# Lustre and Scade: modelling and verifying reactive systems

# Reactive system

- In the course introduction, we have emphasised the "model-driven design" for ESs.
- In this lecture we will review Scade and Lustre: a tool and a modelling language that implement this approach

# Lustre - Scade

- Lustre is a synchronous language to specify reactive systems
- Scade is a tool to manipulate Lustre models

# Reactive system

- Computer system that reacts to the inputs it receives
- The sequence of inputs is not known by the system before its execution begins

# Reactive system

- Example: plane autopilot
  - The inputs are provided by the sensors (speed, attitude,...)
  - The system must keep the plane flying safely whatever the inputs
- To contrast: a compiler is not a reactive system
  - Input = File to compile (does not change during the execution)

# Synchronous language

- In a synchronous language, the following hypothesis is made on the system:
- It boasts a clock that ticks regularly
- In a synchronous model, it is assumed that all the computations are carried out simultaneously at each tick

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - d' = -d if x=0 ; d otherwise
    - x' = x + d

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - d' = -d if x=0 ; d otherwise
    - x' = x + d

x:

d:

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - $d' = -d$ if $x=0$ ; $d$ otherwise
    - $x' = x + d$

x: 0

d: -1

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - d' = -d if x=0 ; d otherwise
    - x' = x + d

| | | |
|---|---|---|
| x: | 0 | -1 |
| d: | -1 | 1 |

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - d’ = -d if x=0 ; d otherwise
    - x’ = x + d

| | | | |
|---|---|---|---|
| x: | 0 | -1 | 0 |
| d: | -1 | 1 | 1 |

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - d' = -d if x=0 ; d otherwise
    - x' = x + d

| | | | | |
|---|---|---|---|---|
| x: | 0 | -1 | 0 | 1 |
| d: | -1 | 1 | 1 | -1 |

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - d’ = -d if x=0 ; d otherwise
    - x’ = x + d

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| x: | 0 | -1 | 0 | 1 | 0 | -1 | 0 | 1 |
| d: | -1 | 1 | 1 | -1 | -1 | 1 | 1 | -1 |

# Synchronous language

- Example: Assume two variables d and x
  - At each tick they are updated:
    - d' = -d if x=0 ; d otherwise
    - x' = x + d

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| x: | 0 | -1 | 0 | 1 | 0 | -1 | 0 | 1 | |
| d: | -1 | 1 | 1 | -1 | -1 | 1 | 1 | -1 | ... |

# Synchronous language

- Variable updates are done in parallel. Each step consists in:
  - reading inputs: x and d
  - writing outputs: x' and d'

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| x: | 0 | -1 | 0 | 1 | 0 | -1 | 0 | 1 | |
| d: | -1 | 1 | 1 | -1 | -1 | 1 | 1 | -1 | ... |

# Historical background

- Lustre = research project started in the '80s in Grenoble.
  - Motivation: reliable development of critical systems:
    - aerospace
    - trains without drivers
    - nuclear power plants

# Historical background

- In the ‘90s, these ideas slowly transferred from the academic world to the industry:
  - Airbus A320 (fly-by-wire)
  - Framatom power plants 1450MW

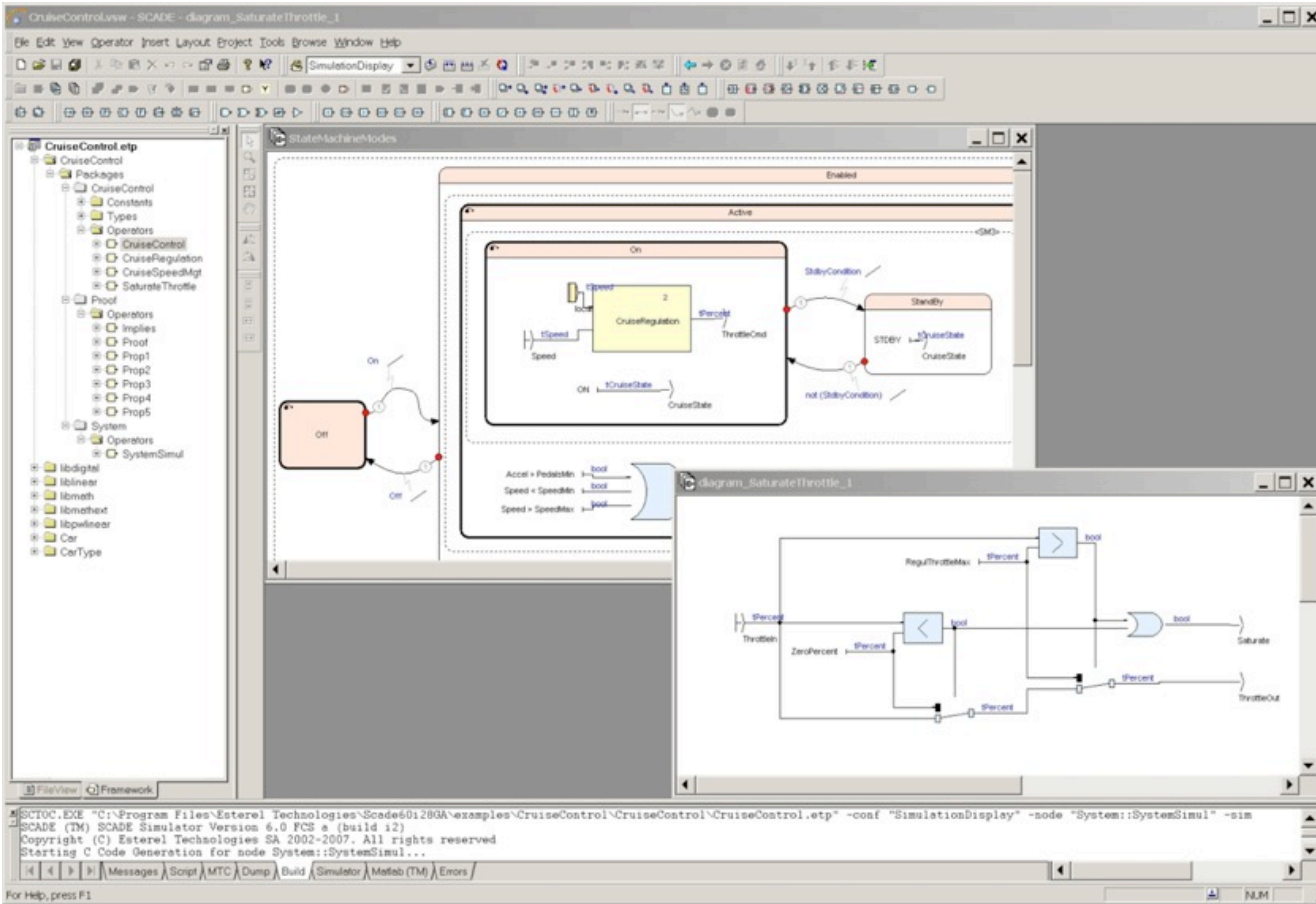
# Historical background

- In 1993, the Verimag laboratory is founded at Grenoble University by academics and industrial partners
- Verimag develops SCADE a commercial tool to model, test and verify synchronous systems using Lustre

# Historical background

- SCADE quickly replaces similar internal projects at Airbus, Merlin-Gerin, etc... and becomes a de facto European standard
- SCADE is afterwards sold to Esterel Technologies that develops and maintains it since then

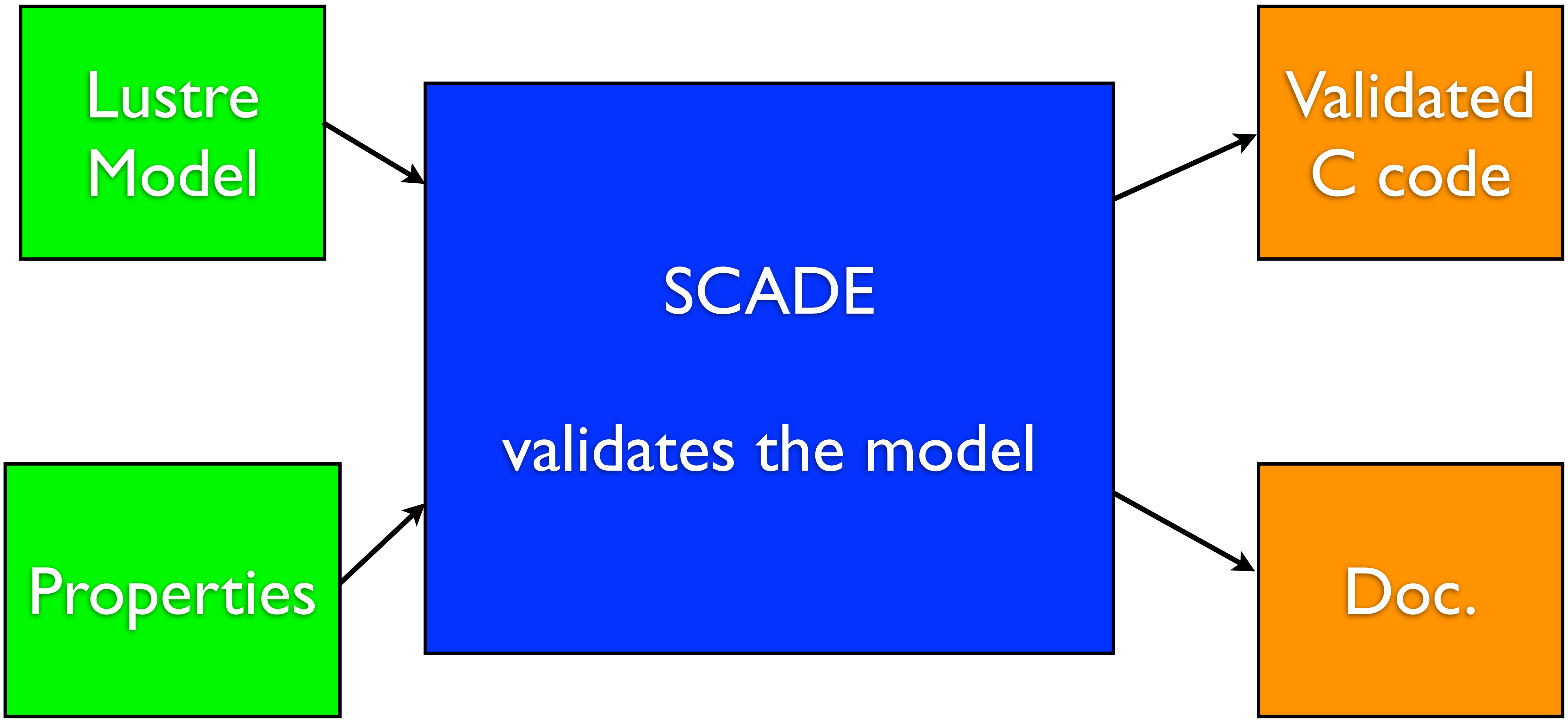
# Scade


CruiseControl.vsw - SCADE - diagram_SaturateThrottle_1
File Edit View Operator Insert Layout Project Tools Browse Window Help
SimulationDisplay
CruiseControl.etp
CruiseControl
Packages
Constants
Types
Operators
CruiseRegulation
CruiseSpeedMgt
SaturateThrottle
Proof
Implies
Prop1
Prop2
Prop3
Prop4
Prop5
System
SystemSimul
libdigital
liblinear
libmath
libmathext
libpwlinear
Car
CarType
FileView
Framework
StateMachineModes
Enabled
Active
On
Off
StandBy
CruiseRegulation
ThrottleCmd
Speed
CruiseState
StdbyCondition
not (StdbyCondition)
STDBY
Accel > PedalsMin
Speed < SpeedMin
Speed > SpeedMax
diagram_SaturateThrottle_1
RegulThrottleMax
ThrottleIn
ZeroPercent
Saturate
ThrottleOut
SCTOC.EXE "C:\Program Files\Esterel Technologies\Scade60i28GA\examples\CruiseControl\CruiseControl\CruiseControl.etp" -conf "SimulationDisplay" -node "System::SystemSimul" -sim
SCADE (TM) SCADE Simulator Version 6.0 FCS a (build i2)
Copyright (C) Esterel Technologies SA 2002-2007. All rights reserved
Starting C Code Generation for node System::SystemSimul...
Messages
Script
MTC
Dump
Build
Simulator
Matlab (TM)
Errors
For Help, press F1
NUM

# Scade

- SCADE is a complete software suite running under Windows. It contains:
  - a GUI to manipulate LUSTRE models graphically
  - a simulator
  - a verification tool
  - a code generator
  - a reporting tool

# Developing with scade



# Lustre - 3 basic concepts

- flow = data
- node = code (how to update flows)
- cycle = one step of execution (allows to precisely define the dynamics of the program)

# Lustre - flow

- Data are sequences of values called flows
  - All the values in a flow have the same type (it is the type of the flow)
  - Example: integer flow: 0,1,2,3,....

# Lustre - nodes

- Nodes update the flows and are the basic bricks of any Lustre program
- Lustre is a functional language: each node defines a function that receives input flows and produces output flows

# Lustre - nodes

- Example:

input flow

output flow

```
node EDGE (X:bool) returns (Y:bool) ;
```

constraint linking X and Y

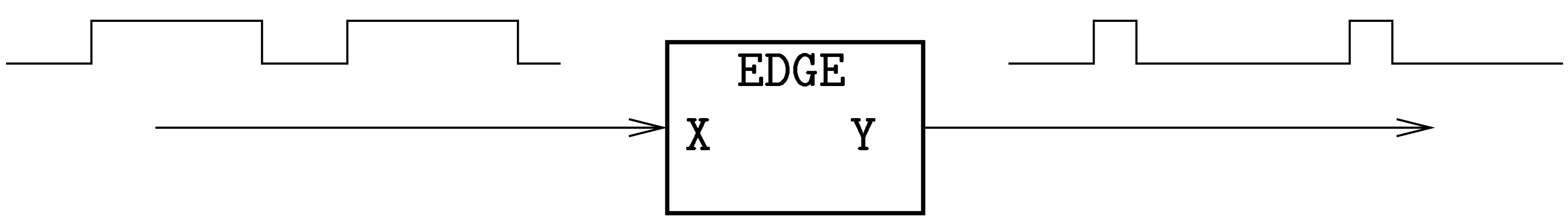
```
let
   Y = false -> X and not pre(X) ;
tel
```

# Lustre - cycles

- One cycle = one execution step of the Lustre program
- At each cycle, a new value is computed and added to each flow
  - Each cycle corresponds to a clock tick

