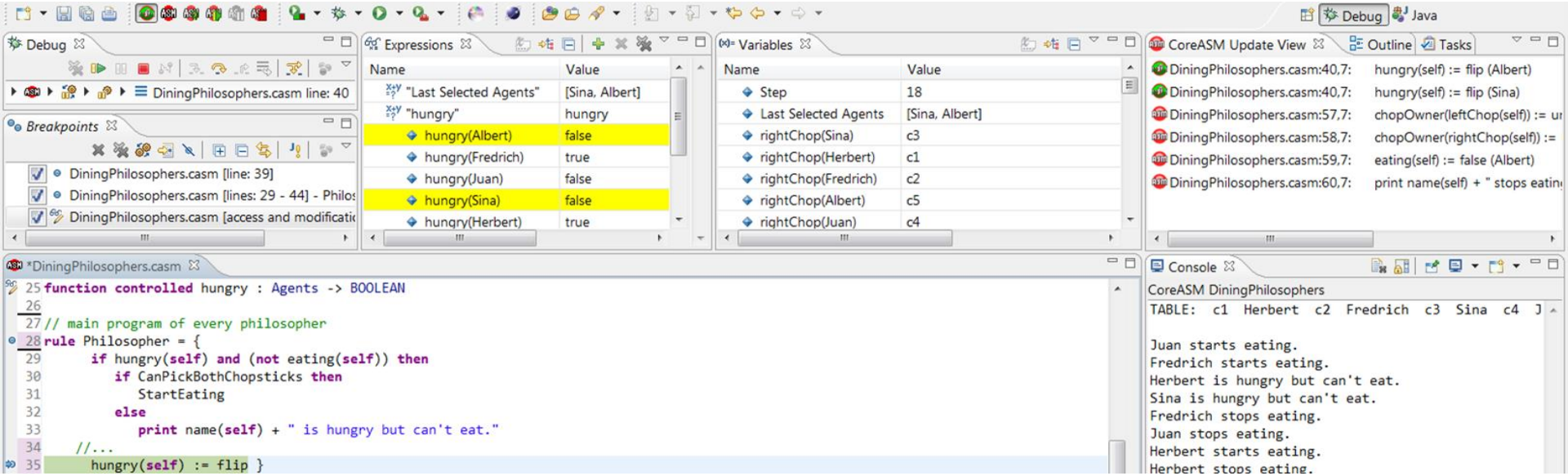
ulm university universität uulm




Alexander Raschke (Ulm), Egon Börger (Pisa)

# Short Introduction to CoreASM

## General Information

- Developed by Farahbod Roozbeh et al. from 2005 to 2010 as PhD-thesis at Simon Fraser University, Vancouver, Canada.

- Open-source tool (Java 1.7)
- Source code (and documentation) available at https://github.com/coreasm

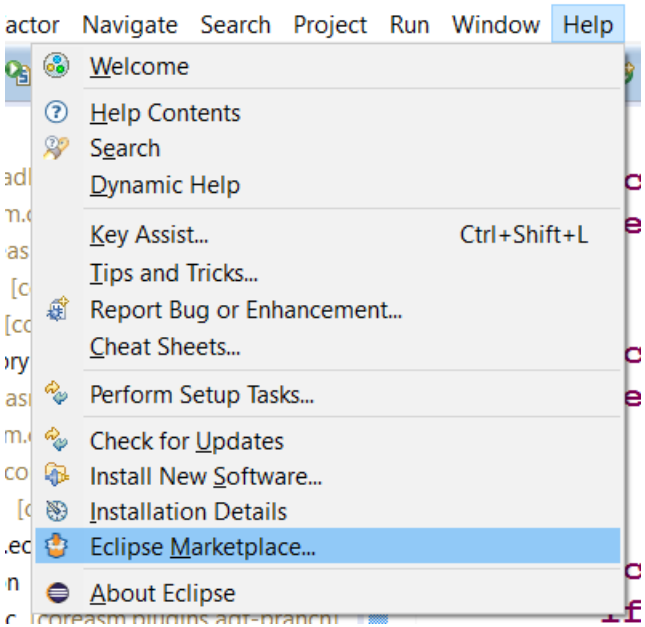
- General Idea:
  - minimal core and everything else is implemented as a plugin (even ConditionalRule (if-then-else) is a plugin!)
  - Easy to extend and very flexible possibilities
  - Set of plugins already shipped with installation (see user manual at https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.engine/rsc/doc/user_manual/CoreASM-UserManual.pdf)

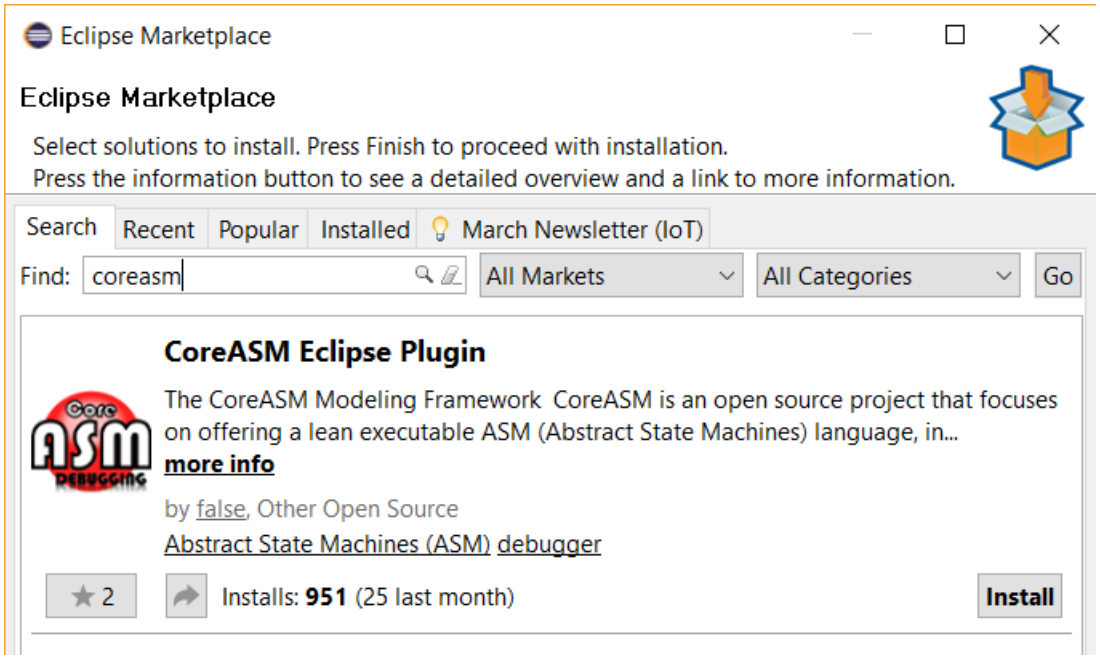
# CoreASM overview

| CoreASM | |
|---|---|
| **Carma**<br>command line interface | **eclipse plugin(s)**<br>- project integration<br>- editor<br>- debugger |
| **CoreASM engine**<br>- Parser, Interpreter, Abstract Storage, Schedulers<br><br>- Basic plugins: Block, Conditional, Forall, …<br>- Standard plugins: TurboASM, Signature (types), …<br>- Additional plugins: Time, Scheduling, Math, … | |

