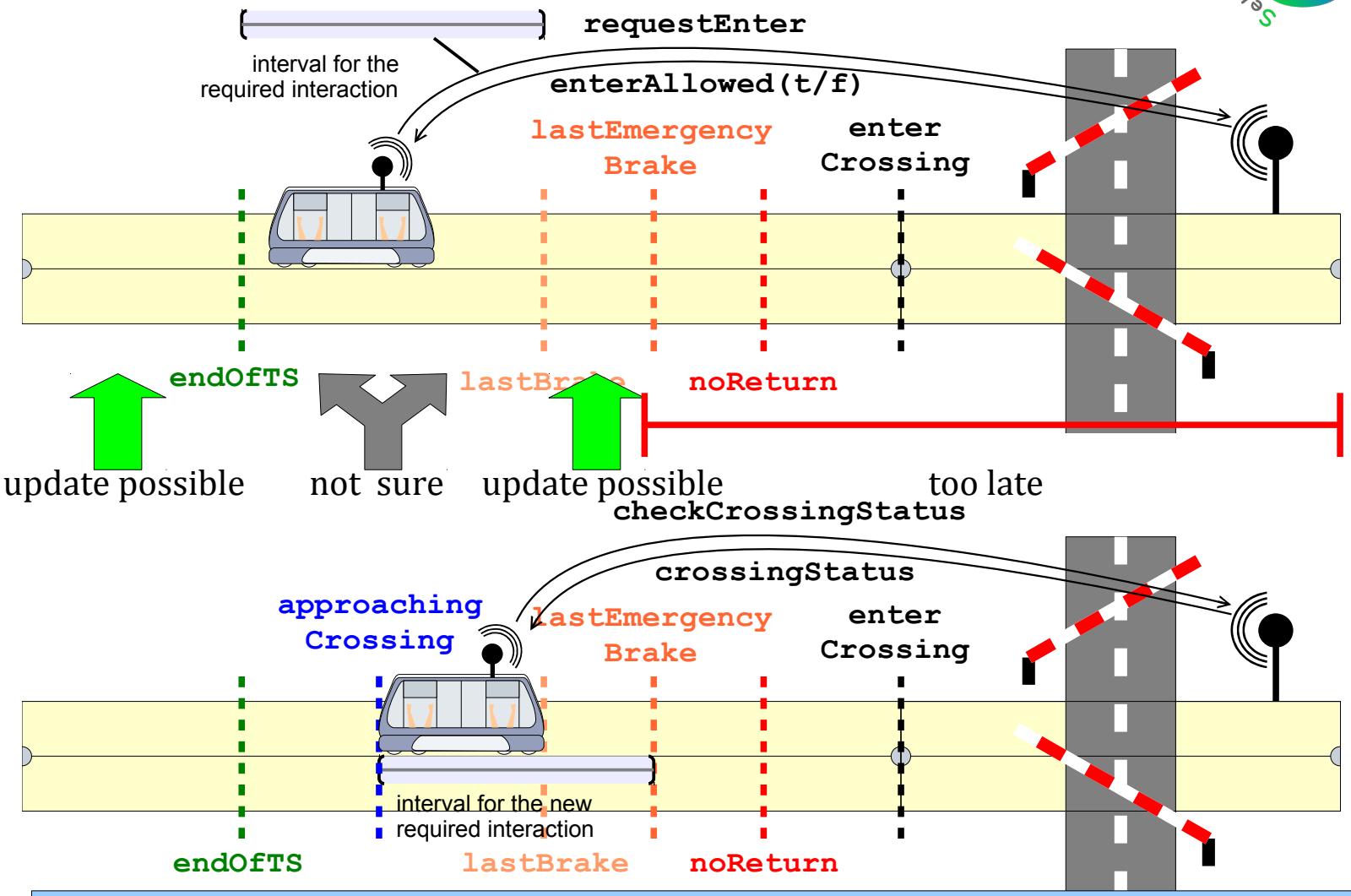
When is a system updatable?