# Lustre - cycles

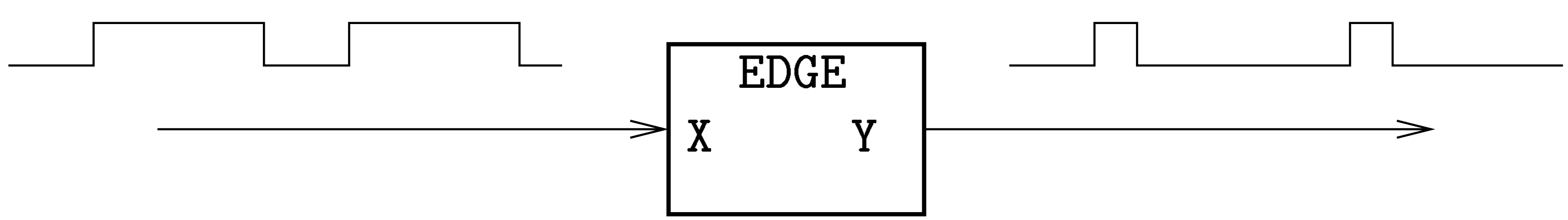
```
Y = false -> X and not pre(X)
```



# Lustre - cycles

init.
value

```
Y = false -> X and not pre(X)
```

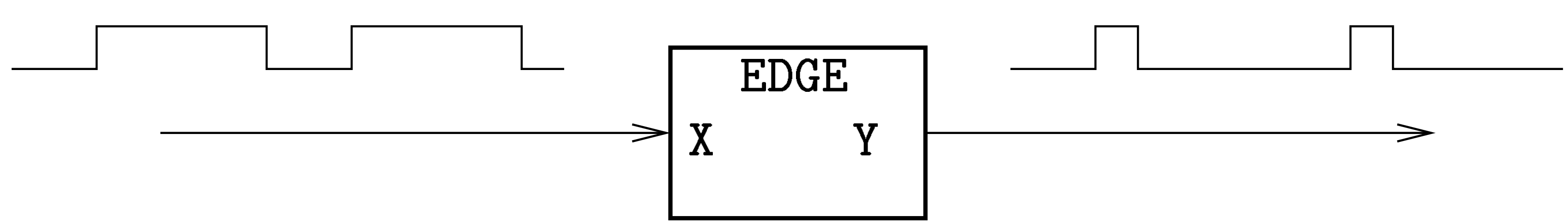


# Lustre - cycles

init.
value

previous
value of X

```
Y = false -> X and not pre(X)
```



# Lustre - cycles

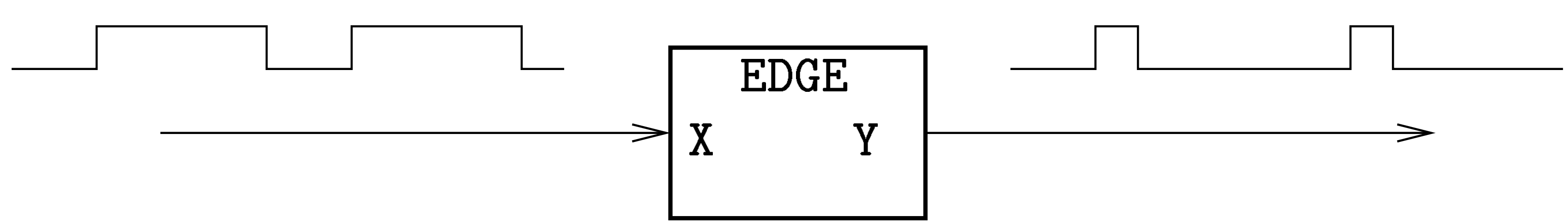
init. value

previous value of X

```
Y = false -> X and not pre(X)
```

Y

X F T T T F F T T



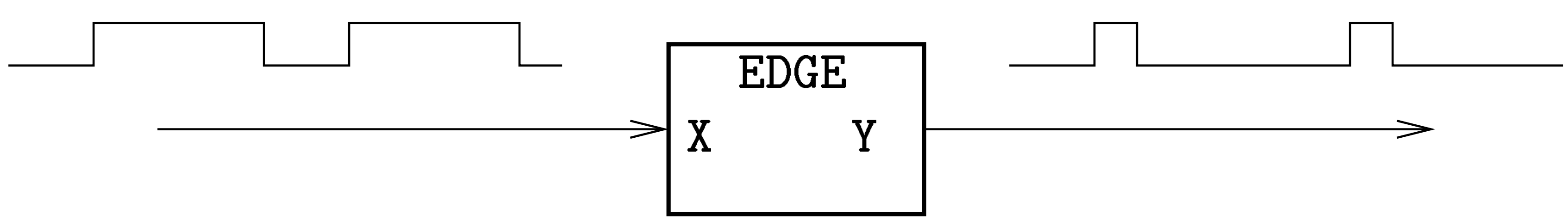
# Lustre - cycles

init.
value

previous
value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | | | | | | | | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



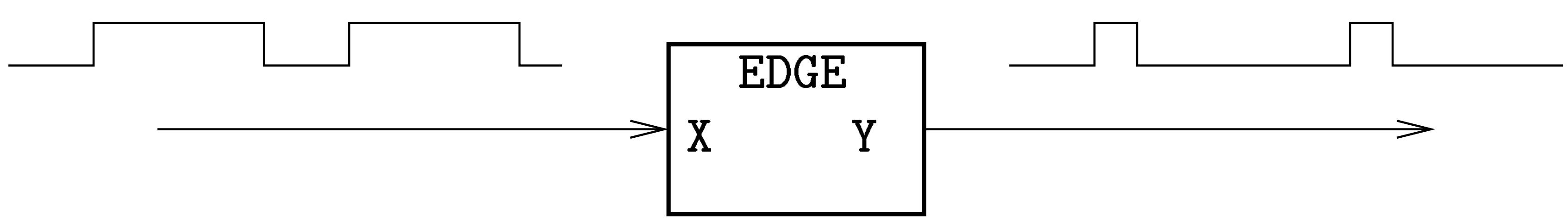
# Lustre - cycles

init.
value

previous
value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | | | | | | | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



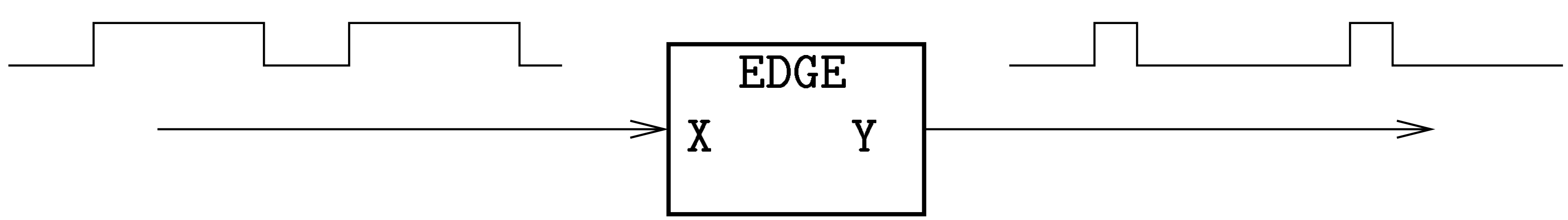
# Lustre - cycles

init.
value

previous
value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | T | | | | | | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



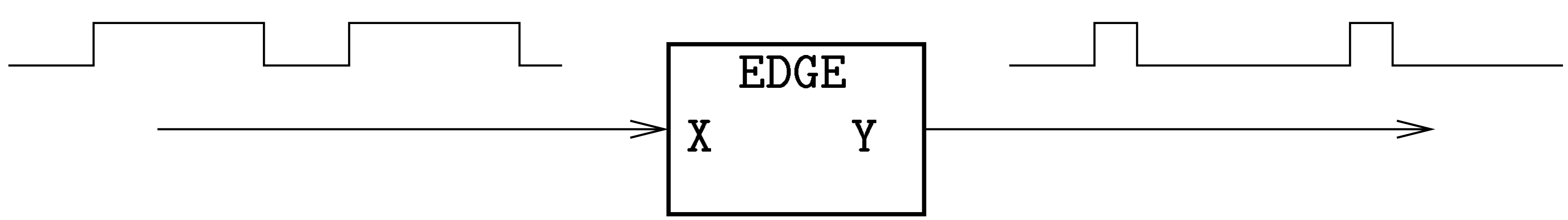
# Lustre - cycles

init. value

previous value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | | | | | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



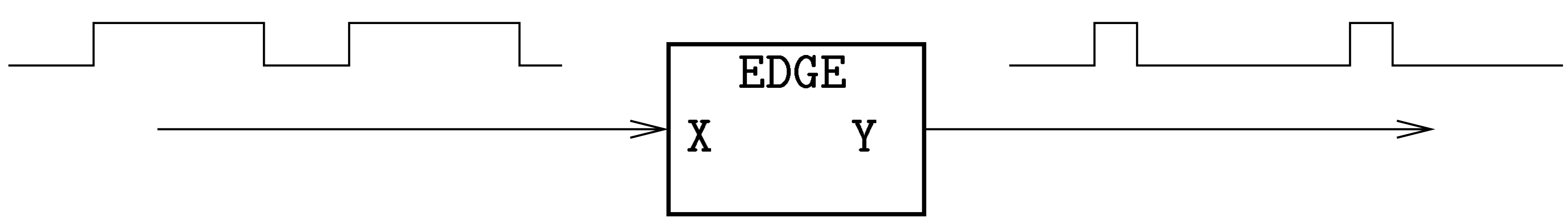
# Lustre - cycles

init.
value

previous
value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | F | | | | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



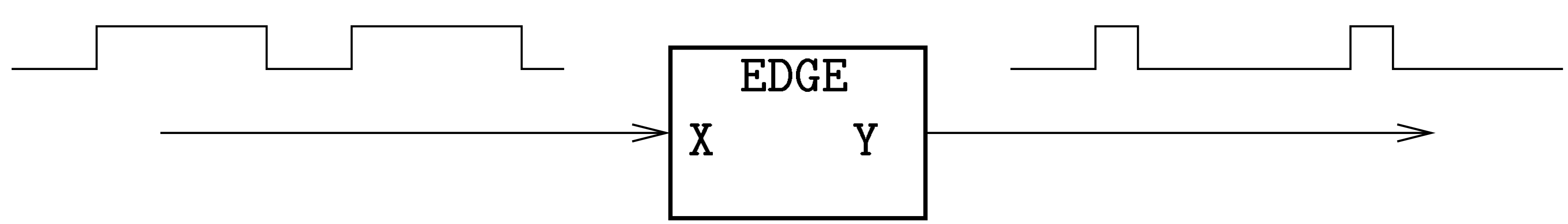
# Lustre - cycles

init. value

previous value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | F | F | | | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



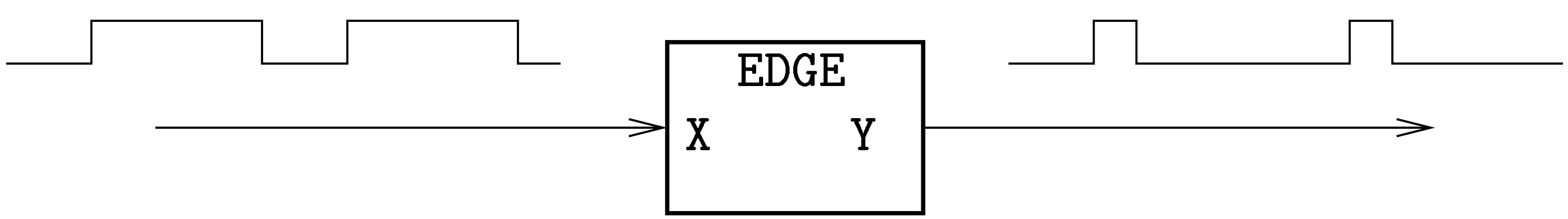
# Lustre - cycles

init. value

previous value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | F | F | F | | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



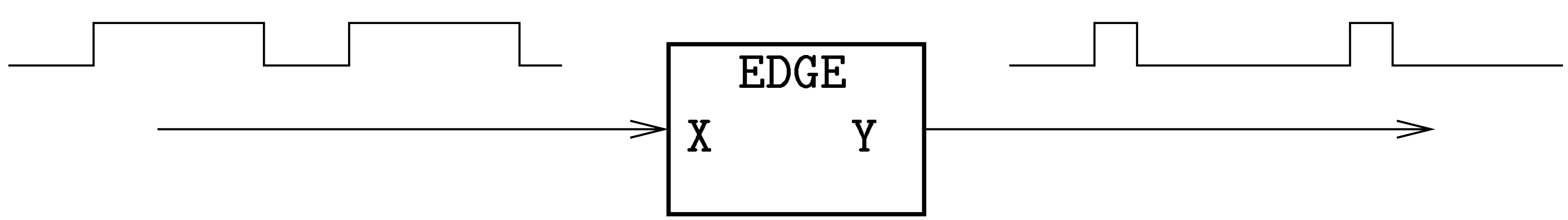
# Lustre - cycles

init. value

previous value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | F | F | F | T | |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



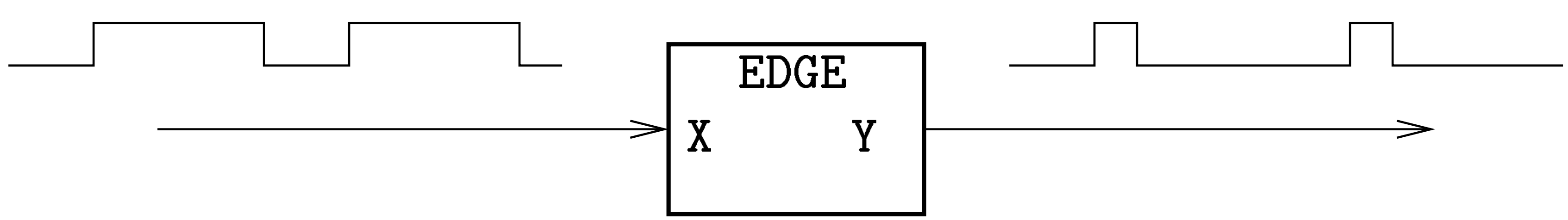
# Lustre - cycles

init. value

previous value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | F | F | F | T | F |
| X | F | T | T | T | F | F | T | T |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



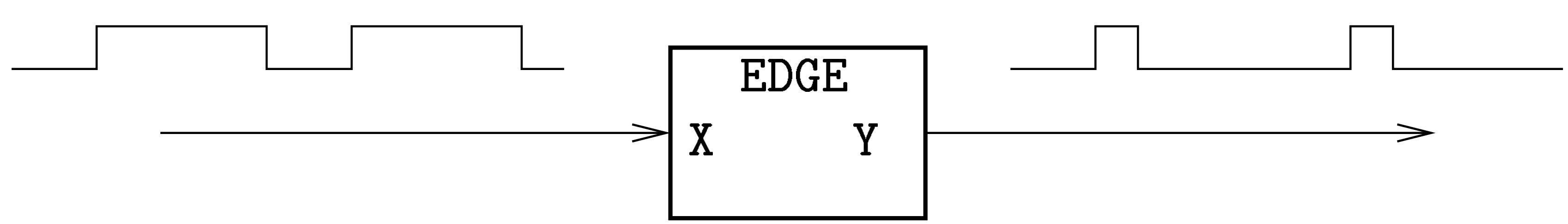
# Lustre - cycles

init. value

previous value of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | F | F | F | T | F | |
| X | F | T | T | T | F | F | T | T | ..... |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |



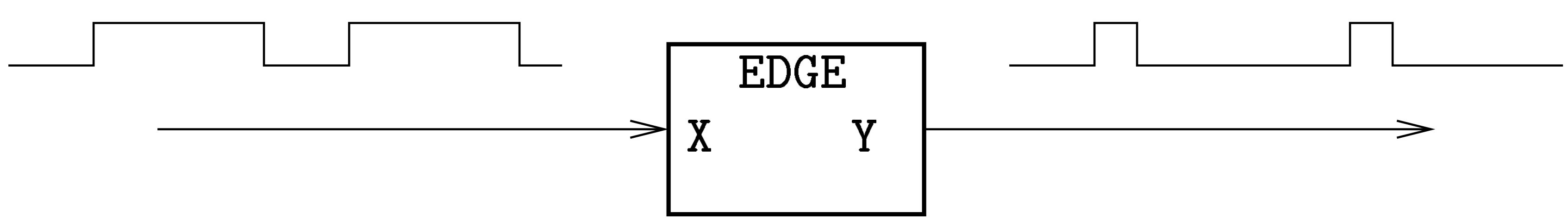
# Lustre - cycles

This node detects rising edges

of X

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Y | F | T | F | F | F | F | T | F | |
| X | F | T | T | T | F | F | T | T | ..... |
| cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |



# Lustre - constants

- Boolean constants: `true`, `false`
- Arithmetic constants
- Constants represent constant flows:
  - eg: `false` represents (`false`, `false`, `false`,...)