# Installation

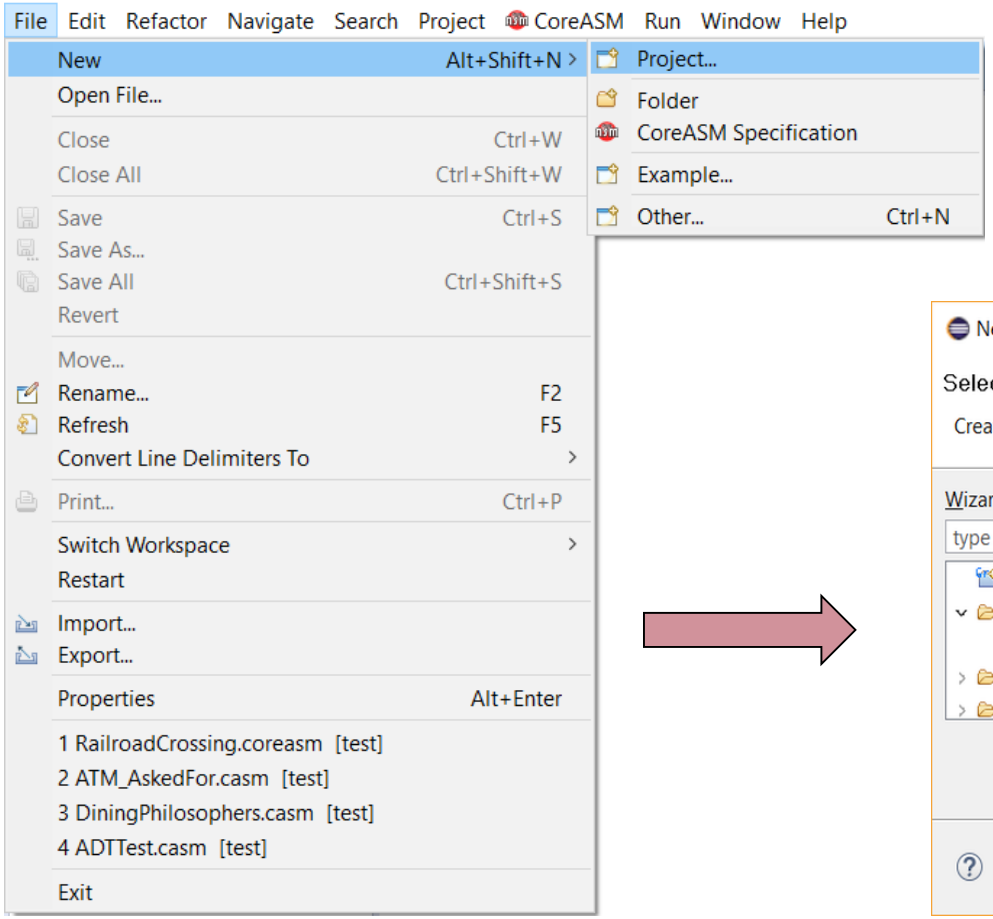
- Easiest way to get CoreASM is via eclipse marketplace: https://marketplace.eclipse.org/content/coreasm-eclipse-plugin

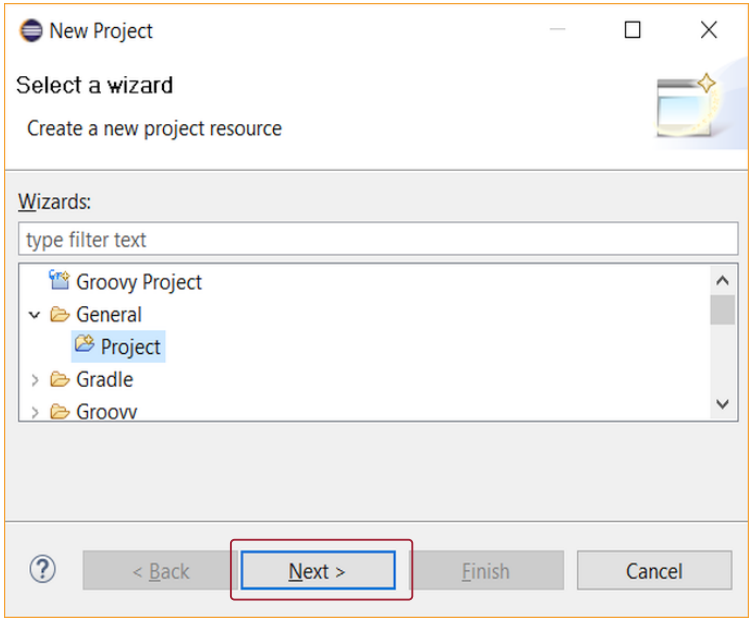


# Getting started

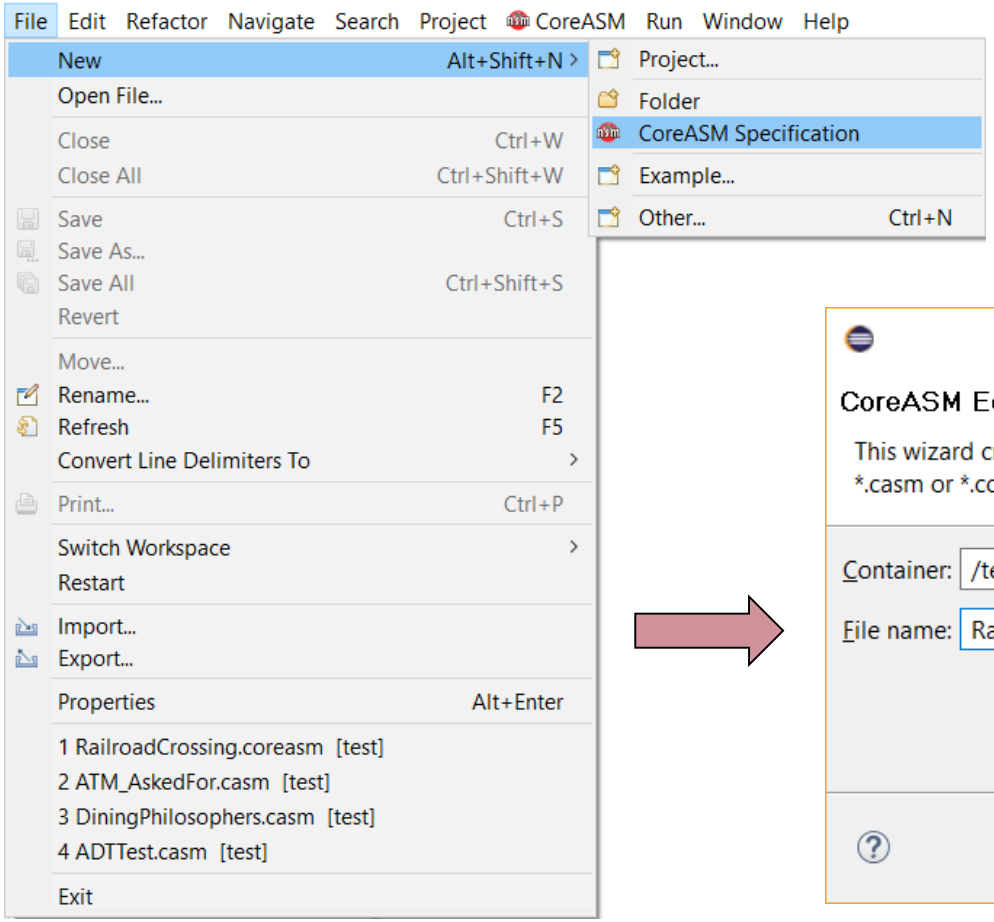
- First, you have to create a new (general) project in eclipse