Fundamental criterion



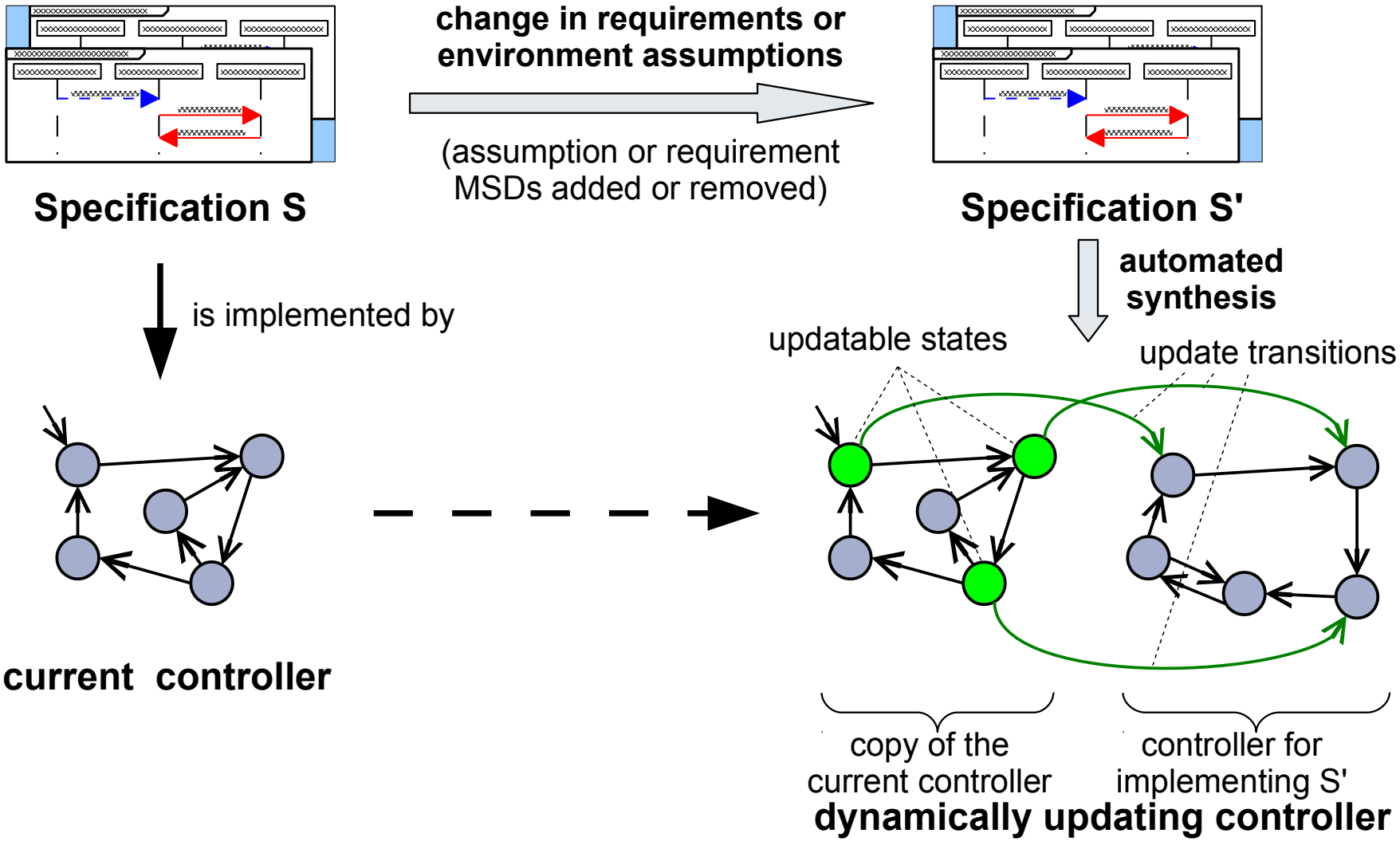
Intuition 1: The system must continue its past execution to satisfy S'.