# Lustre - operators

- **Boolean**: `and`, `or`, `xor`, `not`
- **Arithmetic**: +, -, *, /, `div`, `mod`
- **Comparison**: <, >, <=, >=, =, <>
- `if...then...else`

# Lustre - operators

- `pre`: previous value
  If $A = (a_1, a_2, a_3, \ldots, a_n, \ldots)$
  then `pre`$(A) = (nil, a_1, a_2, a_3, \ldots, a_{n-1}, \ldots)$

- `->`: initialisation of a flow
  if $A = (a_1, a_2, a_3, \ldots, a_n, \ldots)$
  and $B = (b_1, b_2, b_3, \ldots, b_n, \ldots)$
  then $A \texttt{->} B = (a_1, b_2, b_3, \ldots, b_n, \ldots)$

# Lustre - Example

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| X | : | F | T | T | T | F | F | T | T |
| | : | | | | | | | | |
| | : | | | | | | | | |
| | : | | | | | | | | |
| | : | | | | | | | | |
| | : | | | | | | | | |
| Y | : | F | T | F | F | F | F | T | F |

# Lustre - Example

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| X | : | F | T | T | T | F | F | T | T |
| pre(X) | : | nil | F | T | T | T | F | F | T |
| | : | | | | | | | | |
| | : | | | | | | | | |
| | : | | | | | | | | |
| | : | | | | | | | | |
| Y | : | F | T | F | F | F | F | T | F |

# Lustre - Example

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| X | : | F | T | T | T | F | F | T | T |
| pre(X) | : | nil | F | T | T | T | F | F | T |
| not pre(X) | : | nil | T | F | F | F | T | T | F |
| | : | | | | | | | | |
| | : | | | | | | | | |
| | : | | | | | | | | |
| Y | : | F | T | F | F | F | F | T | F |

# Lustre - Example

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| X | : | F | T | T | T | F | F | T | T |
| pre(X) | : | nil | F | T | T | T | F | F | T |
| not pre(X) | : | nil | T | F | F | F | T | T | F |
| X and not pre(X) | : | nil | T | F | F | F | F | T | F |
| | : | | | | | | | | |
| | : | | | | | | | | |
| Y | : | F | T | F | F | F | F | T | F |

# Lustre - Example

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| X | : | F | T | T | T | F | F | T | T |
| pre(X) | : | nil | F | T | T | T | F | F | T |
| not pre(X) | : | nil | T | F | F | F | T | T | F |
| X and not pre(X) | : | nil | T | F | F | F | F | T | F |
| false | : | F | F | F | F | F | F | F | F |
| | : | | | | | | | | |
| Y | : | F | T | F | F | F | F | T | F |

# Lustre - Example

```
Y = false -> X and not pre(X)
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| X | : | F | T | T | T | F | F | T | T |
| pre(X) | : | nil | F | T | T | T | F | F | T |
| not pre(X) | : | nil | T | F | F | F | T | T | F |
| X and not pre(X) | : | nil | T | F | F | F | F | T | F |
| false | : | F | F | F | F | F | F | F | F |
| false -> X and not pre(X) | : | F | T | F | F | F | F | T | F |
| Y | : | F | T | F | F | F | F | T | F |

# Lustre and SCADE

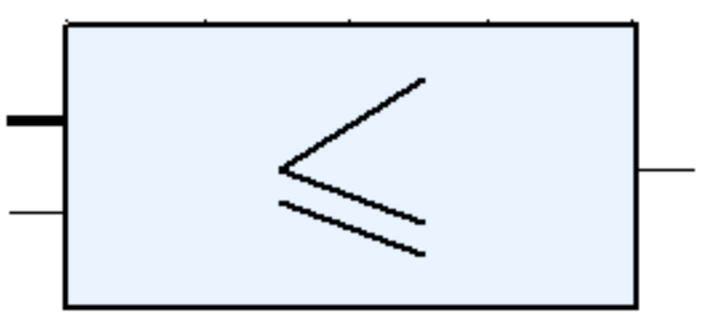
- In SCADE, models are represented graphically
- Scade operators correspond to Lustre operators

# Operators in SCADE

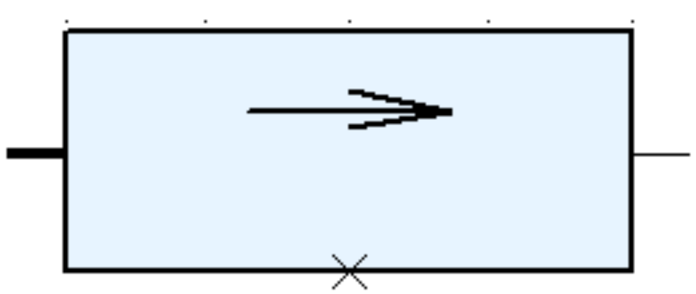

Constant flow


Input flow


Output flow



Comparison

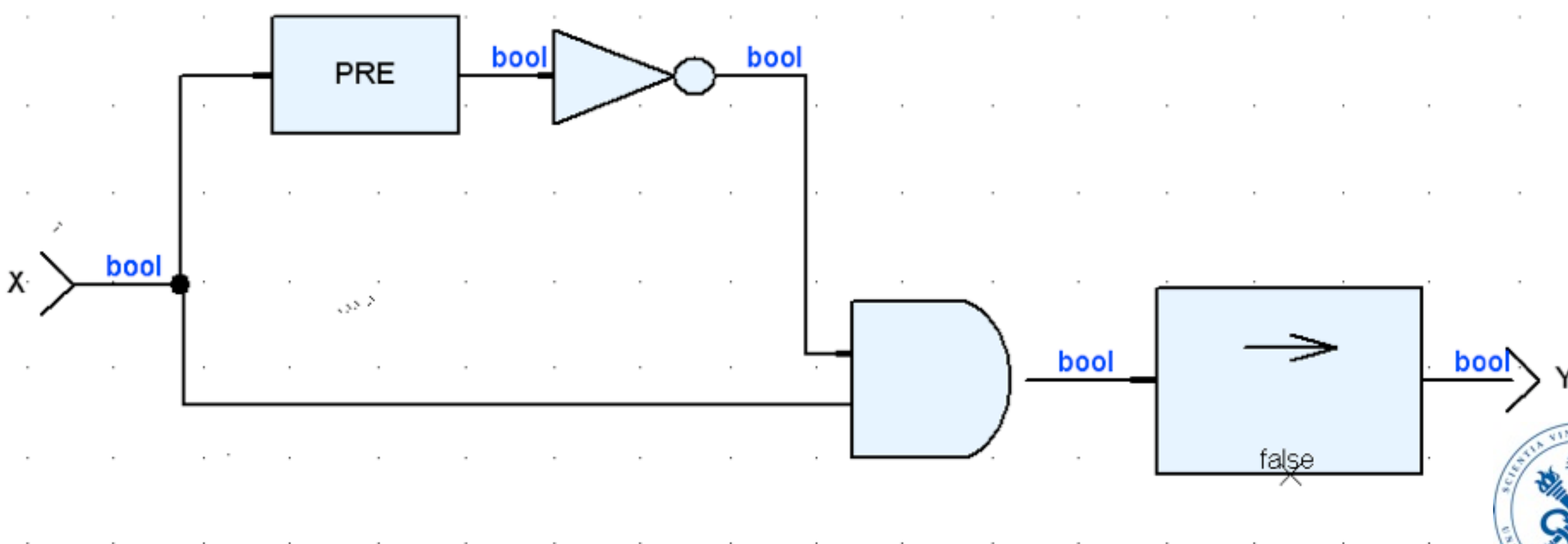


Initialisation

# Operators in SCADE

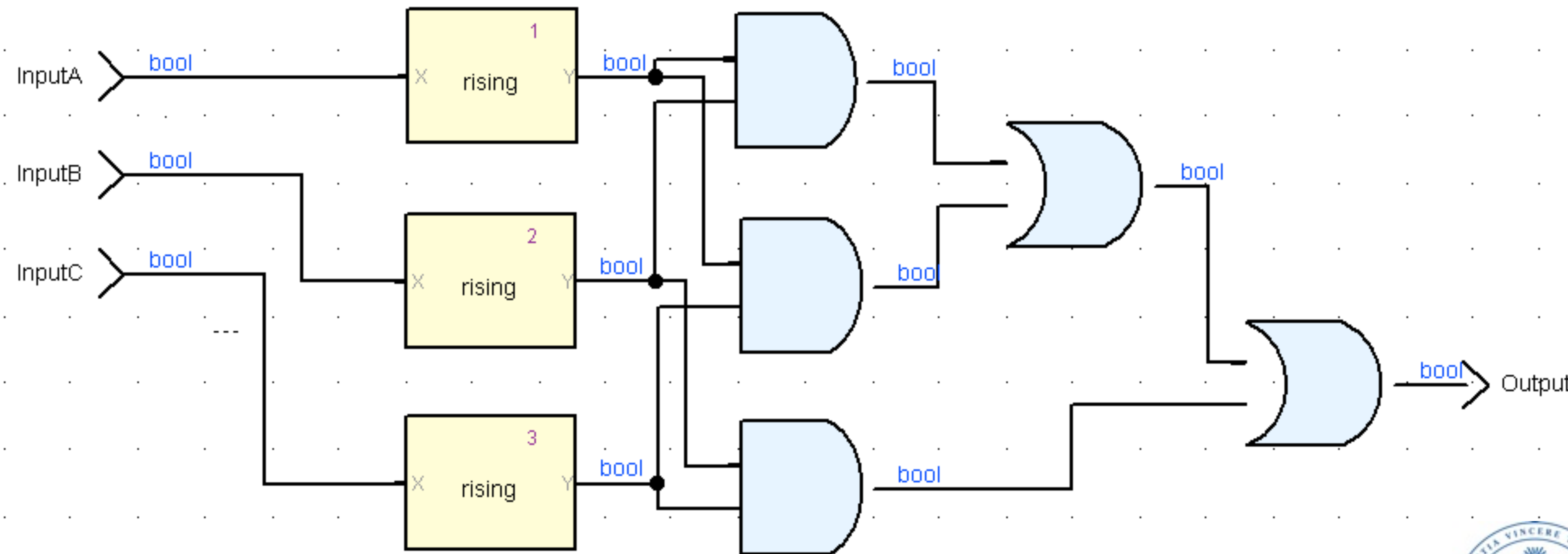
- Operators are combined to create nodes by linking them

```
Y = false -> X and not pre(X)
```



# Operators in SCADE

- Once a node is defined, it becomes an operator that can be invoked in other nodes



# SCADE - Simulator

- Models can be simulated in Scade
- The simulator computes the value of each flow at each cycle.
- The user can enter values for the input flows and observe the output flows (values can be plotted)

# SCADE - Simulator

- Demo... (fichier “operateurs”)

# SCADE - Verification

- Once the model has been debugged thanks to the simulator, it can be formally validated.
- SCADE allows to prove that a Boolean flow chosen by the user is always `true`.

# SCADE - Verification

- One thus has to design a flow (called a monitor) which is true iff a given property is verified
- This produces a proof of the property

$\Phi$ holds on the model

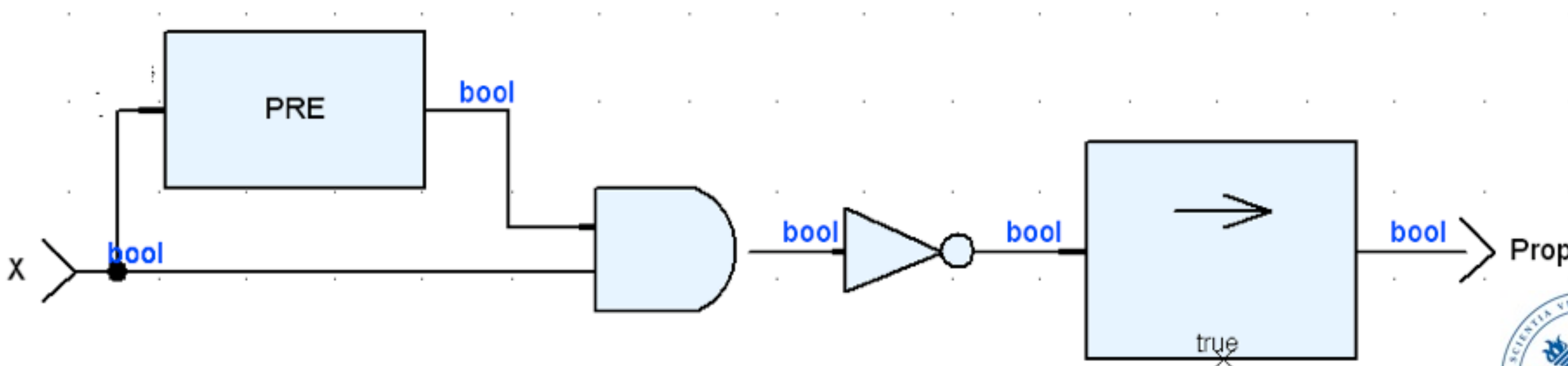
iff

`F = true`

proved by SCADE

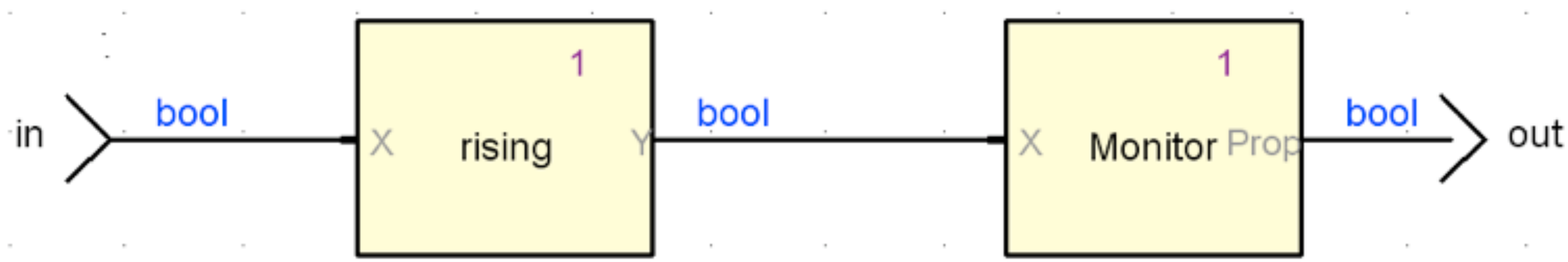
# SCADE - Verification

- Example: for the rising edge node, we want to make sure that we never have `true` as output during two consecutive cycles.
- `prop = true -> not(x and pre(x))`



# SCADE - Verification

- The monitor is then composed with the node to observe



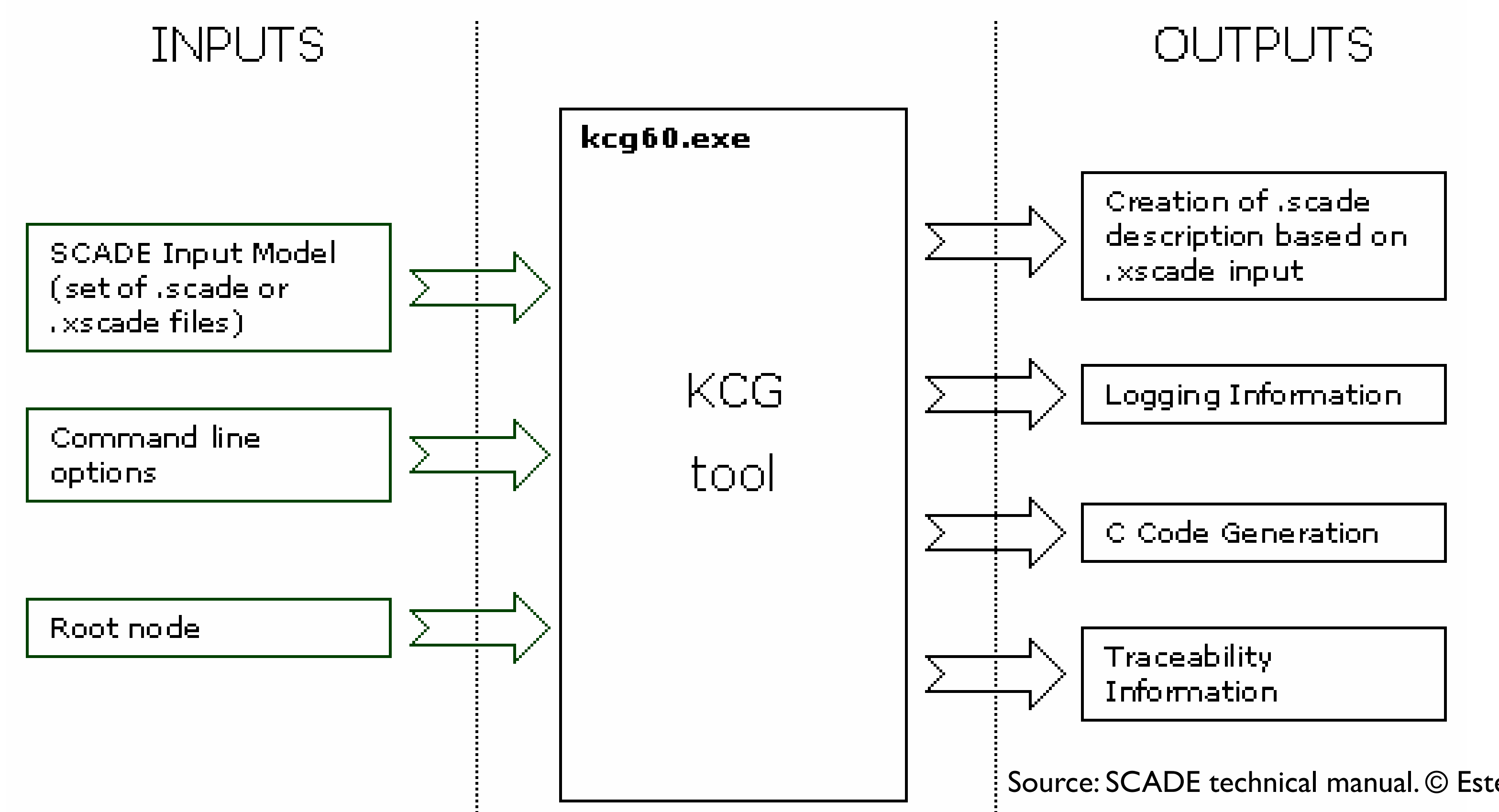

- And we check that the output of the monitor is always true

# SCADE - Verification

- Démo... (fichier “operateurs”)

# SCADE - Generation

- SCADE can generate C code from a given model
- Each node is translated as a function whose input and output parameters correspond to the input and outputs of the node

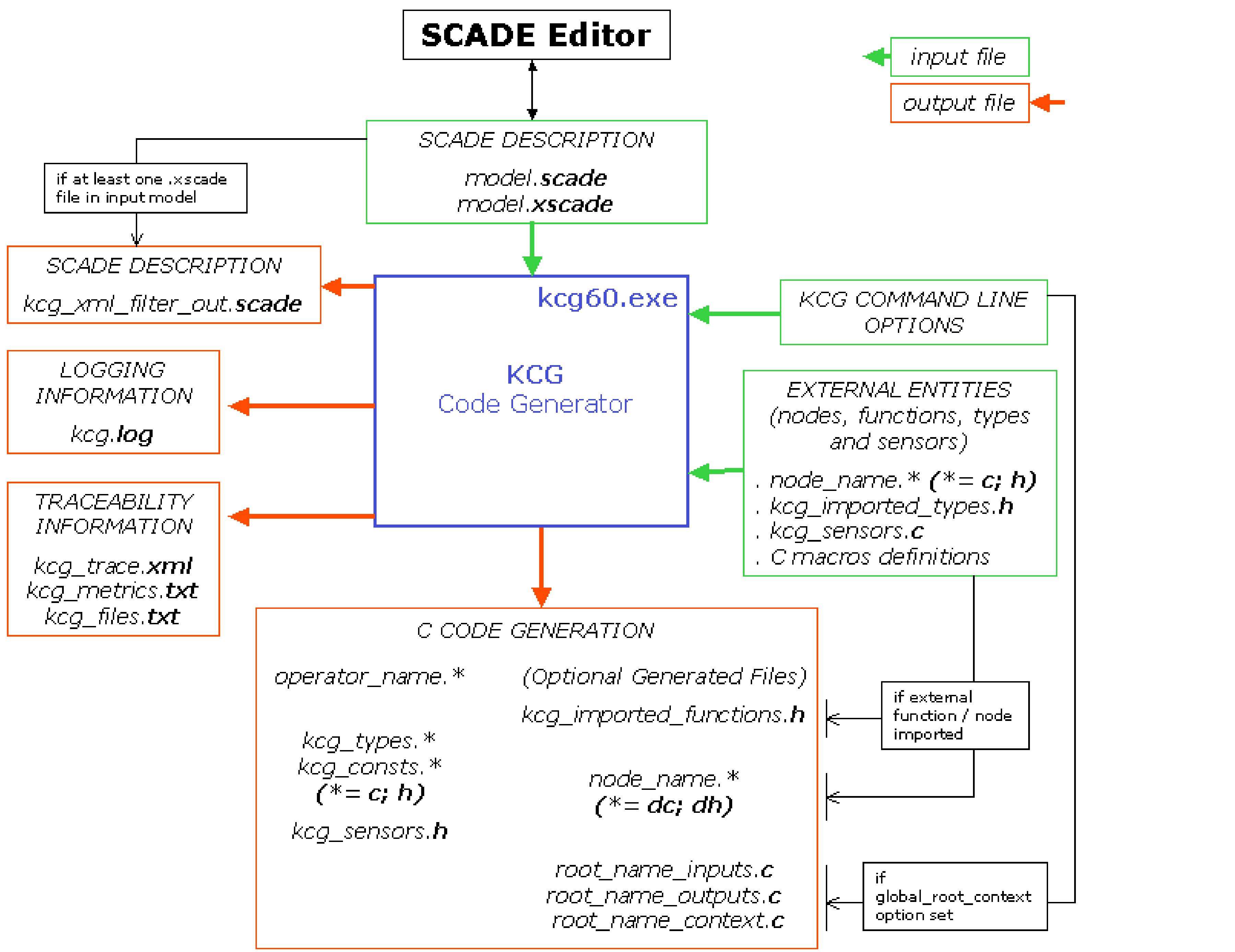



Source: SCADE technical manual. © Esterel Technologies

# SCADE - Generation

- Generated code enjoys the following characteristics:
  - It is deterministic (so is the model)
  - It contains no dynamic memory allocation (no "*out of memory*" at runtime on `malloc`...)
  - The mapping between the code and the model is (quite) clear
    - This allows to trace problems in the code back to the model

# SCADE - Generation



# SCADE - Generation