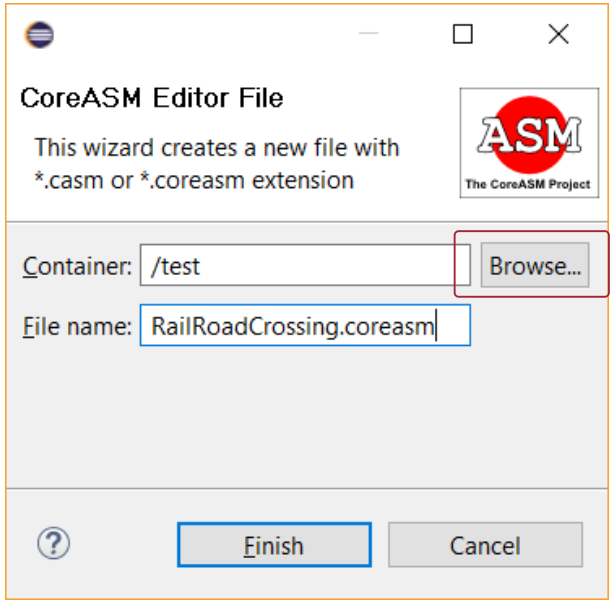


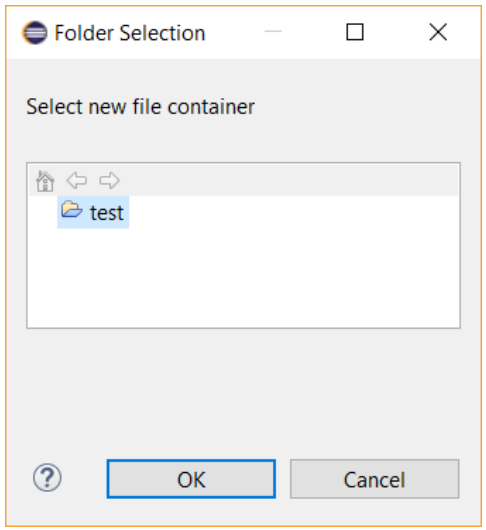



# Getting started

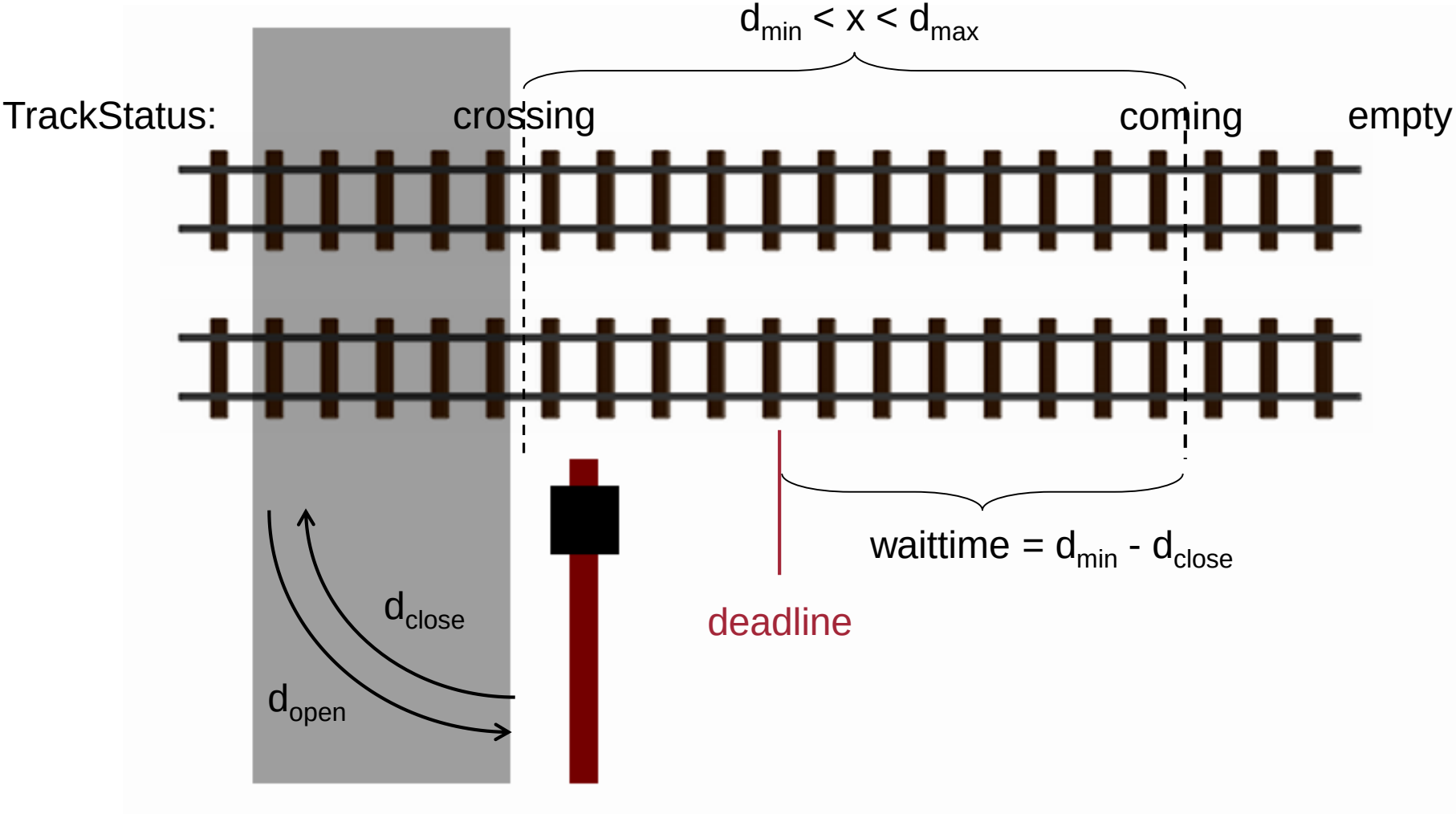
- In this new project container, you can create a CoreASM specification