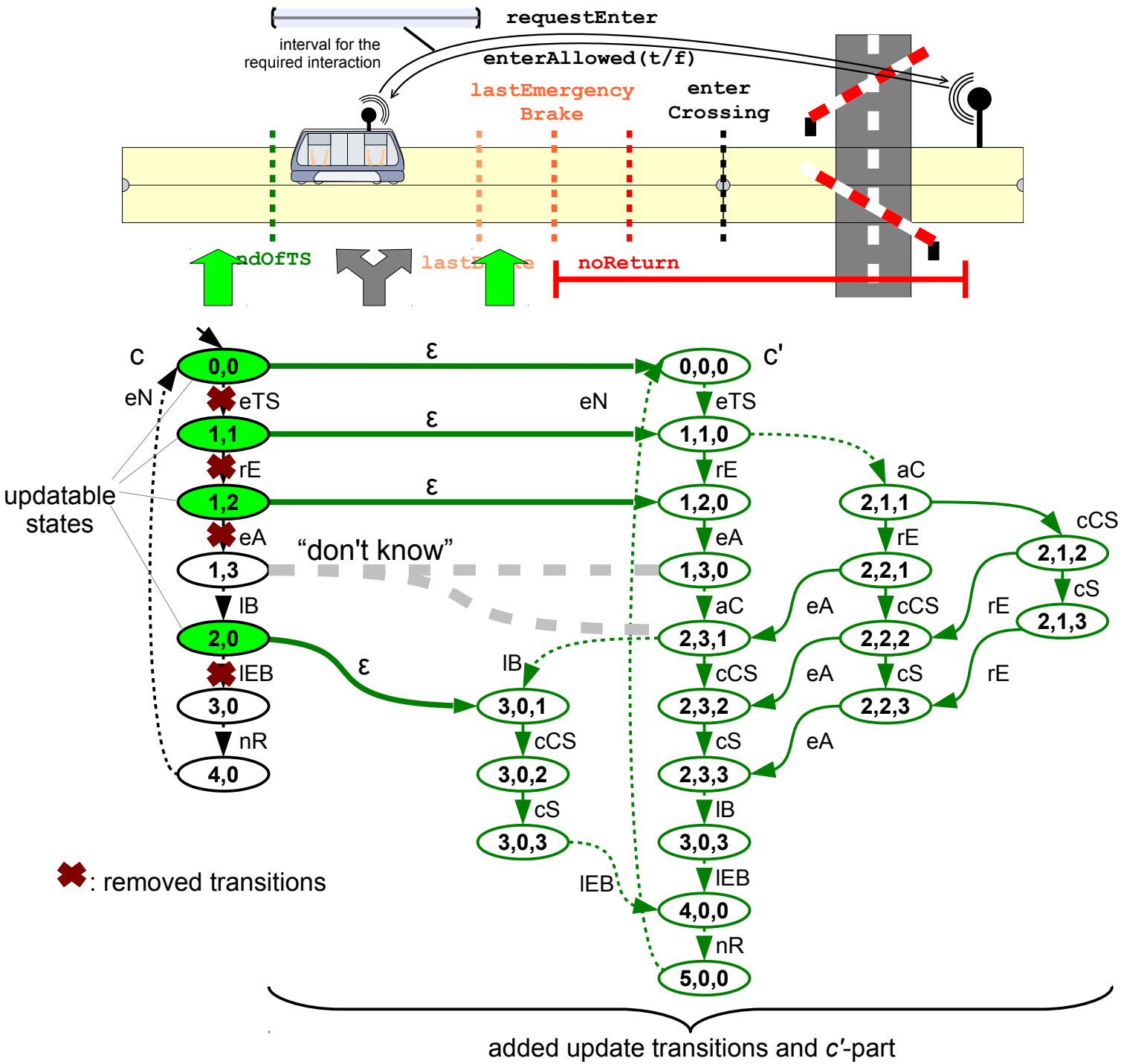
Fundamental criterion



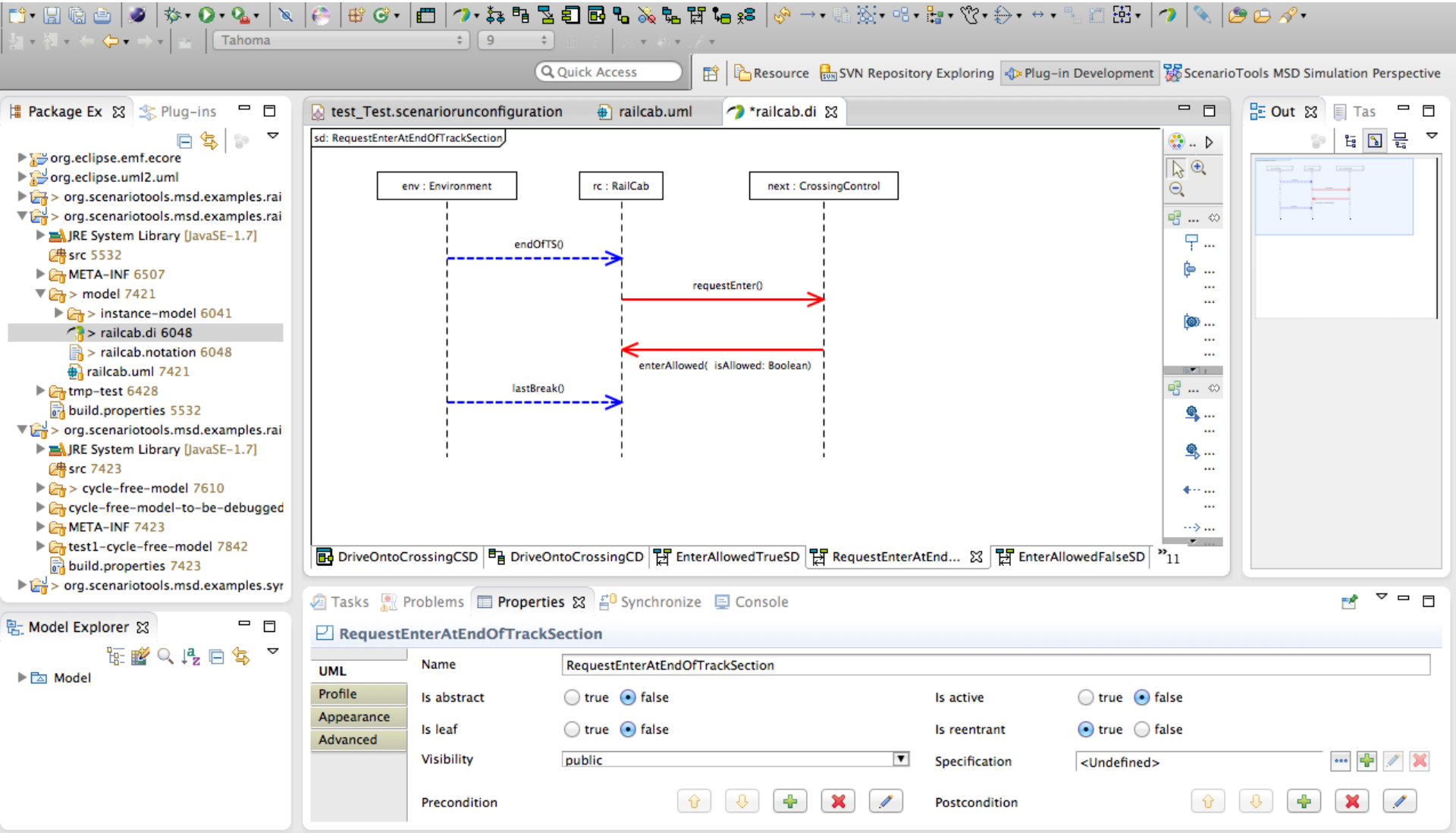
Intuition 2: The considered past execution starts from the last time the initial state is visited.