```
Y = false -> X and not pre(X)
```

## Generated code (rising.c):

```
/* $**************  KCG Version 6.0.0b (build i19) **************
** Command: s2c -config C:/Documents and Settings/Gilles/Bureau/operateurs/KCG 6.0\kcg_s2
** Generation date: 2008-12-09T17:05:12
*************************************************************$ */
#include "kcg_consts.h"
#include "kcg_sensors.h"
#include "rising.h"
void rising_reset(outC_rising *outC /* rising */)
{
  outC->init = kcg_true;
}
void rising(inC_rising *inC /* rising */, outC_rising *outC /* rising */)
{
  outC->Y = ((kcg_true ^outC->init)) & (((kcg_true ^outC->M_pre_)) & (inC->X));
  outC->M_pre_ = inC->X;
  outC->init = kcg_false;
}

/* $**************  KCG Version 6.0.0b (build i19) **************
** rising.c
** Generation date: 2008-12-09T17:05:12
*************************************************************$ */
```

# SCADE - Generation

```
Y = false -> X and not pre(X)
```

## Generated code (rising.h - part):

```
typedef struct { kcg_bool X /* rising::X */; } inC_rising;
/* rising */
/*   ========================== context type  ==========================   */

typedef struct {
  /*    ---------------------------  outputs  ---------------------------    */

  kcg_bool Y /* rising::Y */;
  /*    -----------------------  no local probes  -----------------------    */

  /*    -------------------  initialisation variables  -------------------    */

  kcg_bool init;
  /*    -------------------------  local memories  -----------------------    */

  kcg_bool M_pre_;
  /*    --------------------  no sub nodes' contexts  --------------------    */

} outC_rising;
/* rising */
/*   =======================  state vector type  =======================
```

# SCADE - Generation

```
Y = false -> X and not pre(X)
```

## Genrated Lustre Code (part):

```
/* $**************  KCG Version 6.0.0b (build i19) *************
** -
** Generation date: 2008-12-09T17:05:12
*************************************************************$ */
/* $**************  KCG Version 6.0.0b (build i19) *************
** Command: x2s C:/Documents and Settings/Gilles/Bureau/operateurs/rising.xscade
** Generation date: 2008-12-09T17:05:12
*************************************************************$ */

/* xscade source: C:/Documents and Settings/Gilles/Bureau/operateurs/rising.xscade */
node rising(X : bool) returns (Y : bool)
var
  _L1 : bool;
  _L3 : bool;
  _L4 : bool;
  _L5 : bool;
  _L6 : bool;
let
  _L1= X;
  Y= _L3;
  _L3= false -> _L4;
  _L4= _L6 and _L1;
  _L5= pre _L1;
  _L6= not _L5;
tel;
```

# SCADE - Generation

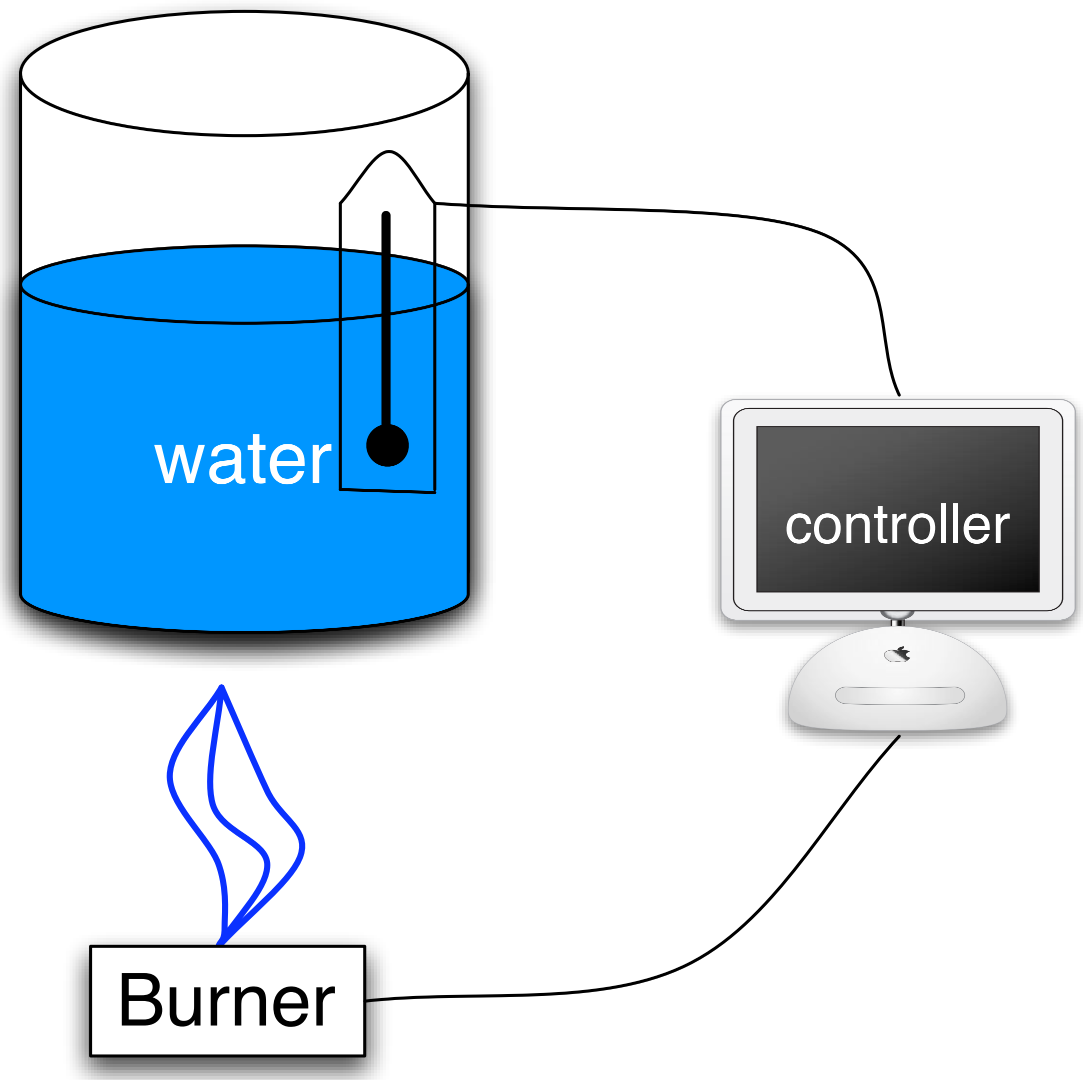
```
Y = false -> X and not pre(X)
```

## Traceability (kcg_trace.xml)


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- $*************** KCG Version 6.0.0b (build i19) ***************
** Command: s2c -config C:/Documents and Settings/Gilles/Bureau/operateurs/KCG 6.0\kcg_s2c_config
** Generation date: 2008-12-09T17:05:12
*************************************************************$ -->
<Model date="2008-12-09T17:05:12" commandLine="s2c -config C:/Documents and Settings/Gilles/Bureau
<RootNode scadeName="rising" targetCycleFct="rising" targetInitFct="rising_reset" headerFile="r
  <InCtxType targetName="inC_rising"/>
  <OutCtxType targetName="outC_rising"/>
  <StateVector targetName="rising_SV"/>
  <Input scadeName="X" scadeType="bool" targetName="X" targetType="kcg_bool"/>
  <Output scadeName="Y" scadeType="bool" targetName="Y" targetType="kcg_bool" inCtx="true"/>
</RootNode>
<PredefType scadeName="char" targetName="kcg_char"/>
<PredefType scadeName="bool" targetName="kcg_bool"/>
<PredefType scadeName="real" targetName="kcg_real"/>
<PredefType scadeName="int" targetName="kcg_int"/>
<Package scadeName="truthtables">
  <EnumType targetName="TruthTableValues_truthtables">
    <EnumVal scadeName="T" targetName="T_truthtables"/>
    <EnumVal scadeName="F" targetName="F_truthtables"/>
    <EnumVal scadeName="X" targetName="X_truthtables"/>
  </EnumType>
</Package>
</Model>
```

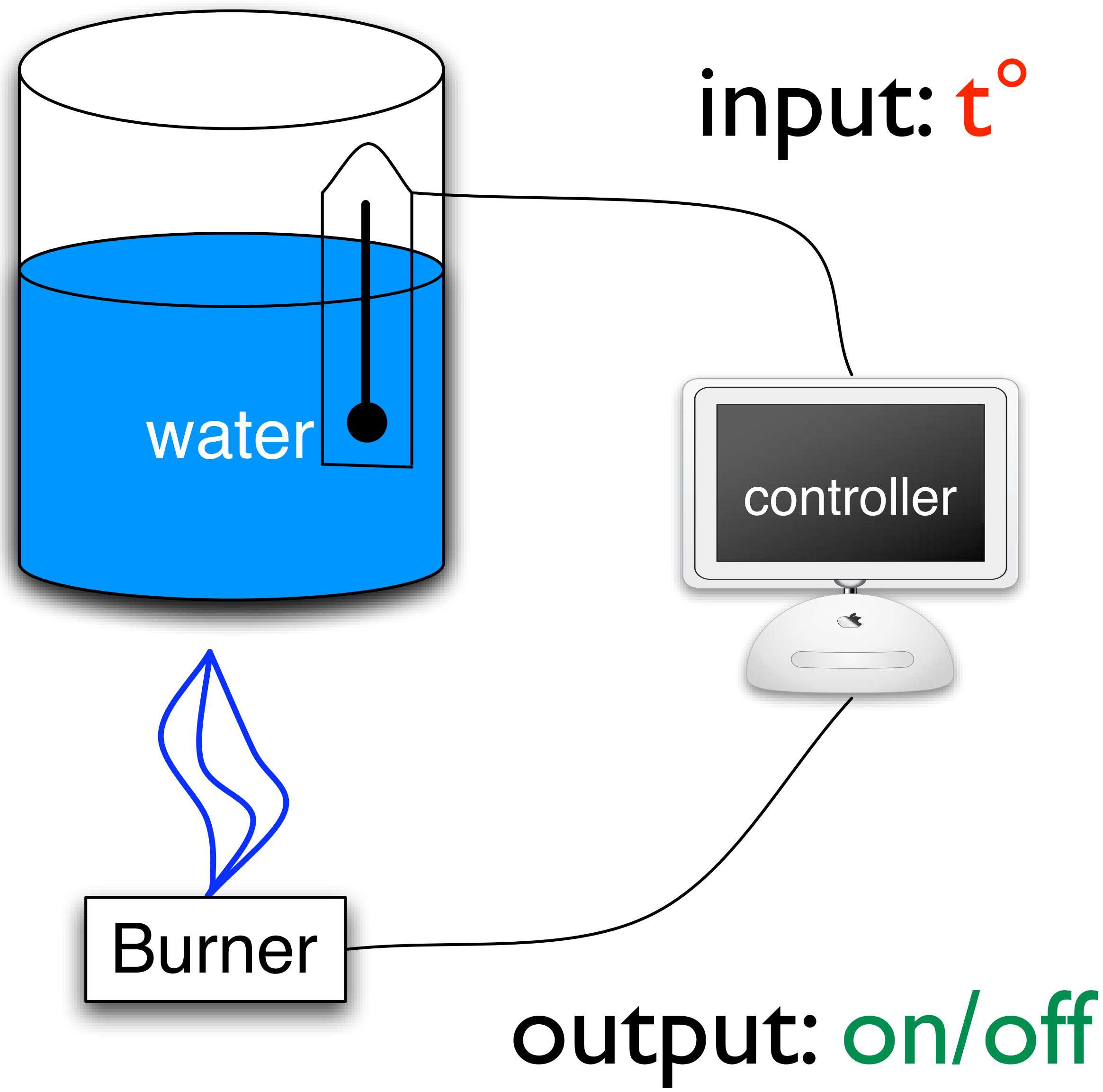
# Example - gas burner

- The controller must keep the temperature within a given interval
- Example: between 50 and 60 degrees (included)
- Initially, the temp. is in the interval