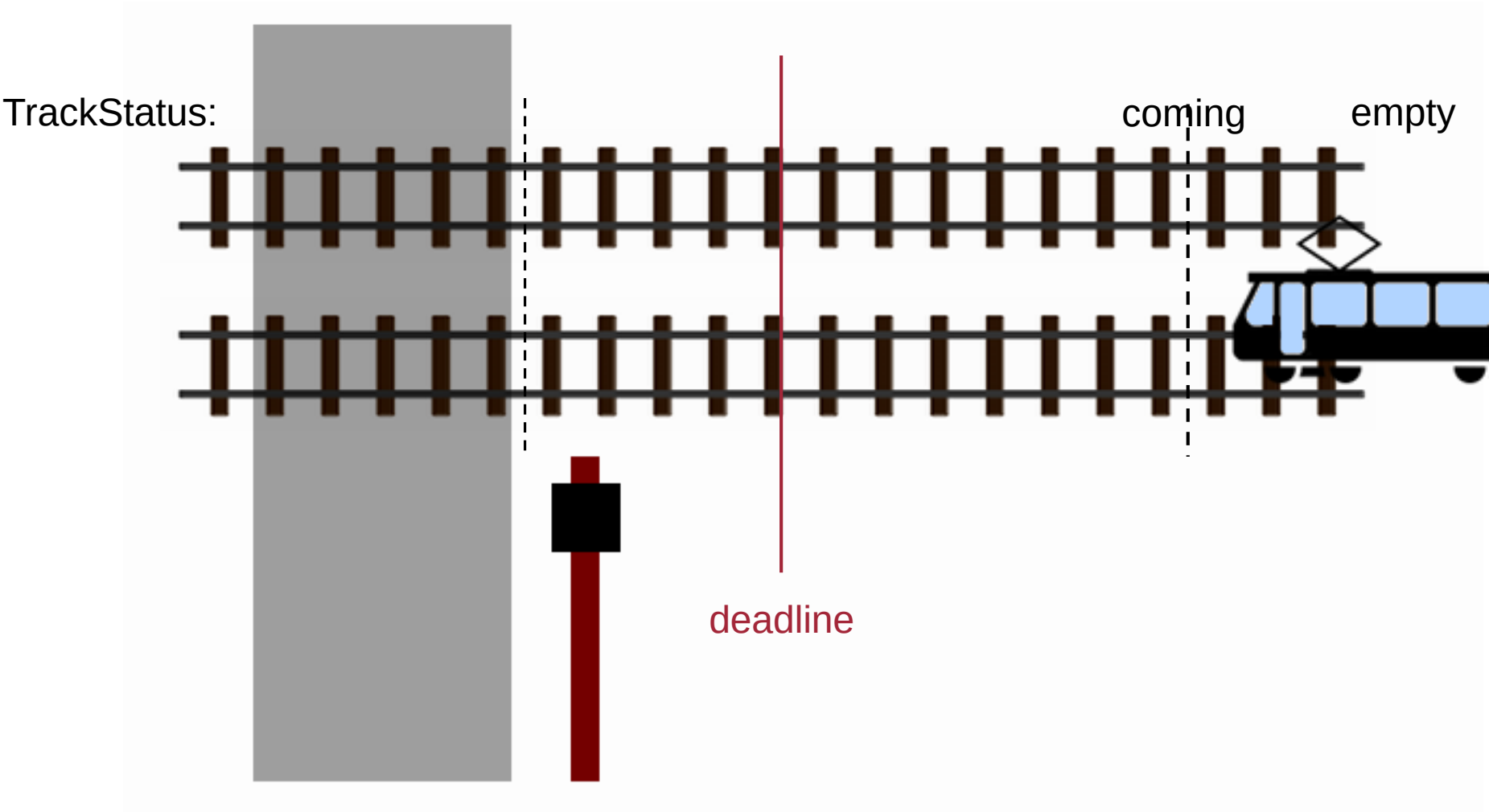



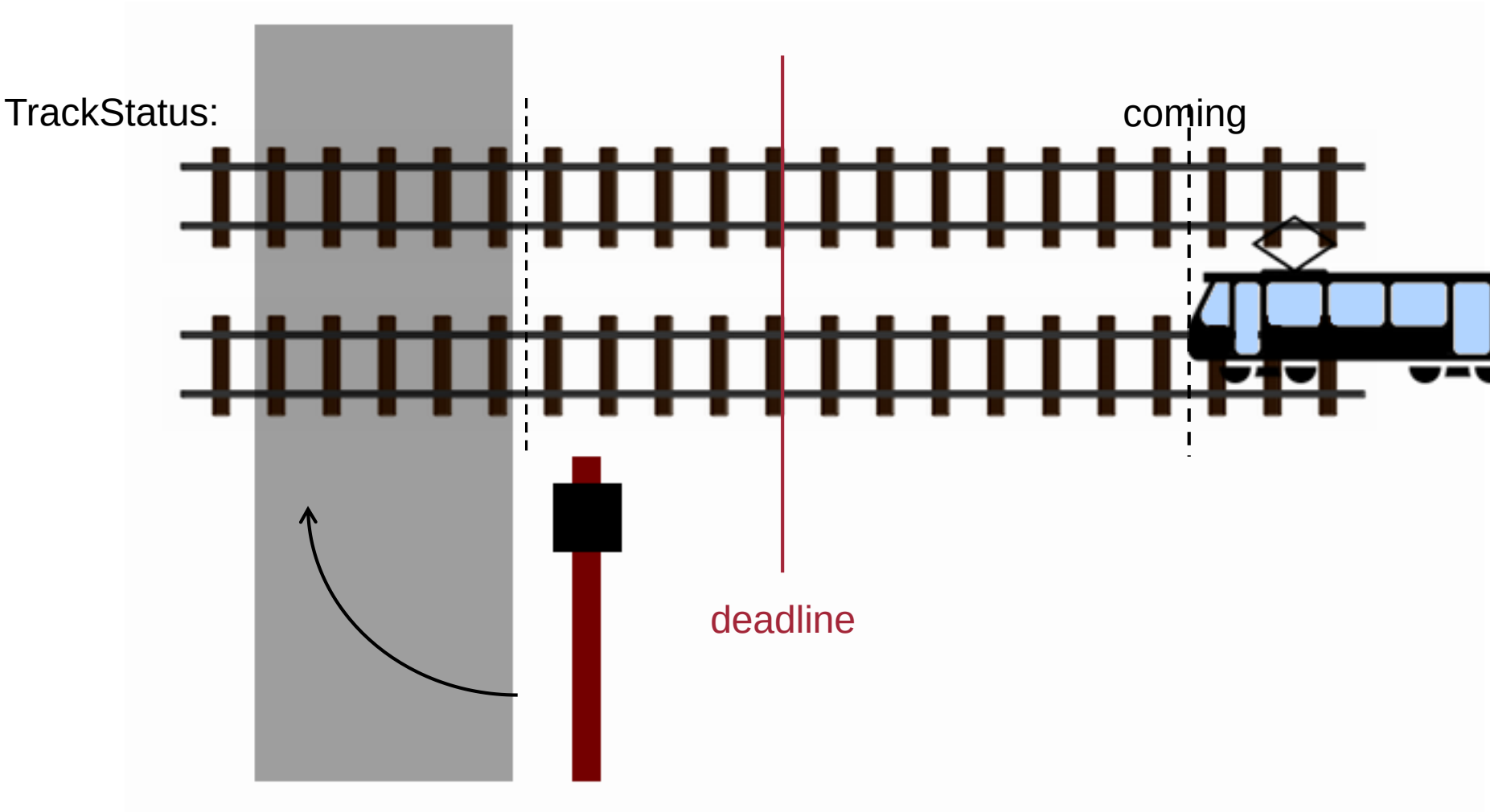
# First example: Railroad crossing



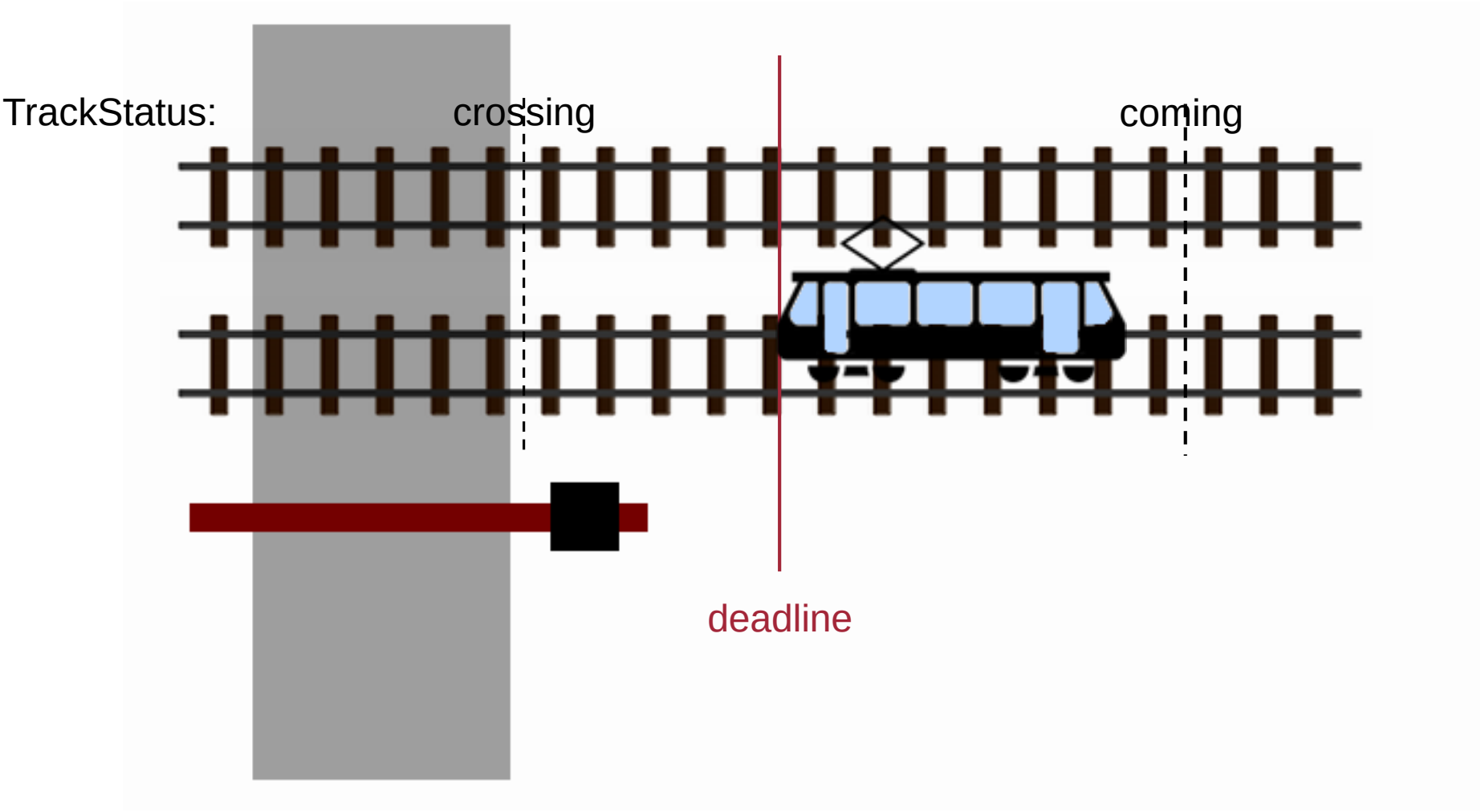
# First example: Railroad crossing



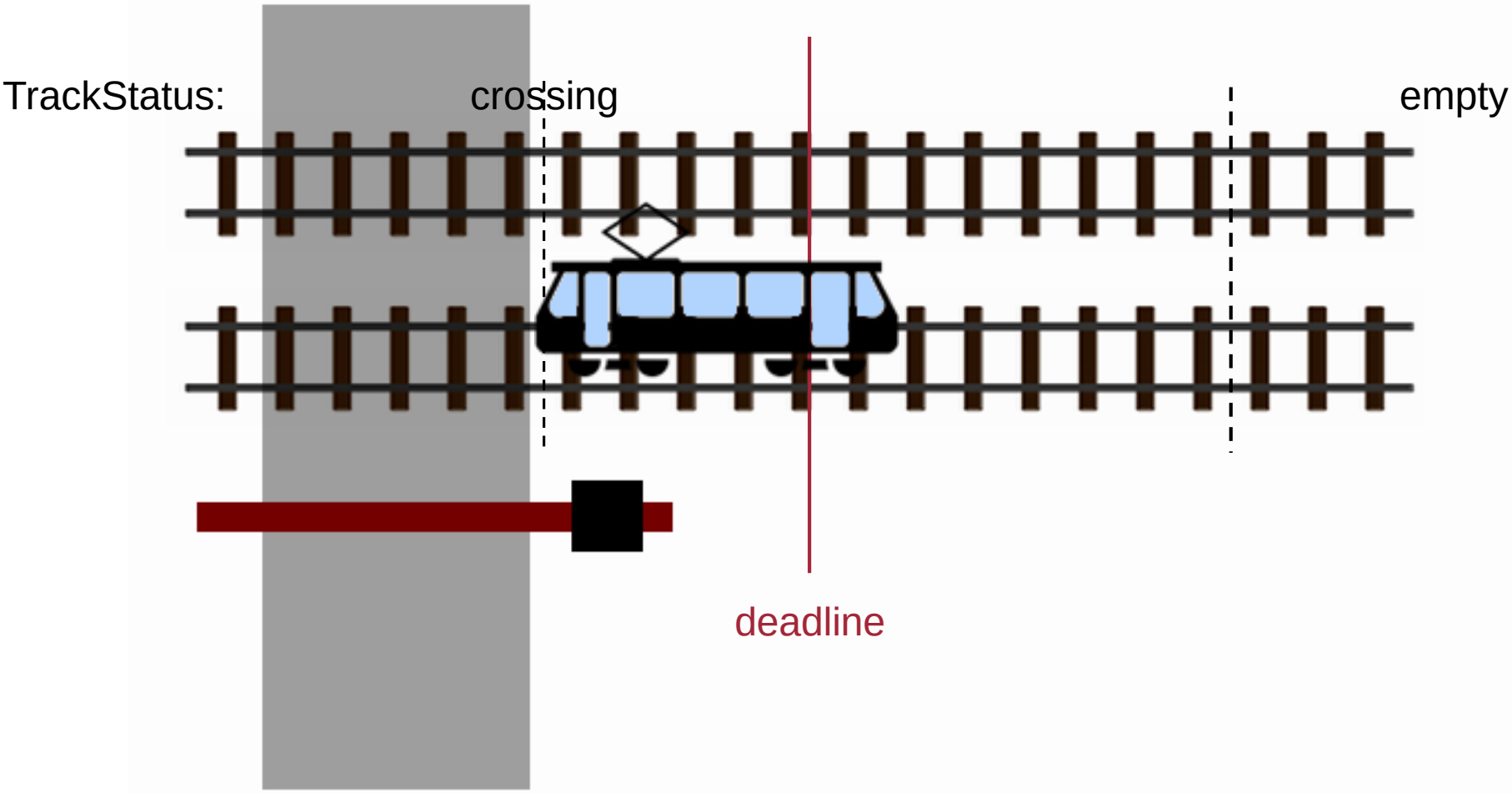
# First example: Railroad crossing



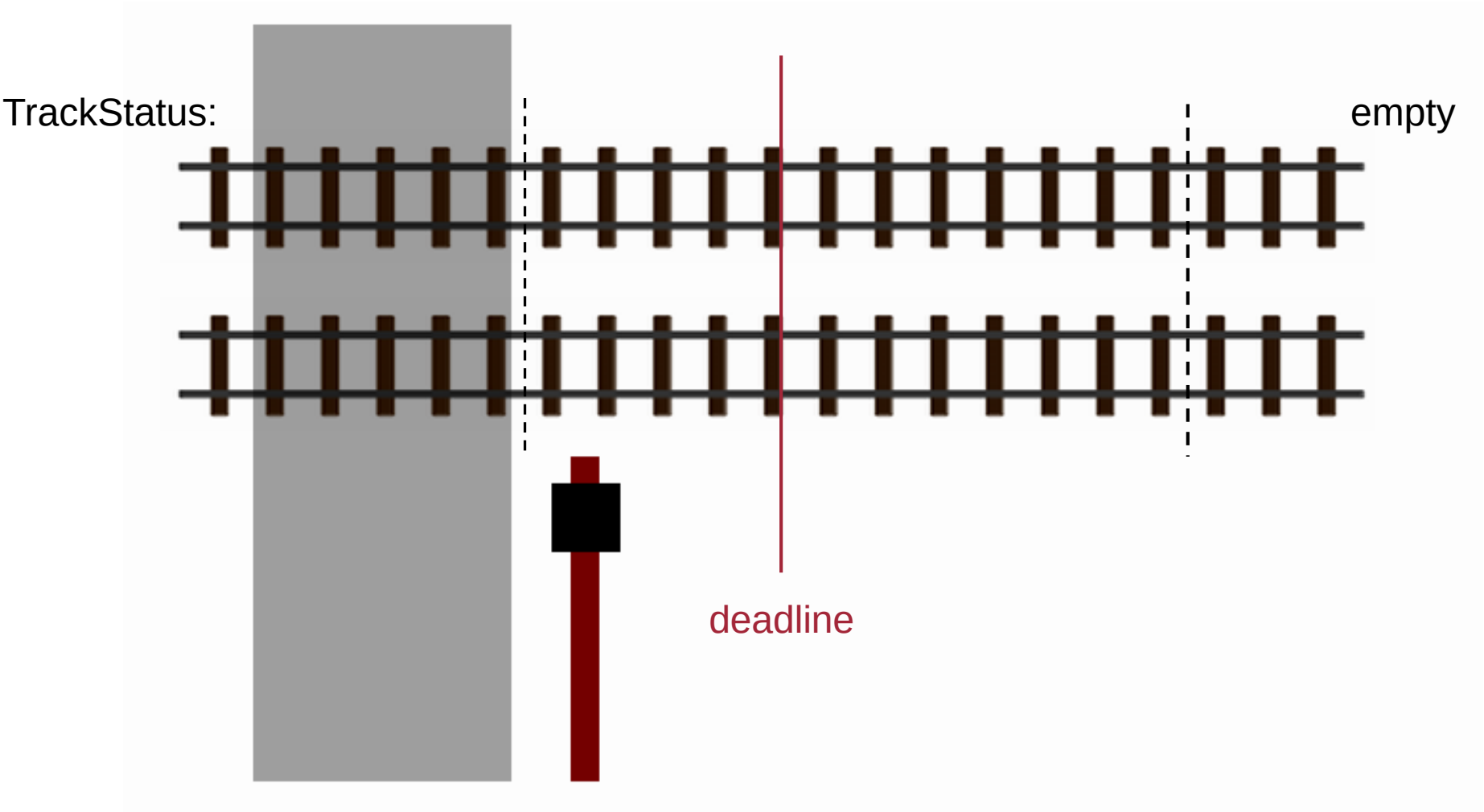
# First example: Railroad crossing



# First example: Railroad crossing



# First example: Railroad crossing



# From Mathematical Notation to Executable Specifications

$$
\begin{array}{l}
\textsc{TrackCtl} = \\
\quad \textbf{forall } x \in TRACK \\
\qquad SetDeadline(x) \\
\qquad SignalClose(x) \\
\qquad ClearDeadline(x) \\
\quad SignalOpen \\
\textbf{where} \\
\quad SetDeadline(x) = \textbf{if } TrackStatus(x) = coming \textbf{ and} \\
\qquad Deadline(x) = \infty \textbf{ then } Deadline(x) := currtime + WaitTime \\
\quad SignalClose(x) = \textbf{if } currtime = Deadline(x) \textbf{ then } Dir := close \\
\quad ClearDeadline(x) = \textbf{if } TrackStatus(x) = empty \textbf{ and} \\
\qquad Deadline(x) < \infty \textbf{ then } Deadline(x) := \infty \\
\quad SignalOpen = \textbf{if } Dir = close \textbf{ and } SafeToOpen \textbf{ then } Dir := open \\
\quad SafeToOpen = \forall x \in TRACK \\
\qquad TrackStatus(x) = empty \textbf{ or } currtime + d_{open} < Deadline(x)
\end{array}
$$

$$\textsc{GateCtl} = \textsc{Switch}((Dir = open, opened), (Dir = close, closed))$$

E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003

## From Mathematical Notation to Executable Specifications

In the signature we find, besides the monitored system time *currtime*, a monitored function $TrackStatus\colon TRACK \to \{empty, coming, inCrossing\}$ and a controlled function $Deadline\colon TRACK \to REAL \cup \{\infty\}$ to measure the allowable $WaitTime = d_{min} - d_{close}$ between the appearance of a train and the latest possible moment to start the gate closing (for the gate to be closed in time). A function $Dir \in \{open, close\}$ controlled by the track control signals when to open or close the gate. The actual gate status *opened* or *closed* is the value of the gate control state, which therefore is called *GateStatus*.