Synthesizing Dynamically Updating Controller

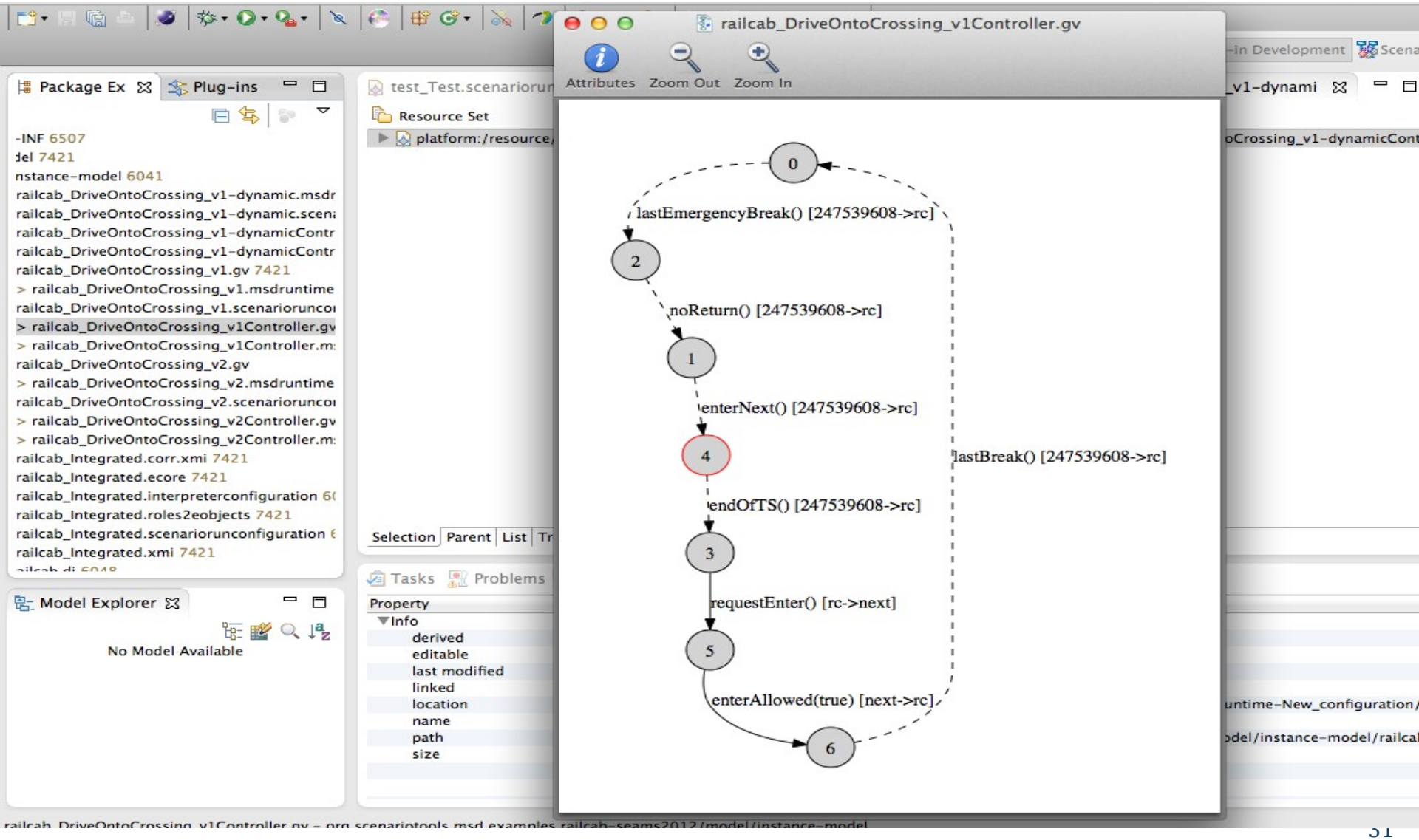



requestEnter
interval for the required interaction
enterAllowed(t/f)
lastEmergency Brake
enter Crossing
noReturn
c
c'
updatable states
"don't know"
0,0
1,1
1,2
1,3
2,0
3,0
4,0
0,0,0
1,1,0
1,2,0
1,3,0
2,3,1
2,3,2
2,3,3
3,0,3
4,0,0
5,0,0
2,1,1
2,1,2
2,1,3
2,2,1
2,2,2
2,2,3
3,0,1
3,0,2
ε
eN
eTS
rE
eA
IB
IEB
nR
aC
cCS
cS
: removed transitions
added update transitions and c'-part