# Example - gas burner

- The controller turns the input flow (t°) into an output flow (Boolean)



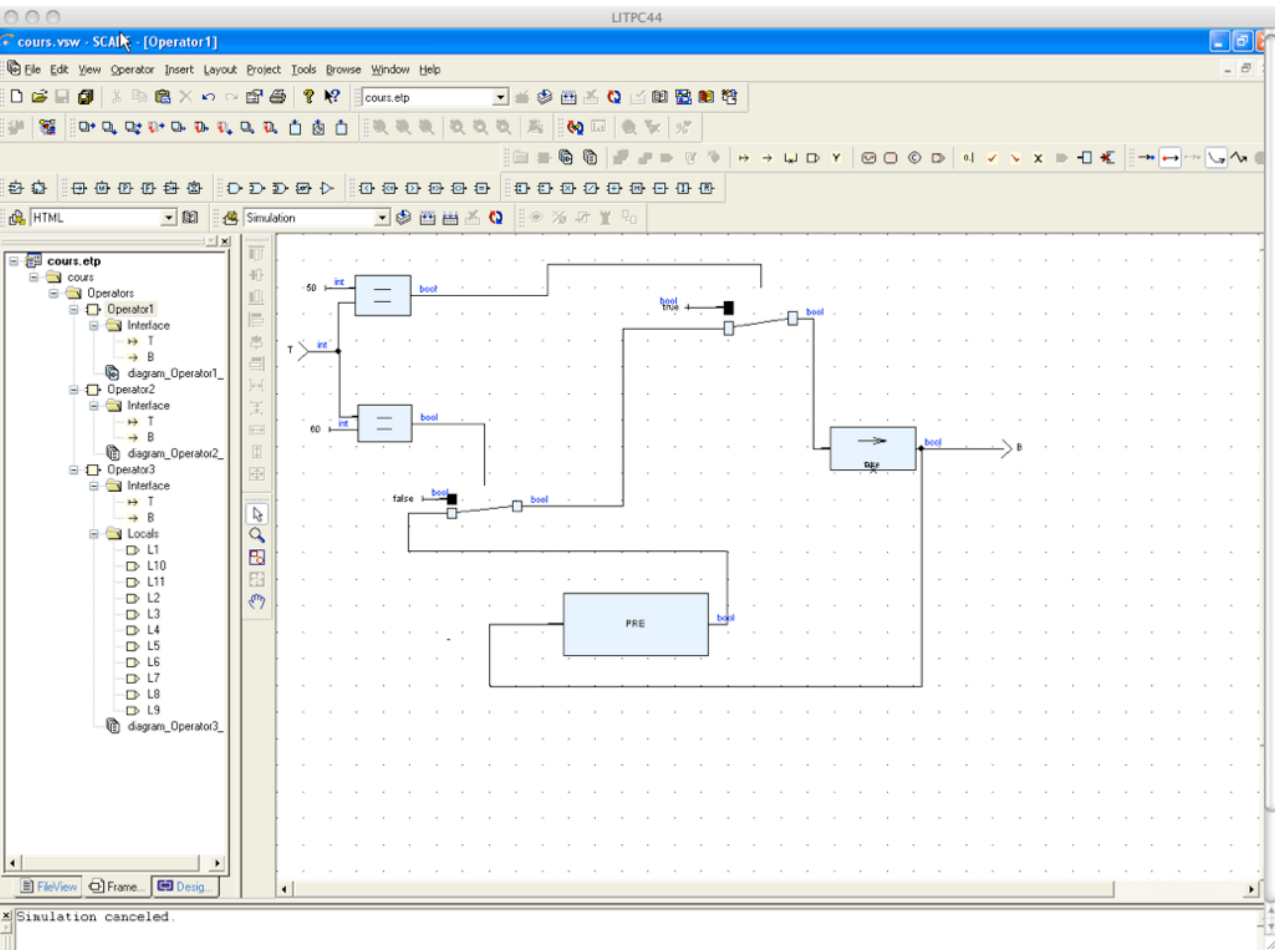
# Example - gas burner

- To find a proper model of the controller, we need hypothesis on the environment (= what has to be controlled)
  - When the burner switches on, temperature rises
  - When the burner switches off, temperature goes down

# Example - gas burner

- We propose the following control strategy:
  - When temperature reaches 50 °c, turn on the burner
  - When it reaches 60 °c, turn off the burner.

```
B = false -> (
     if T=50 then true
     else if T=60 then false
     else pre(B)
    )
```




vendredi 2 mars 2012

# Example - gas burner

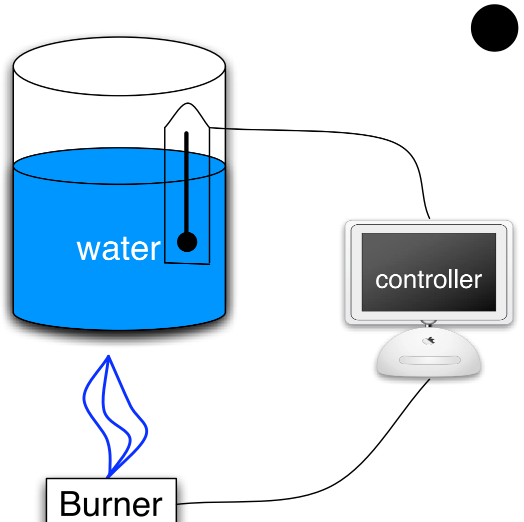
- To check that model of the controller we need a model of the environment
    - environment = burner + thermometer + tank
- This allow to simulate & verify the whole system

# Example - gas burner

- Hypothesis
  - Burner switches instantaneously
  - When heated, the temp. in the tank grows by 3°/time unit
  - When not heated, the temp. goes down by 1°/tu
  - Thermometer is precise

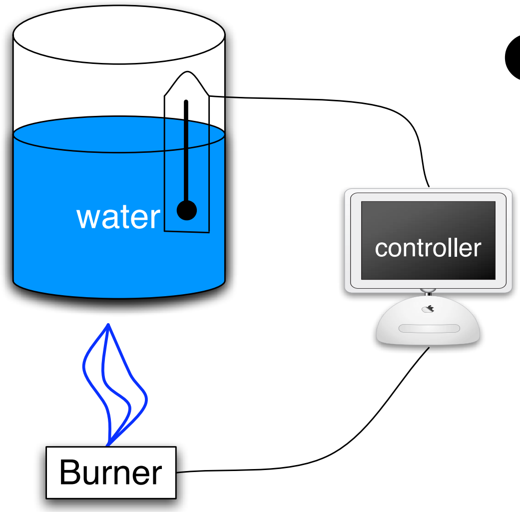
# Example - gas burner

- Thermometer:
  - 1 input: the temperature (real) of the tank: `temp`
  - 1 output: the reading: `reading`
  - equation:
    `temp = reading`



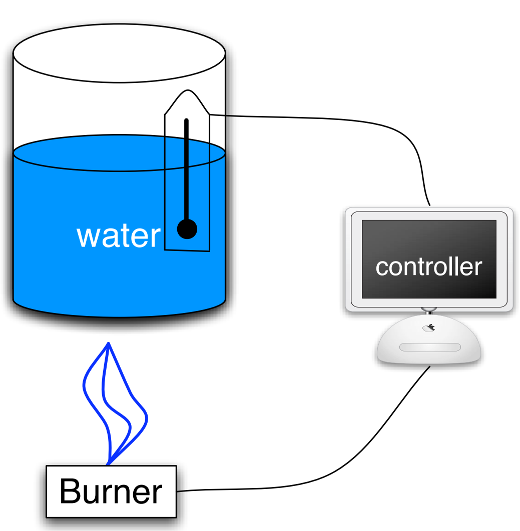
# Example - gas burner

- Burner:
  - 1 input: the command from the controller: `switch`
  - 1 output: the "flame" (Boolean): `heating`
  - equation: `heating = switch`

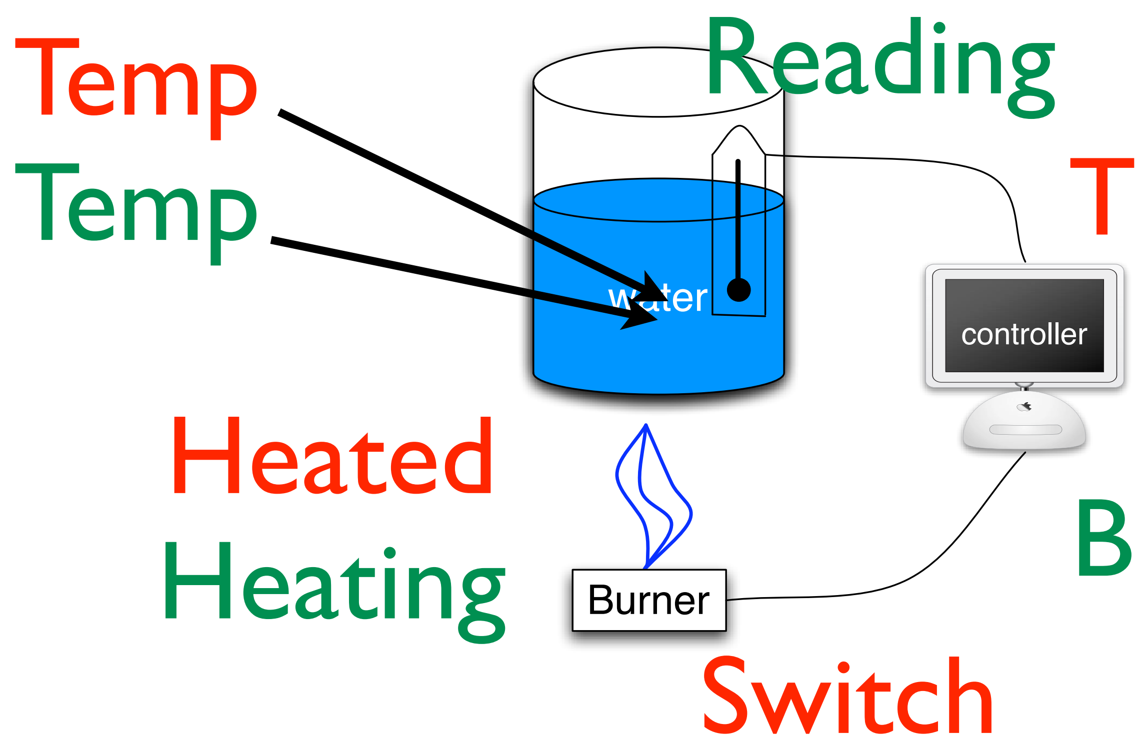


# Example - gas burner

- Tank:
  - 1 input: a Boolean indicating if the tank is being heated or not: `heated`
  - 1 output: the temperature: `temp`
  - equation:
    `temp = 55 -> (if heated then pre(temp)+3 else pre (temp)-1)`

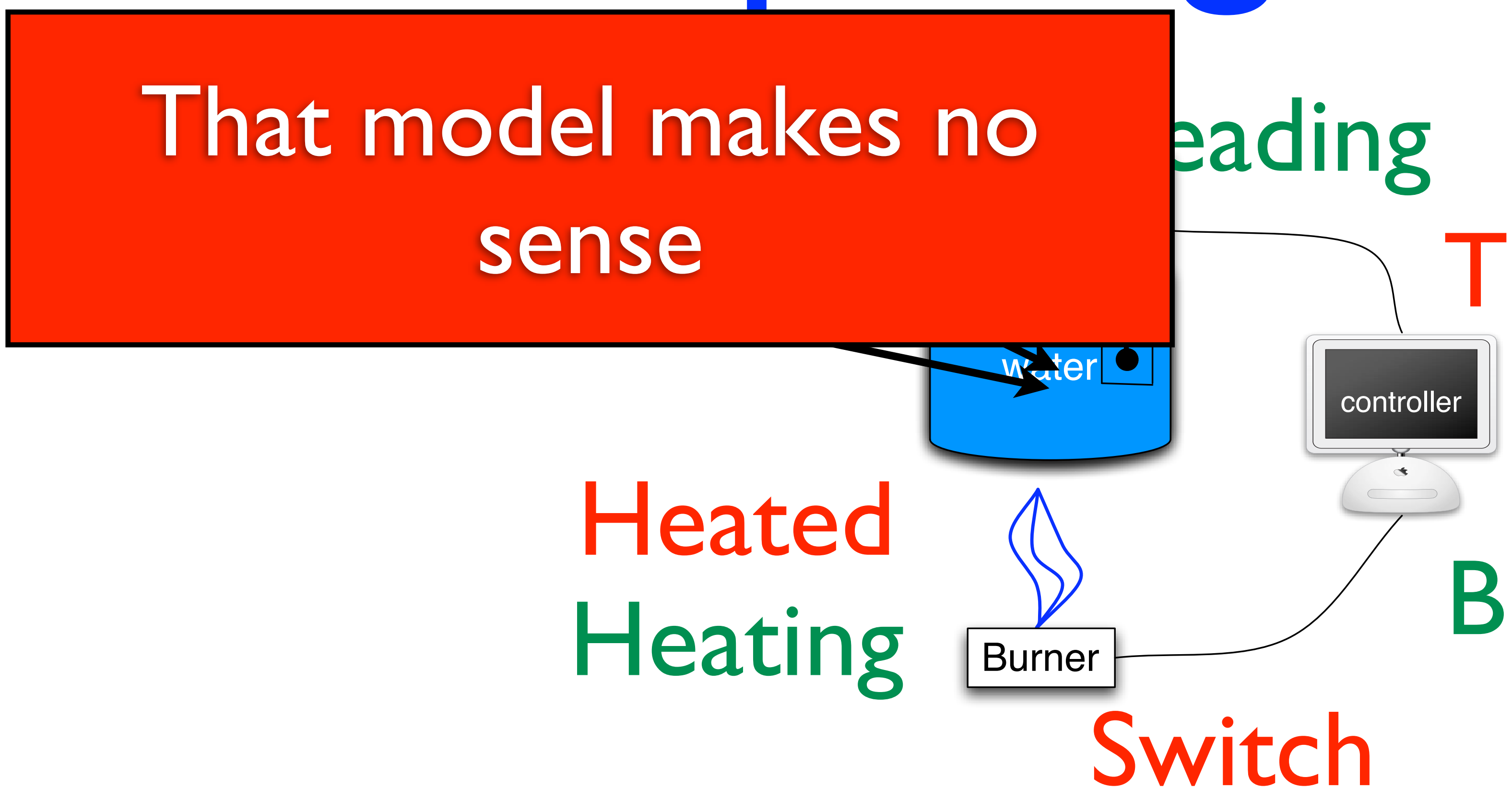


# Example - gas burner




```
Tank/Heated = Burner/Heating ;
CNTL/T = Thermo/Reading ;
Thermo/Temp = Tank/Temp ;
Burner/Switch = CNTL/B ;
```

# Example - gas burner




```
Tank/Heated = Burner/Heating ;
CNTL/T = Thermo/Reading ;
Thermo/Temp = Tank/Temp ;
Burner/Switch = CNTL/B ;
```