The sync RailCrossCtl ASM consists of the basic ASMs TrackCtl and GateCtl controlling the tracks and the gate in the presence of the environment which sets the monitored function *TrackStatus*. For each track the deadline is set upon arrival of a train, the signal to close is sent to the gate control upon deadline expiration, and the deadline is cleared when the track becomes empty. The signal to open is sent to the gate only when it is safe to do so. GateCtl is an instance of the FlipFlop on p. 47. We refer to its two control state transitions as *OpenGate* and *CloseGate*.

E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003

## First example in CoreASM

```
CoreASM RailRoadCrossing

use StandardPlugins
use TimePlugin
use MathPlugin
```

Every specification must start with this keyword

This name is arbitrary; it is recommended to choose a useful name

StandardPlugins is a collection of several plugins; it is recommended to use it in every specification

MathPlugin is used to calculate with NUMBERs

TimePlugin provides a monitored function "now" representing the current system time (incremented after every step)

rules are introduced with this keyword

don't miss the "**par**"/"**endpar**" keywords! Intendation is not recognized!

```
rule TrackControl =
  par
    forall x in Track do
    par
      SetDeadline(x)
      SignalClose(x)
      ClearDeadline(x)
    endpar
    SignalOpen
  endpar
```

many rules have a "**do**" at the end!

"element of" is translated to "**in**"

no semicolons at the end of lines

$$
\begin{aligned}
&\textsc{TrackCtl} = \\
&\quad \textbf{forall } x \in TRACK \\
&\qquad SetDeadline(x) \\
&\qquad SignalClose(x) \\
&\qquad ClearDeadline(x) \\
&\quad SignalOpen \\
&\textbf{where} \\
&\quad SetDeadline(x) = \textbf{if } TrackStatus