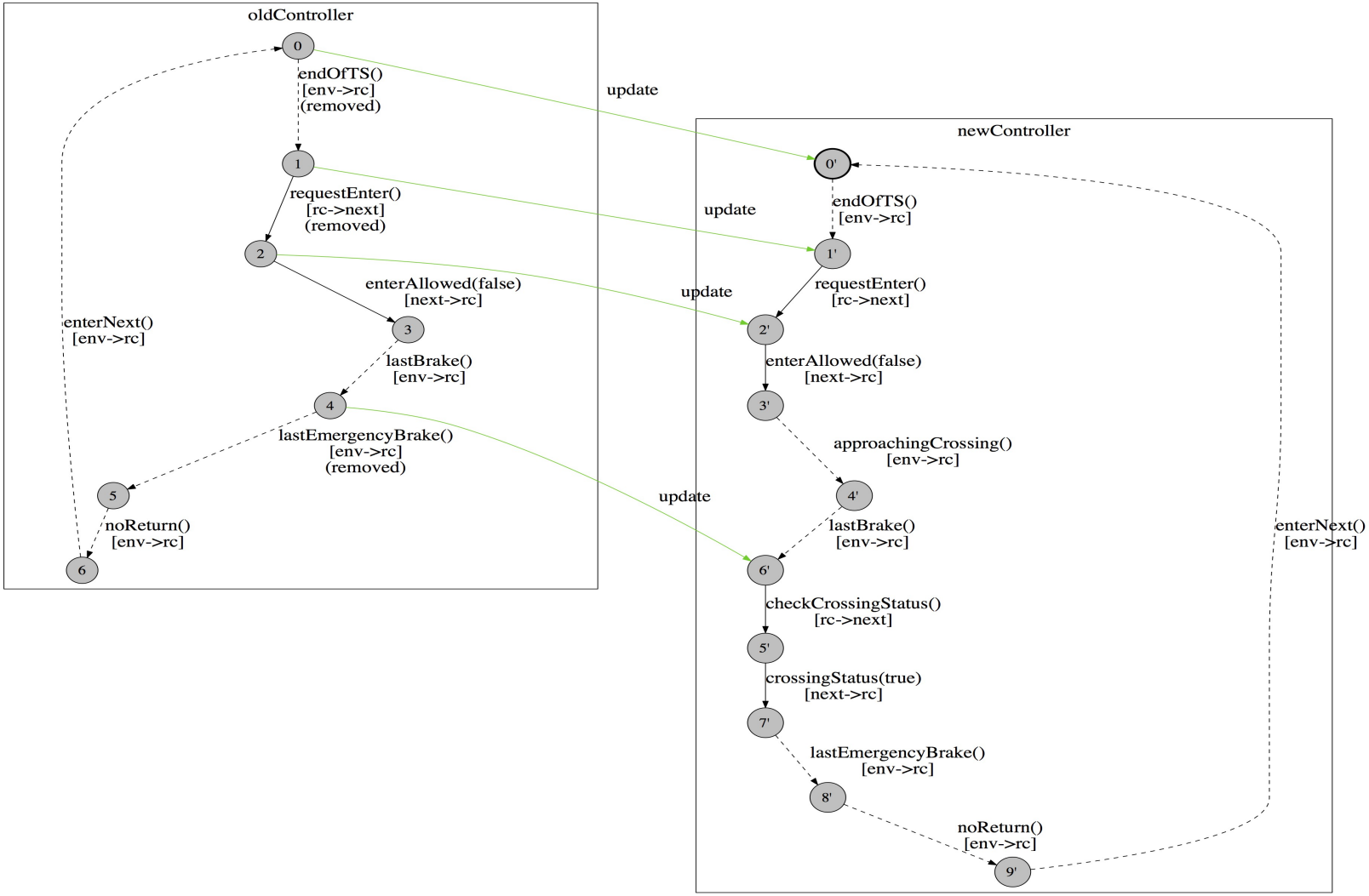
Modeling MSD Specifications



Controller Synthesis



Dynamically Updating Controller



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

deep se
dependable evolvable pervasive software engineering group

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?

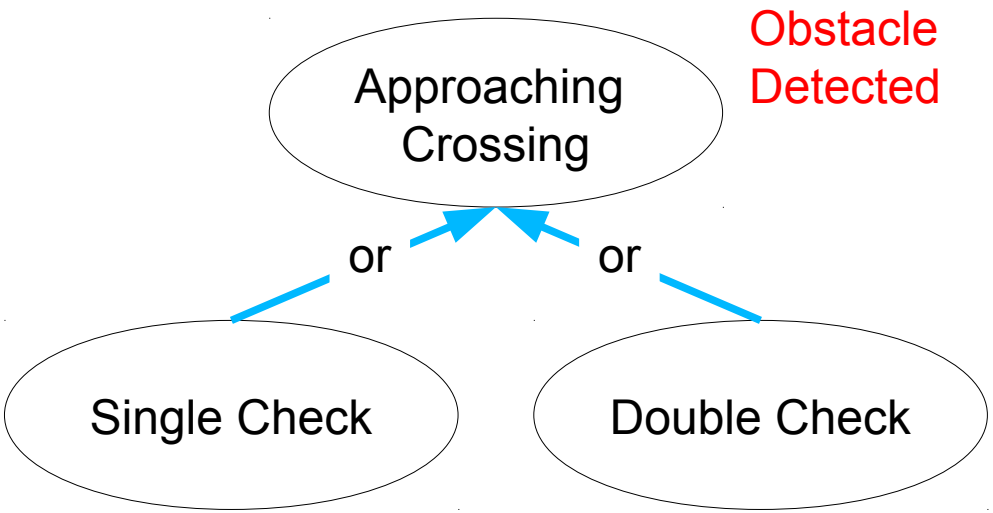