# Example - gas burner

That model makes no sense

eading

T

Equations are constraints, not instructions

Switch

```
Tank/Heated = Burner/Heating ;
CNTL/T = Thermo/Reading ;
Thermo/Temp = Tank/Temp ;
Burner/Switch = CNTL/B ;
```

# Example - gas burner

That model makes no sense

eading

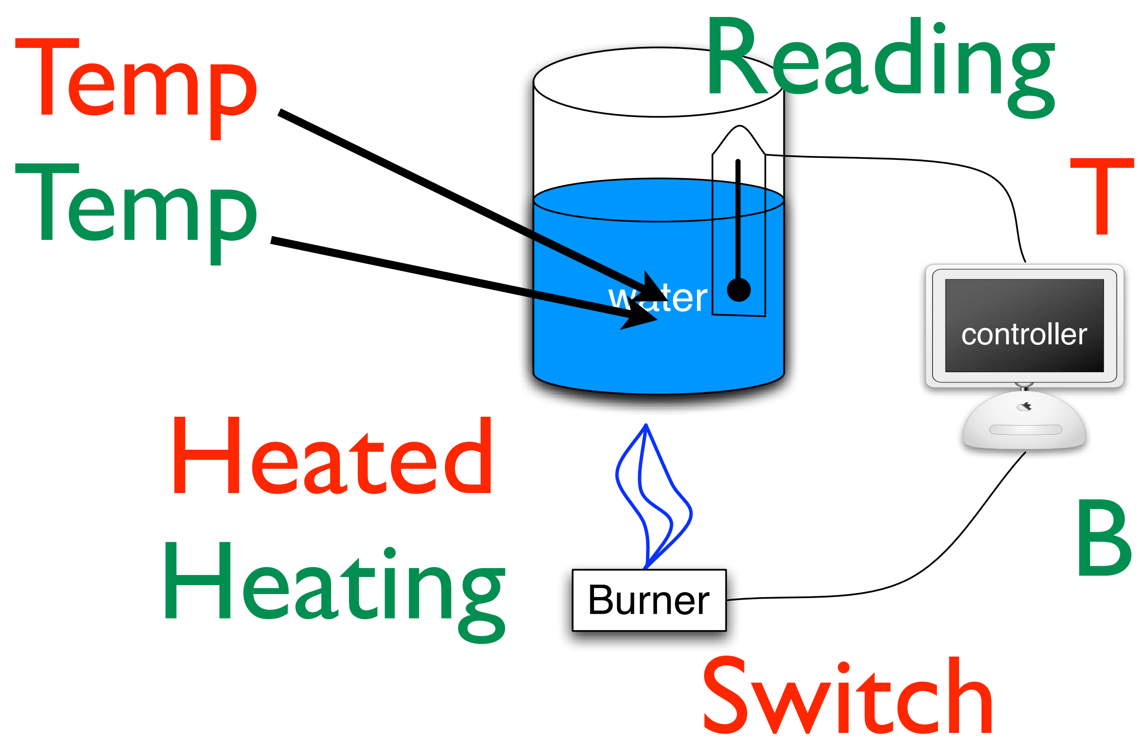
T

Equations are constraints, not instructions

There are cyclic dependencies !

```
Tank/Heated = B
CNTL/T = Thermo/Reading ;
Thermo/Temp = Tank/Temp ;
Burner/Switch = CNTL/B ;
```

# Example - gas burner




```
Tank/Heated = Burner/Heating ;
CNTL/T = Pre (Thermo/Reading) ;
Thermo/Temp = Tank/Temp ;
Burner/Switch = CNTL/B ;
```

# Example - gas burner

- Démo... (fichier “cours”)

# Example - gas burner

- The simulator in SCADE allows us to find a *bug* in our controller
  - For some initial temperatures, we never reach exactly 60 °c, and the burner will never switch off