\end{aligned}
$$

"where" for local definitions is not supported, hence all definitions are global

$$SetDeadline(x) = \textbf{if } TrackStatus(x) = coming \textbf{ and } Deadline(x) = \infty \textbf{ then } Deadline(x) := currtime + WaitTime$$
$$SignalClose(x) = \textbf{if } currtime = Deadline(x) \textbf{ then } Dir := close$$
$$ClearDeadline(x) = \textbf{if } TrackStatus(x) = empty \textbf{ and } Deadline(x) < \infty \textbf{ then } Deadline(x) := \infty$$
$$SignalOpen = \textbf{if } Dir = close \textbf{ and } SafeToOpen \textbf{ then } Dir := open$$
$$SafeToOpen = \forall x \in TRACK \; TrackStatus(x) = empty \textbf{ or } currtime + d_{open} < Deadline(x)$$

logical operators are "**and**", "**or**", "**not**"

```
rule SetDeadline(x) =
  if trackStatus(x) = coming and deadline(x) = infinity then
    deadline(x) := now + waitTime

rule SignalClose(x) =
  if now >= deadline(x) and now <= deadline(x) + span then
    dir := close

rule ClearDeadline(x) =
  if trackStatus(x) = empty and deadline(x) < infinity then
    deadline(x) := infinity

rule SignalOpen =
  if dir = close and safeToOpen then
    dir := open

derived safeToOpen =
  forall t in Track holds
     trackStatus(t) = empty or (now + dopen) < deadline(t)
```

defined in MathPlugin

defined in TimePlugin

UpdateRule is "**:=**"

comparison is "**=**"

"safeToOpen" is a **derived** function, not a rule

this **forall** is an expression and not a rule; that's why we used "**holds**" instead of "**do**"

$$\textsc{GateCtl} = \textsc{Switch}((Dir = open, opened), (Dir = close, closed))^{12}$$

The rule "Switch" is not predefined in CoreASM. Thus, we simulate its behaviour.

```
rule GateControl =
  par
    if dir = open and gateState = closed then gateState := opened
    if dir = close and gateState = opened then gateState := closed
  endpar
```

The conditional rule is "**if … then … else …**". If you skip the "else"-part, it is treated as if you write "… else skip".