- Our criterion of updatable states can help:
  - **Automatic identification** of safe updatable states
  - **More timely adaptation**
    - No need to wait for quiescence

2) Automatic generation of self-adaptive systems

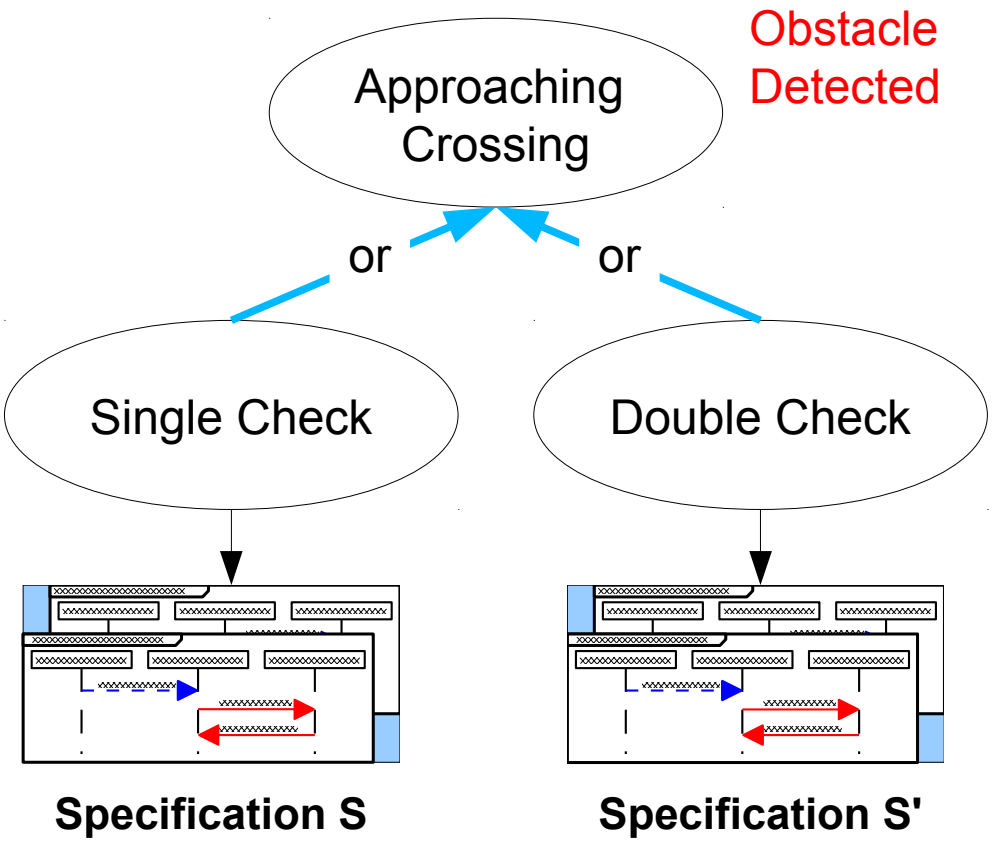
Integrating goal models and scenario-based specifications:

- 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

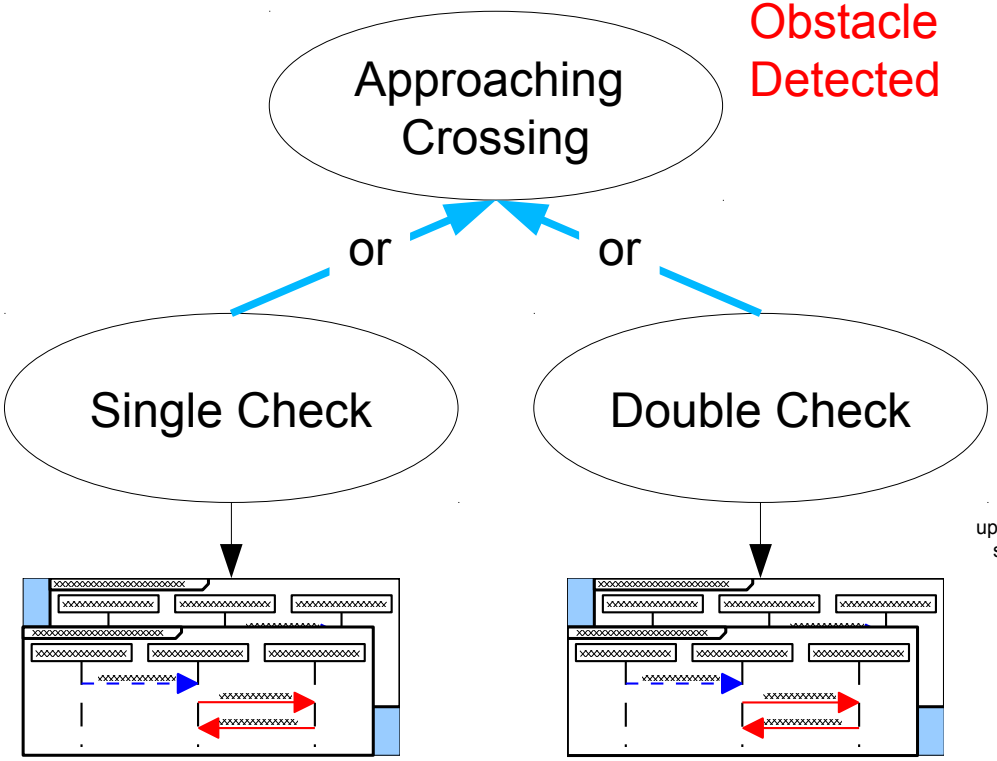
RailCab Example



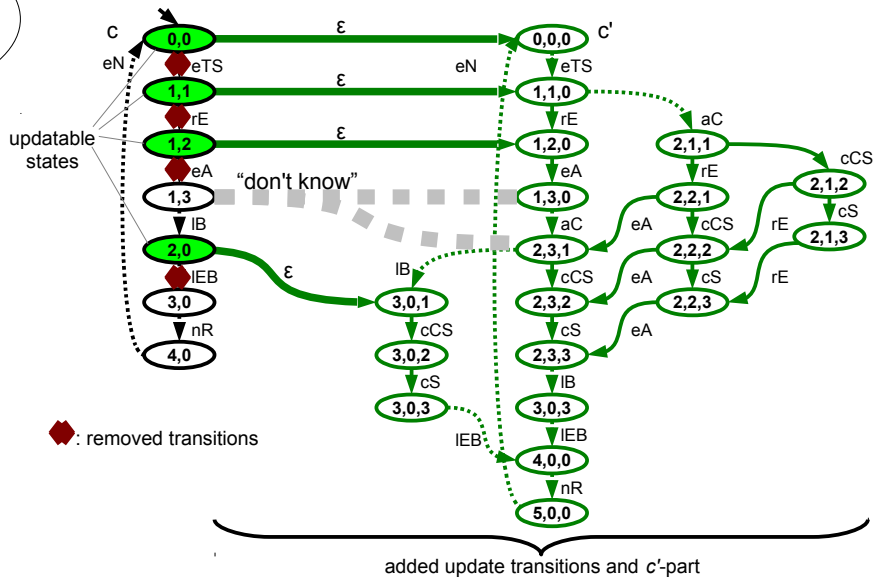
RailCab Example



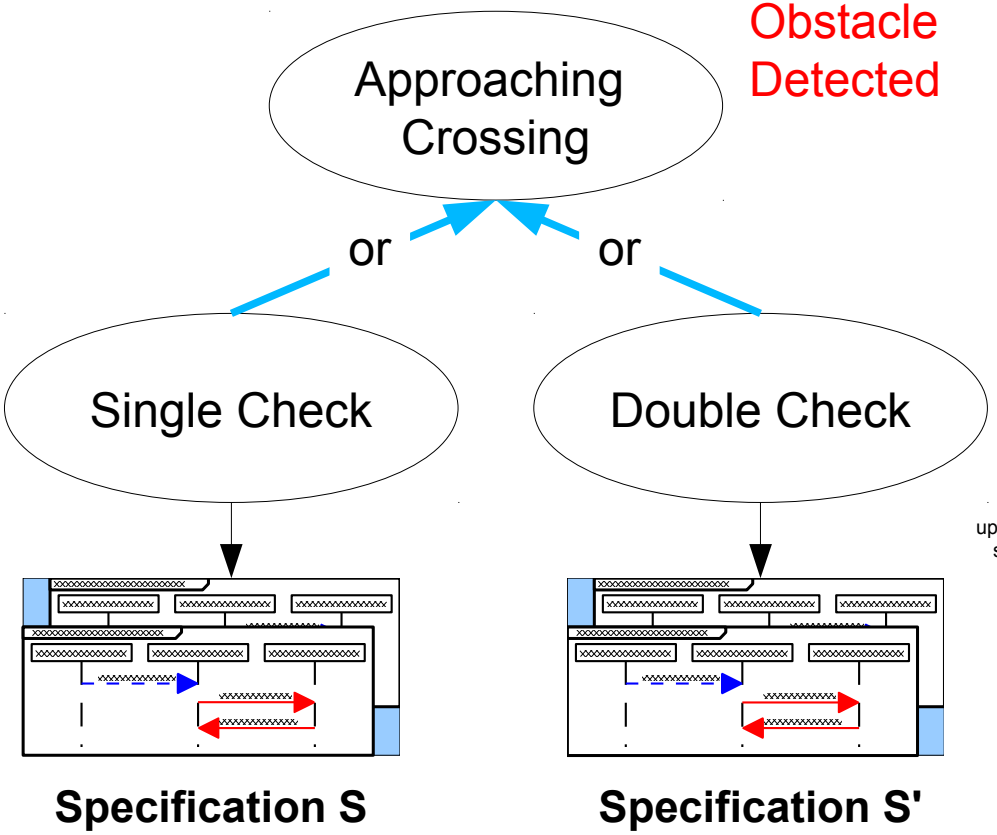