# Example - gas burner

```
B = false -> (
     if T=50 then true
     else if T=60 then false
     else pre(B)
    )
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| T: | 55 | ... | 56 | 59 | 62 | 63 | ... |
| B: | False | | True | True | True | True | |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | |
|---|---|---|---|
| T: | 55 | ... | 56 |
| B: | False | | True |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

T: 55 ... 56

B: False True

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | |
|---|---|---|---|---|
| T: | 55 | ... | 56 | 59 |
| B: | False | | True | True |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | |
|---|---|---|---|---|
| T: | 55 | | 56 | 59 |
| B: | False | ... | True | True |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | |
|---|---|---|---|---|
| T: | 55 | ... | 56 | 59 |
| B: | False | | True | True |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | | |
|---|---|---|---|---|---|
| T: | 55 | ... | 56 | 59 | 58 |
| B: | False | | True | True | False |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | | |
|---|---|---|---|---|---|
| T: | 55 | ... | 56 | 59 | 58 |
| B: | False | | True | True | False |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | | |
|---|---|---|---|---|---|
| T: | 55 | | 56 | 59 | 58 |
| B: | False | … | True | True | False |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| T: | 55 | ... | 56 | 59 | 58 | 57 |
| B: | False | | True | True | False | False |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | | | |
|---|---|---|---|---|---|---|
| T: | 55 | ... | 56 | 59 | 58 | 57 |
| B: | False | | True | True | False | False |

# Example - gas burner

- This can be seen as a limit on the *sampling rate* of the CPU.
- We need to widen the conditions

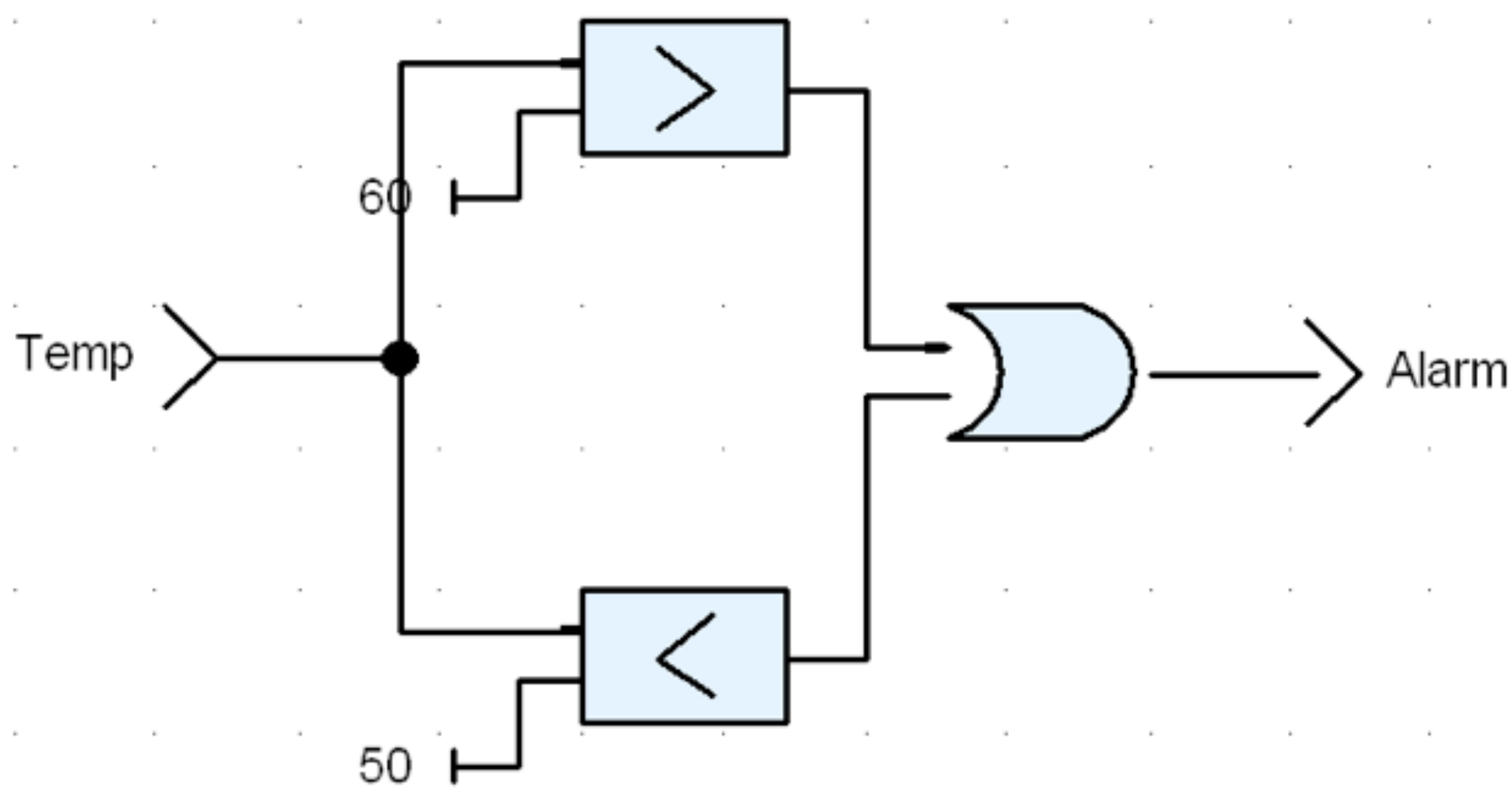
```
if T>=57 then false
```

Remind: `T` is the temperature computed at the previous cycle

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| T: | 55 | … | 56 | 59 | 58 | 57 | … |
| B: | False | | True | True | False | False | |

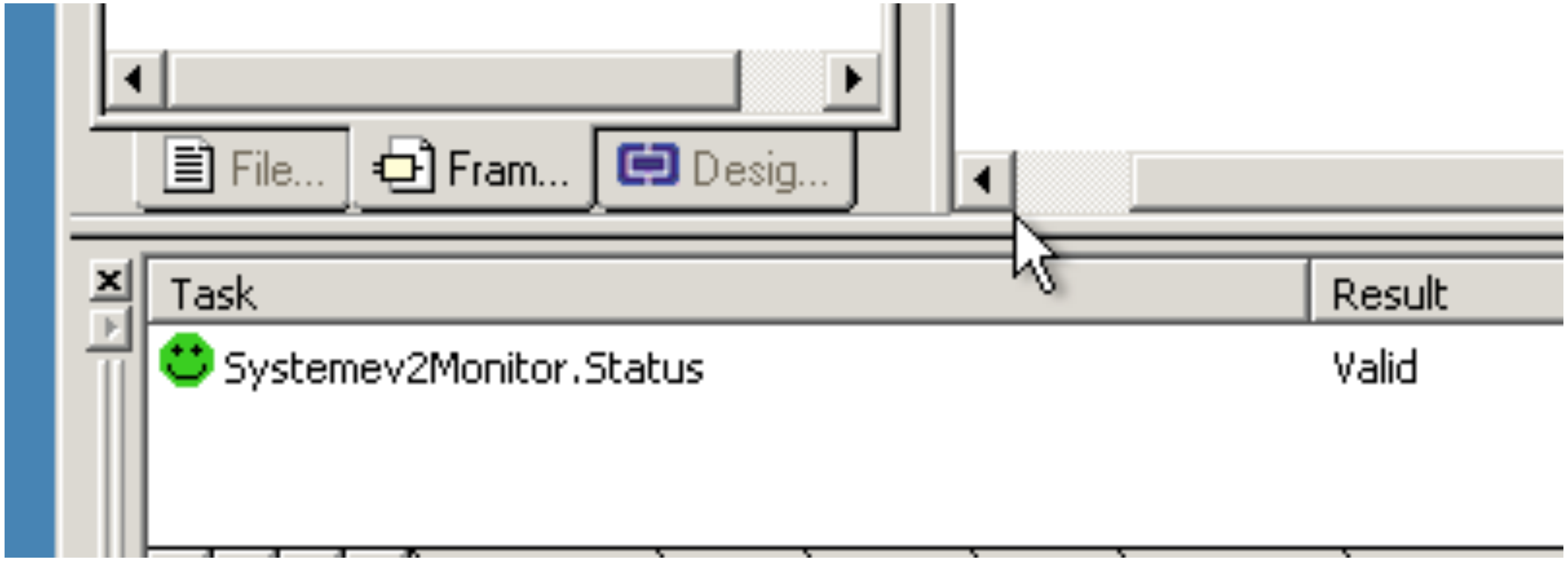
# Example - gas burner

- Let us now validate our controller by defining a monitor that will raise an alarm if the temperature goes off limits



# Example - gas burner

- Scade can prove that the property is valid...



# Example - gas burner

- Démo... (fichier system-moniteur.avi)

# Scade - State machines

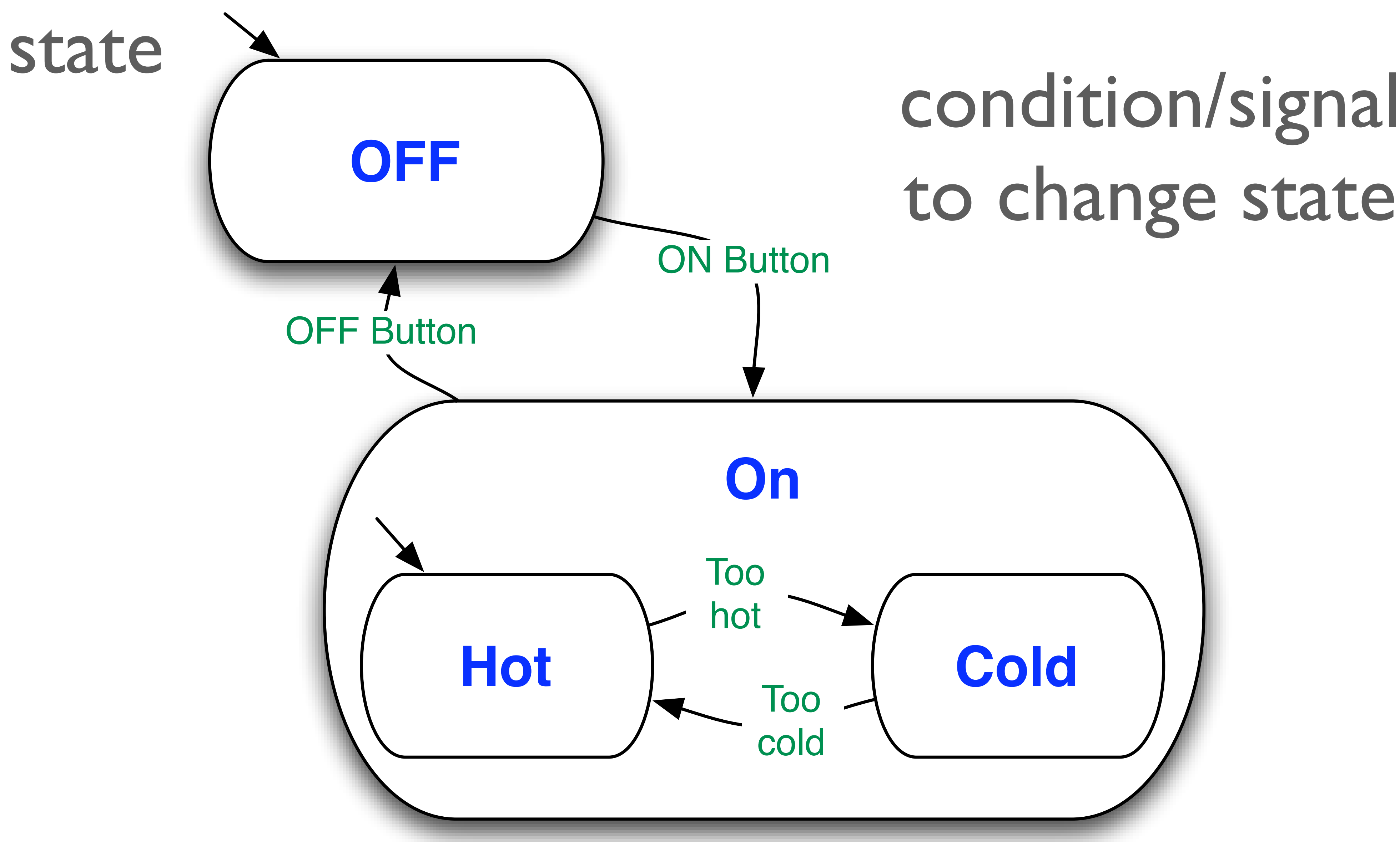
- The operators described so far manipulate data through flows