## Important part: Initialization

- You cannot run the specification, yet, because you have not defined an init rule:

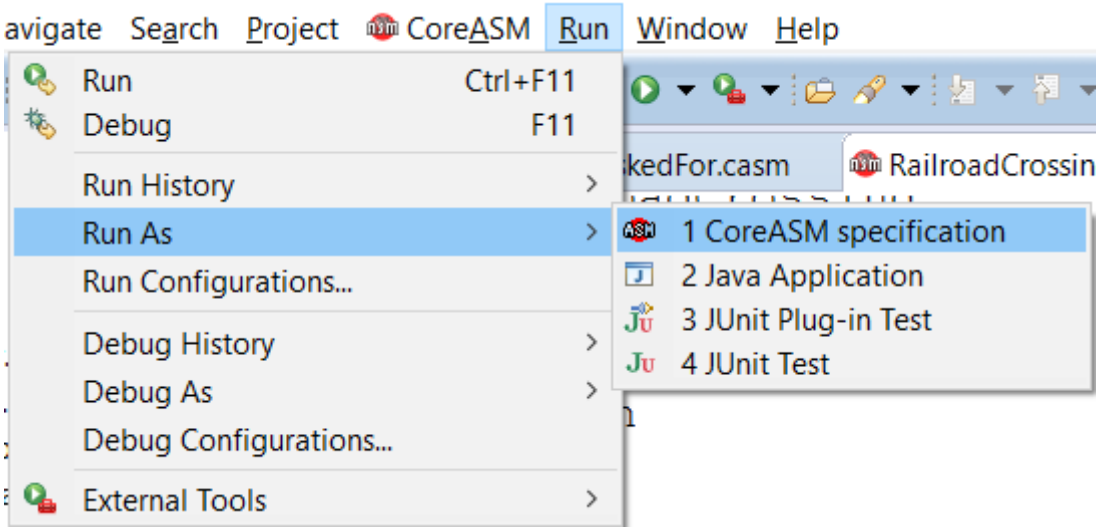
With the "**init**"-keyword you specify one "start"-rule that is executed at each step

```
init InitRule

rule InitRule = par
  TrackControl
  GateControl
endpar
```

You can write the "**par**"-keyword also at the end of a line

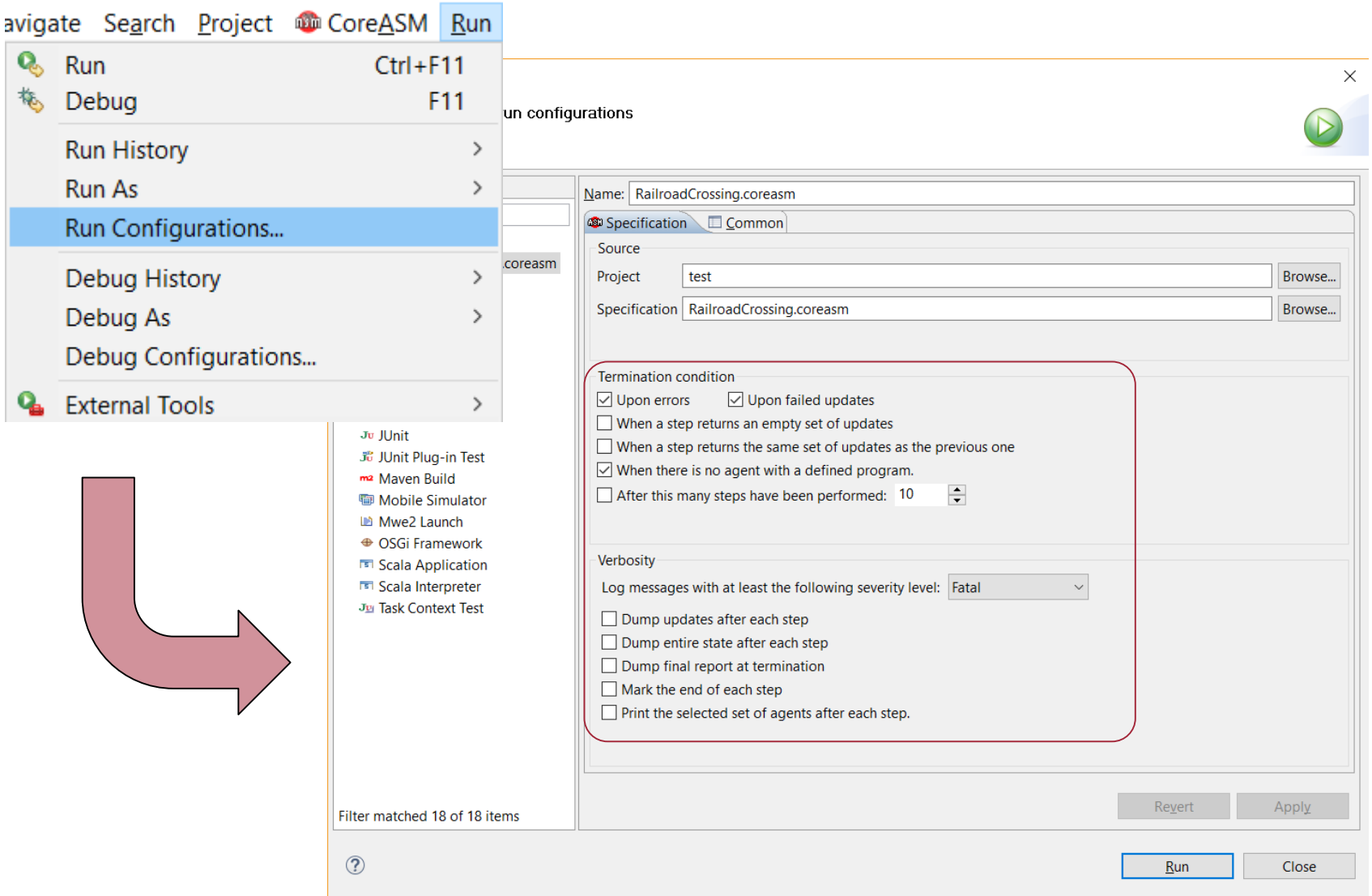
# Run the specification




**Error message:**

```
Cannot perform a 'forall' over undef. Forall domain must be an
enumerable element. (check C:\...\RailroadCrossing.coreasm:100,15)
    in TrackControl()
    called from InitRule()
```

# Change the "Run configuration"



# Change the "Run configuration"

Termination condition

- [x] Upon errors
- [x] Upon failed updates
- [ ] When a step returns an empty set of updates
- [ ] When a step returns the same set of updates as the previous one
- [x] When there is no agent with a defined program.
- [ ] After this many steps have been performed: 10

Verbosity

Log messages with at least the following severity level: Fatal

- [ ] Dump updates after each step
- [ ] Dump entire state after each step
- [ ] Dump final report at termination
- [ ] Mark the end of each step
- [ ] Print the selected set of agents after each step.

```
#--- end of step 1
```

## A closer look at the error message

```
Cannot perform a 'forall' over undef. Forall domain must be an
enumerable element. (check C:\...\RailroadCrossing.coreasm:100,15)
    in TrackControl()
    called from InitRule()
```