RailCab Example

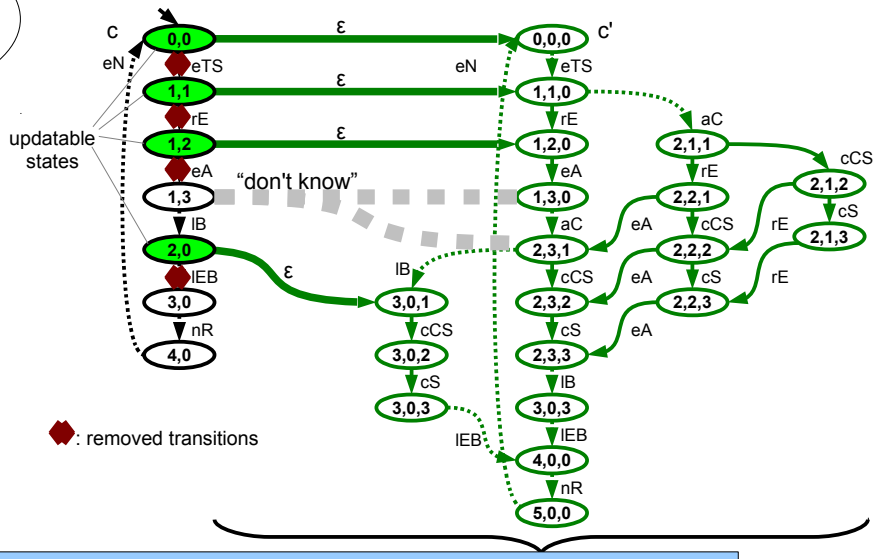


Specification S **Specification S'**



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

deep se
dependable evolvable pervasive software engineering group