- So far, no types or controlled function definitions at all
- every identifier used in the specification is initialized with "**undef**"
- possible types in CoreASM (with plugins)
  - boolean
  - universe (a special built-in kind of set)
  - enumeration
  - number
  - string
  - collections (set, bag, list, queue, stack, map)

# Remove "undef"s by defining/initializing all identifiers

definition of universes as sets

```
universe Track = {track1, track2}

enum TrackStatus = {empty, coming, crossing}
enum GateState = {opened, closed}
enum Direction = {open, close}
```

enumerations do not define sets, but possible values of a location

line comments with "//"
block comments with "/*.. */"

```
rule InitRule = par
  // initialize all constants
  dmin := 5000
  dmax := 10000
  dopen := 2000
  dclose := 2000
  waitTime := 3000
  dcrossing := 3000
  span := 500
  gateState := opened
  startTime := now
  // initialize the track sensors
  forall t in Track do
  par
    trackStatus(t) := empty
    deadline(t) := infinity
    passingTime(t) := 0
  endpar
  TrackControl
  GateControl
endpar // of the InitRule
```

## Still errors due to parallelism

- Initialization is performed in parallel ➔ update with initial state not yet available when "TrackControl" or "GateControl" are executed.
- Typical pattern to ensure initialization executed only once and at the beginning:

predefined set of agents (+ init-rule)

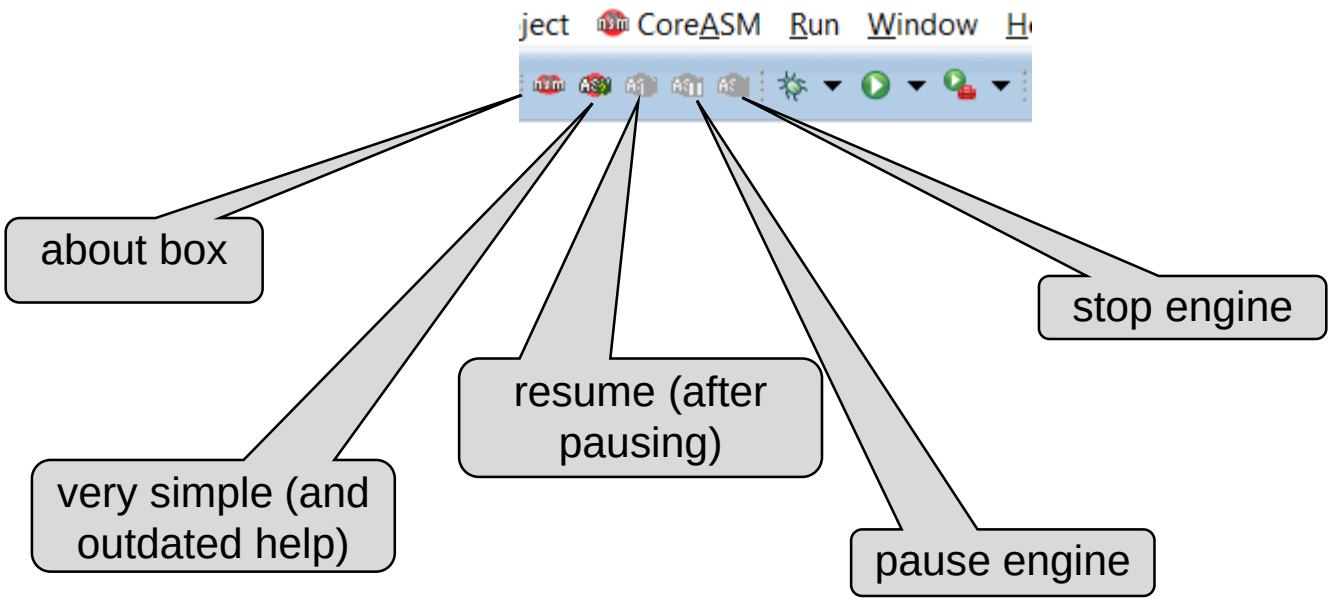
```
universe Agents = {trackController, gateController}
universe Track = {track1, track2}

rule InitRule = par
  // initialize all constants
  …
  // initialize the track sensors
  …
  program(trackController) := @TrackControl
  program(gateController) := @GateControl
  program(self) := undef
endpar
```

assignment of programs (= rules) to agents. **program** is a predefined function on the set of **Agents**. @<rulename> refers to the rule itself (not only the name)

The program of the initialization-agent is removed (set to undef). "**self**" is a predefined keyword to specify the currently executing agent

# Running, but… how to stop?



## Running, but…

- … no output is produced, except (if you made the changes in the run configuration):

```
#--- end of step 1
#--- end of step 2
#--- end of step 3
…
```

Don't forget to add "observer" to the set of Agents and to assign this rule as program for this agent!

- Solution: provide an observer

The TurboASM rule "**seqblock**" executes the rules inside sequentially.

```
rule ObserverProgram =
  seqblock
    print "Time: " + ((now - startTime) / 1000) + " seconds"
    forall t in Track do
      print "Track " + t + " is " + trackStatus(t)
    print "Gate is " + gateState
    print ""
  endseqblock
```

"**print**" uses Java "toString"-method to generate string for arbitrary types for console output, followed by a newline

## Running, but…

- … always the same output is produced
- Solution: provide an environment simulating the movement of a train:

```
rule EnvironmentProgram =
  choose t in Track do par
    if trackStatus(t) = empty then
      if random < 0.001 then par
        trackStatus(t) := coming
        passingTime(t) := now + dmin
    endpar
    if trackStatus(t) = coming then
      if passingTime(t) < now then par
        trackStatus(t) := crossing
        passingTime(t) := now + dcrossing
    endpar
    if trackStatus(t) = crossing then
      if passingTime(t) < now then
        trackStatus(t) := empty
    endpar
```

"**choose**" chooses an arbitrary element of the given set

"**random**" provides a number x with 0 <= x < 1

Again, don't forget to add "environment" to the set of Agents and to assign this rule as program for this agent!

## Best practices

```
option Signature.NoUndefinedId strict

function gateState: -> GateState initially opened
function trackStatus: Track -> TrackStatus
function deadline: Track -> NUMBER
function dir : -> Direction initially open
function passingTime: Track -> NUMBER
function startTime: -> NUMBER

derived dmin = 5000
derived dmax = 10000
derived dopen = 2000
derived dclose = 2000
derived waitTime = dmin - dclose
derived dcrossing = 3000
derived span = 500
```

don't allow undefined identifiers

define function types

define initial values (where possible) at the declaration

Define constants as derived functions

## From mathematical notation to executable specification

Roundup:

- Initialization
- definition of agents, disabling of init agent
- type definitions optional
- output with “print”
- modeling of environment

Some more features making the development easier:

- syntax highlighting
- code completion
- quick fixes
- debugging
- update set comparison

## Further documentation

- Please read the **User manual** to get familiar with the language: https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.engine/rsc/doc/user_manual/CoreASM-UserManual.pdf

- Please read the **Debugging manual** to get familiar with the debugging possibilities: https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.eclipse/rsc/doc/CoreASM_Eclipse_Debugger_Manual.pdf

- Please read the **Design documentation**, if you want to know details about how CoreASM works or if you want to contribute to this project: https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.engine/rsc/doc/CoreASM-DesignDocumentation.pdf

## Where to get it

eclipse marketplace (http://marketplace.eclipse.org/search/site/coreasm)

eclipse update site
(http://webcoreasm.informatik.uni-ulm.de/coreasm-repository/)

Sources on github
(https://github.com/coreasm)

If you have any question, don't hesitate to contact us via coreasm@uni-ulm.de

Michael Stegmaier

Alexander Raschke

## Copyright Notice