- In a realistic application, the way we manipulate date depends on the running mode

# Scade - State machines

- Example: air-co in a car
  - It can be on or off
  - When on, it can heat or refresh
  - There are thus three running modes
  - Outside *stimuli* (events) change the current running mode

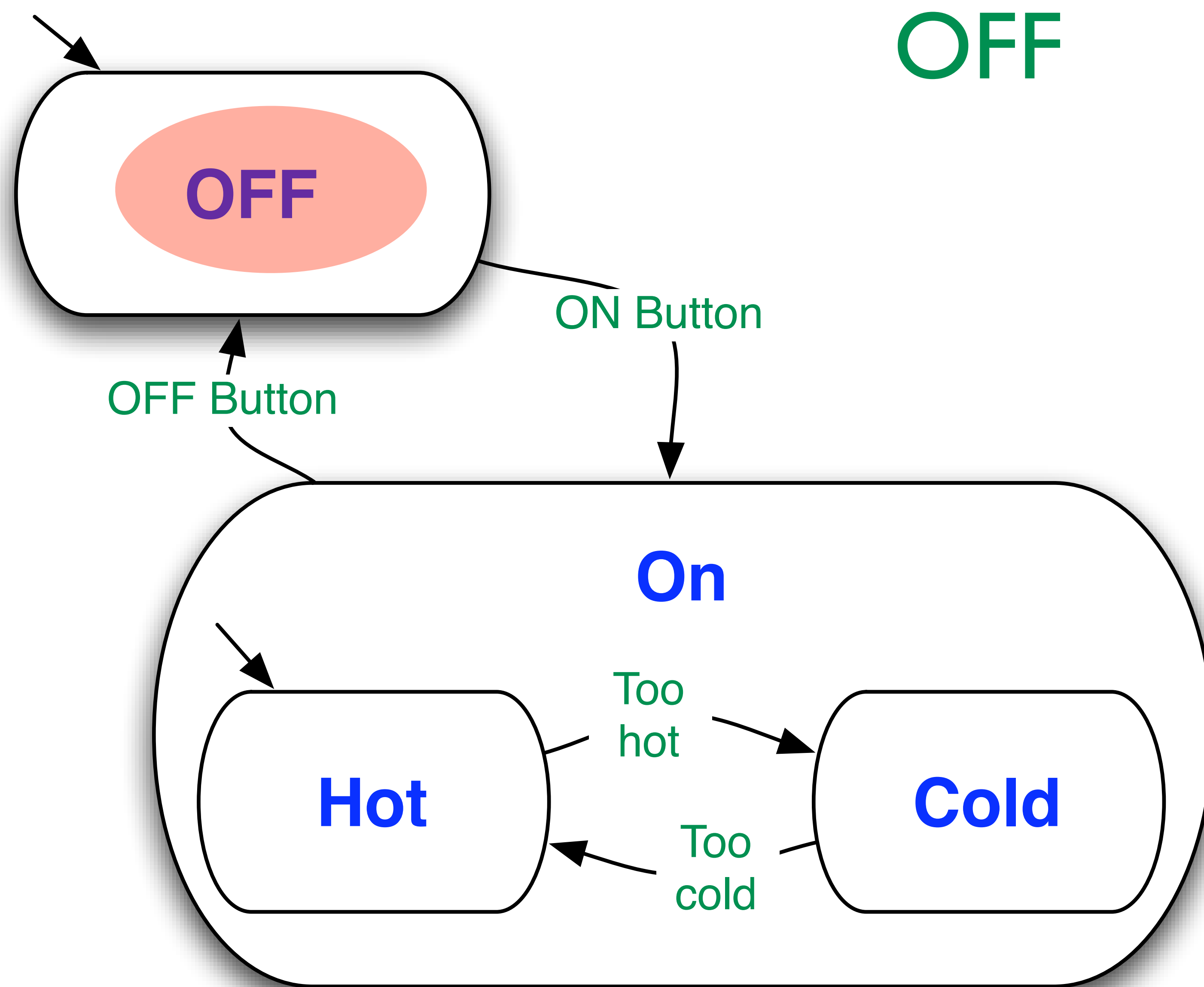
# Scade - State machines

- Example: car air-co



# Scade - State machines

- Example: car air-co

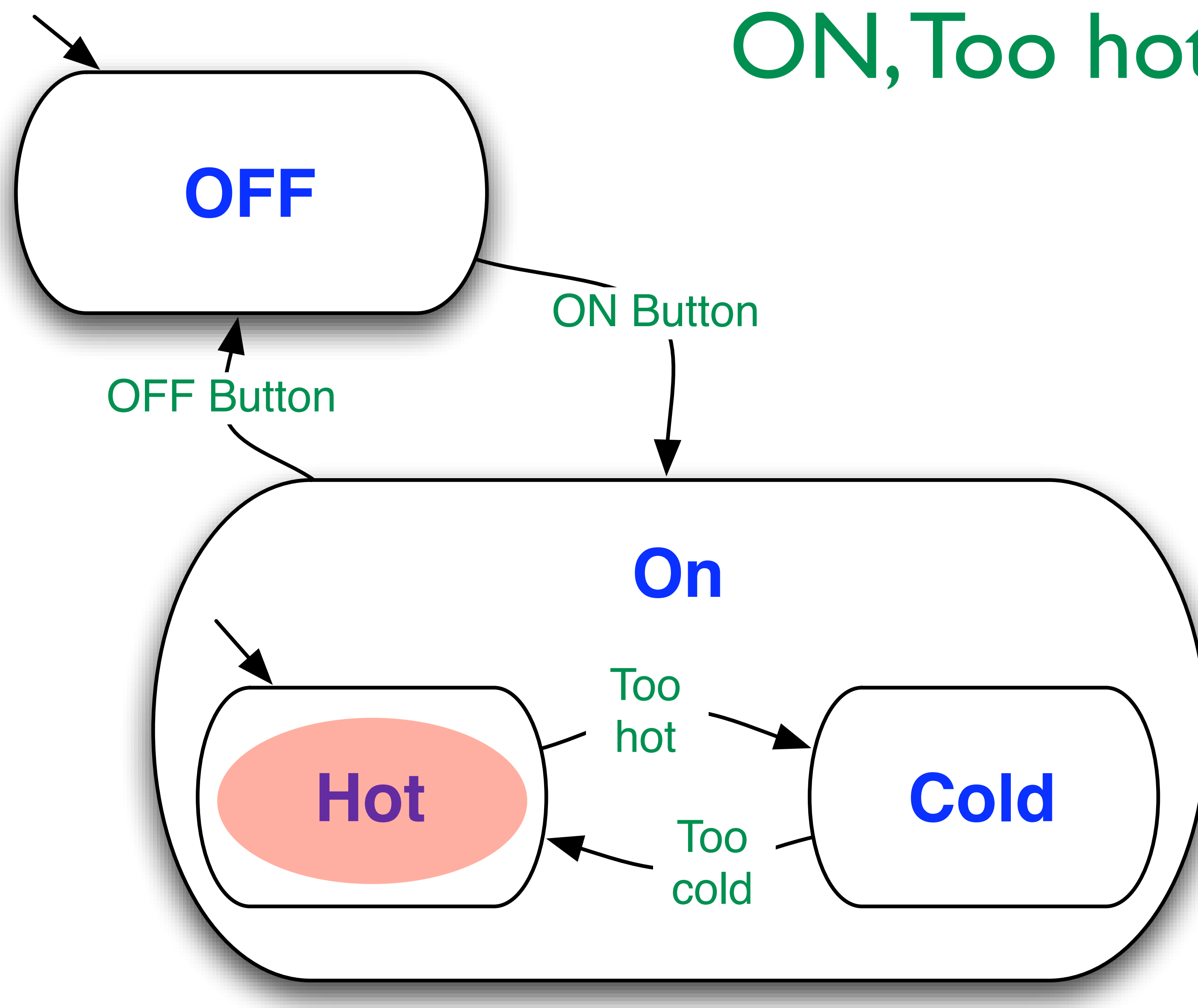



OFF

# Scade - State machines

- Example: car air-co

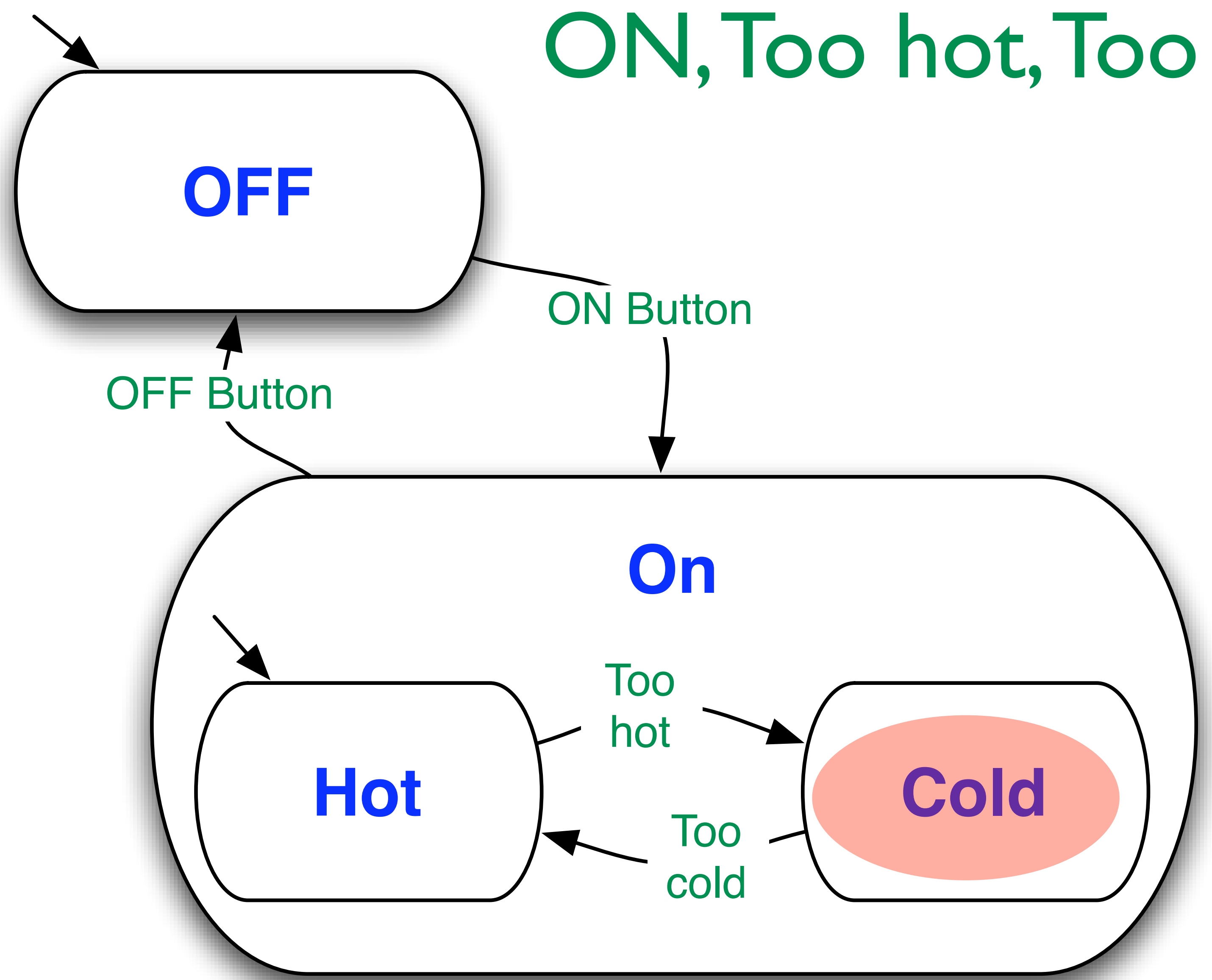
ON, Too hot



# Scade - State machines

- Example: car air-co

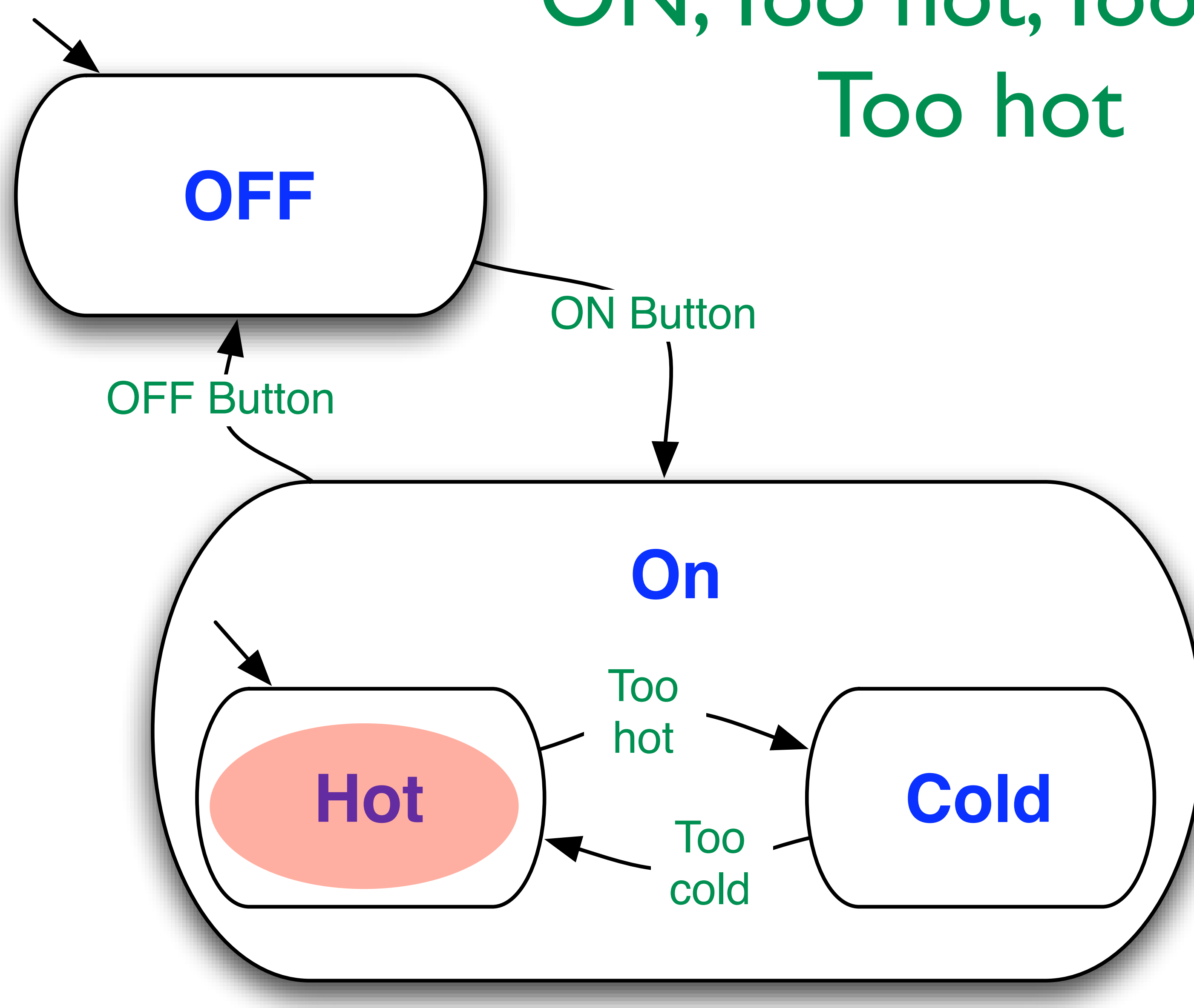
ON, Too hot, Too cold



# Scade - State machines

- Example: car air-co

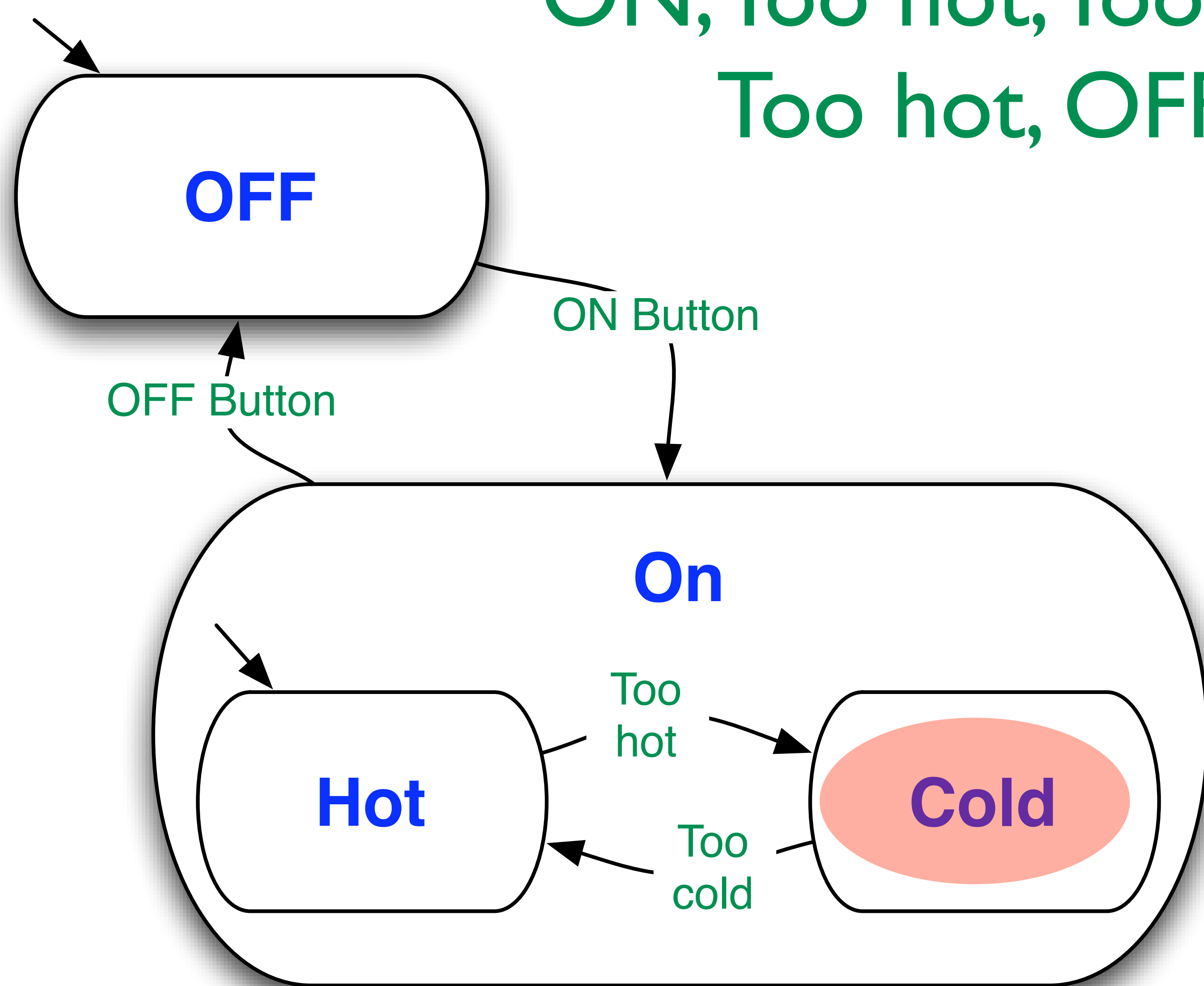
ON, Too hot, Too cold,
Too hot



# Scade - State machines

- Example: car air-co

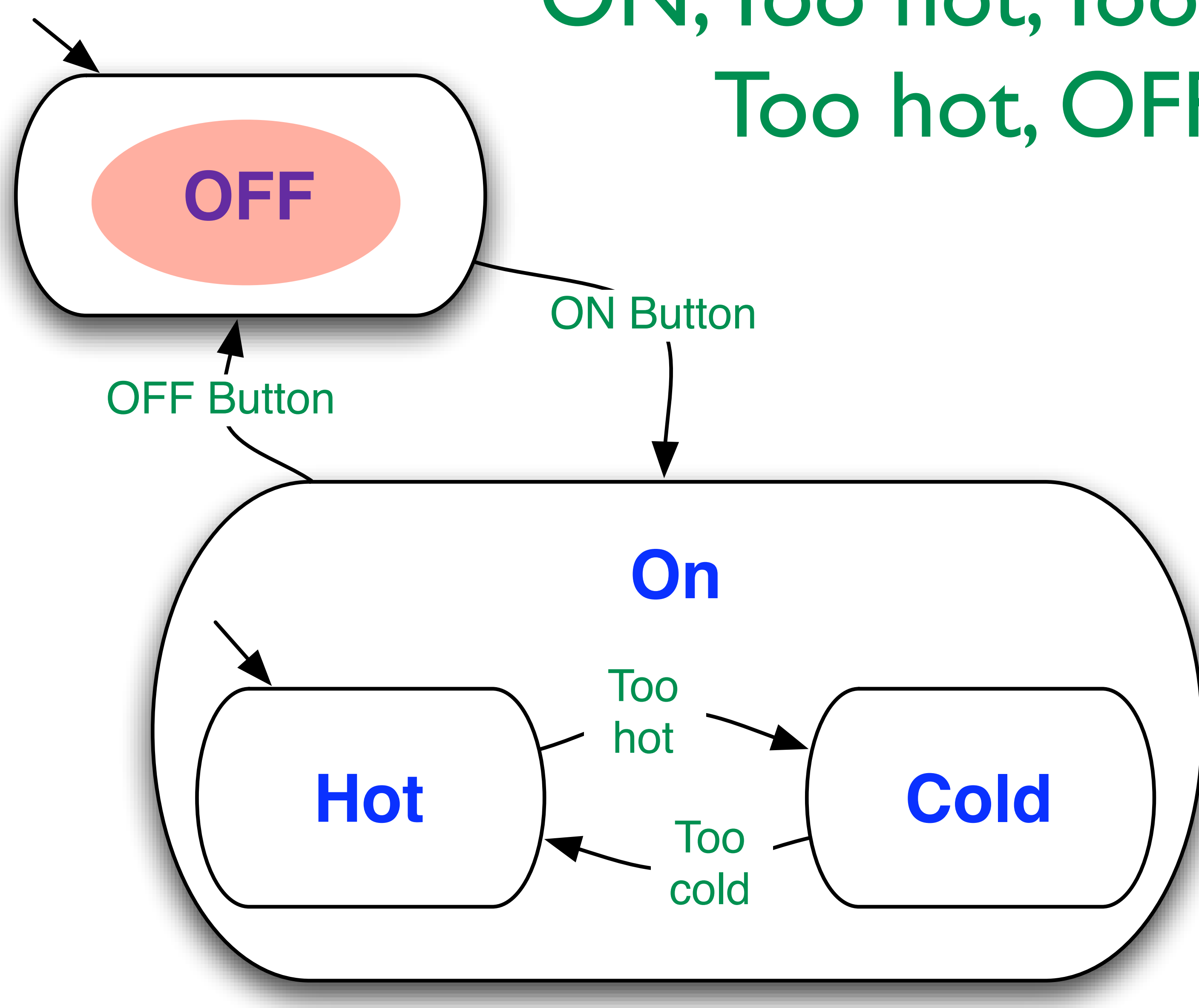
ON, Too hot, Too cold,
Too hot, OFF



# Scade - State machines

- Example: car air-co

ON, Too hot, Too cold,
Too hot, OFF



# State machines

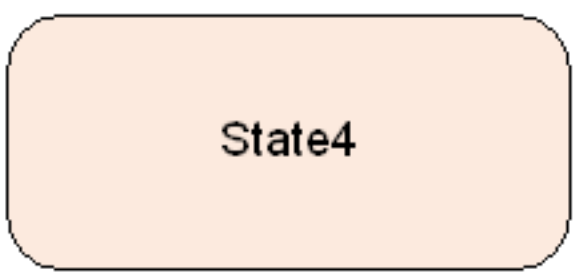
- State machines thus allow to combine two classical concepts:
  - data flows (via LUSTRE flows)
  - control flows (control depends upon the state of the state machine)

# Scade - State machines

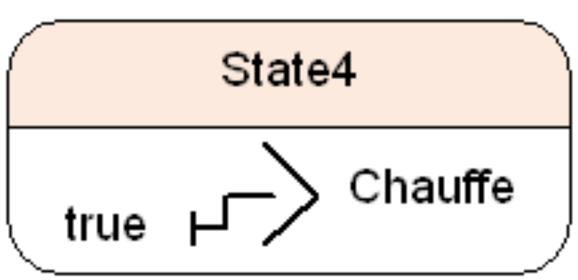
- Remark: We could do without state machines. We can encode them through flows:
  - a flow S encodes the current state.
  - the equations of the other flows contained `if` to test the value of S and determine the effect to apply.
- But state machines are more practical !

# Scade - State machines


One state machine




One state (w/o equation)




One state with one equation

# Scade - State machines

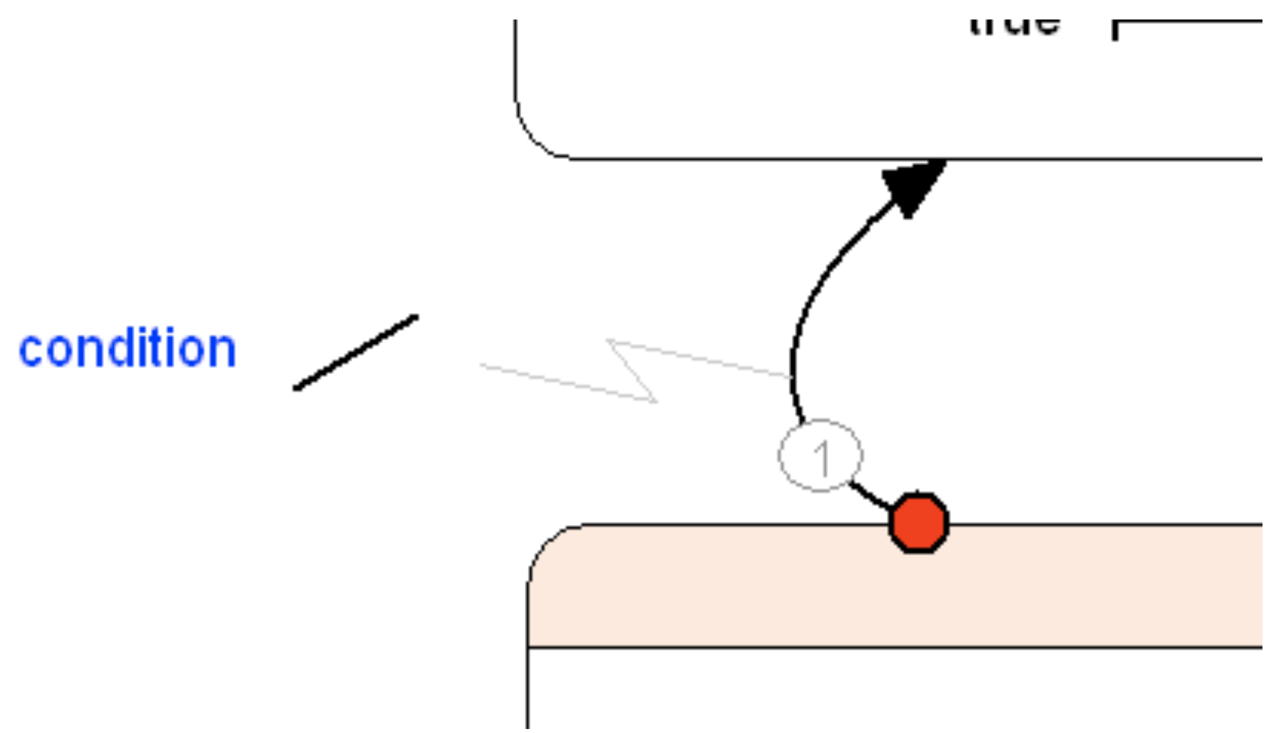
- Notion of signal:
  - Signals allow communications between state machines and:
    - other state machines
    - equations
    - the environment

# Scade - State machines

- Notion of signal:
  - A signal has a name.
  - It can be seen as a Boolean flow which is always `false`, except when the signal is sent.

# Scade - State Machines

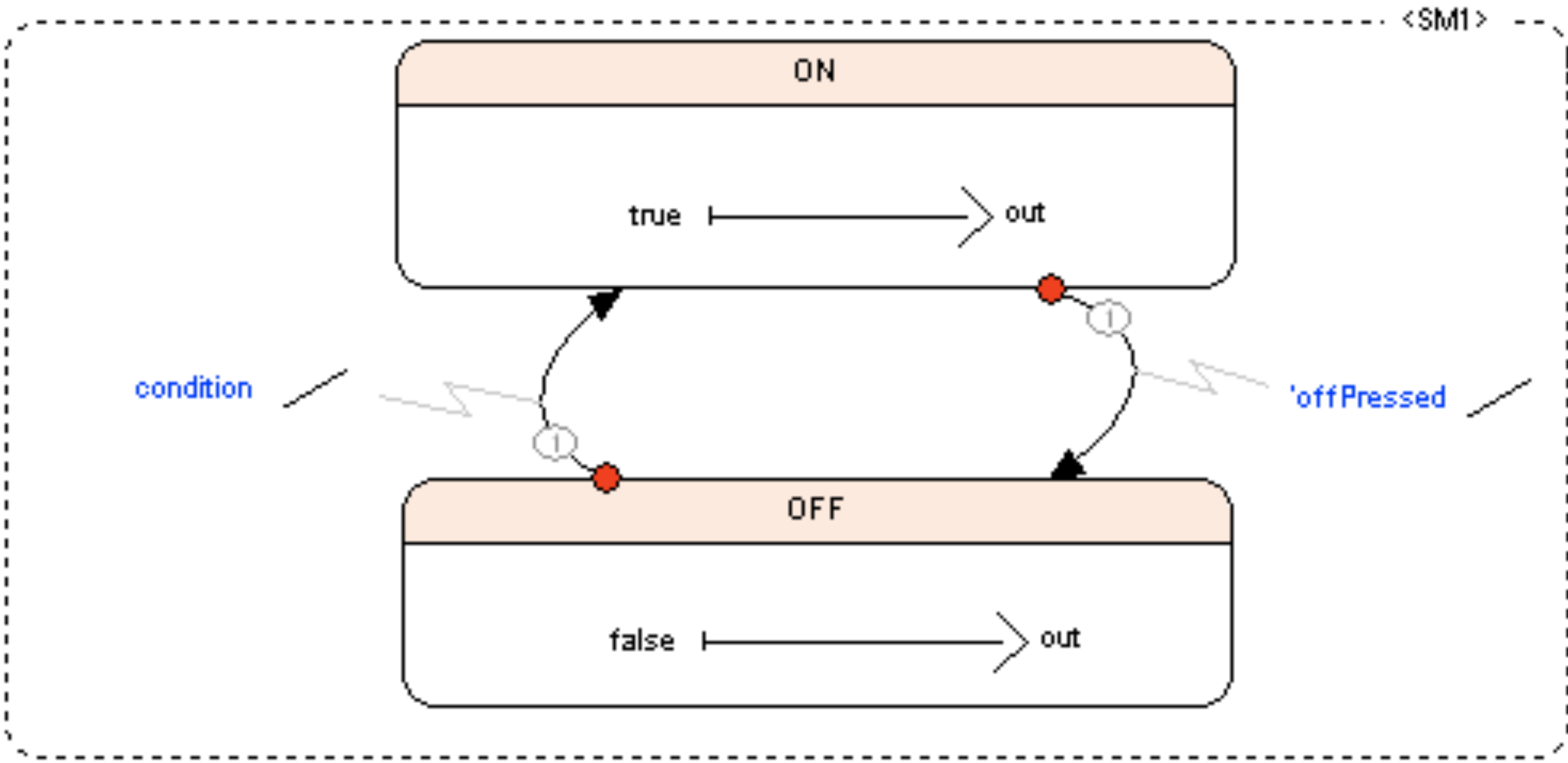
- Semantics of the state machines (summary):
  - At each time there is an active state per state machine
  - Edges allow to change the active state



# Scade - State machines

- Conditions can be added to the edges:
  - Boolean guards
  - signals: `'s` tests that `s` is sent.
- We can send a signal when a transition is taken

# Scade - State machines

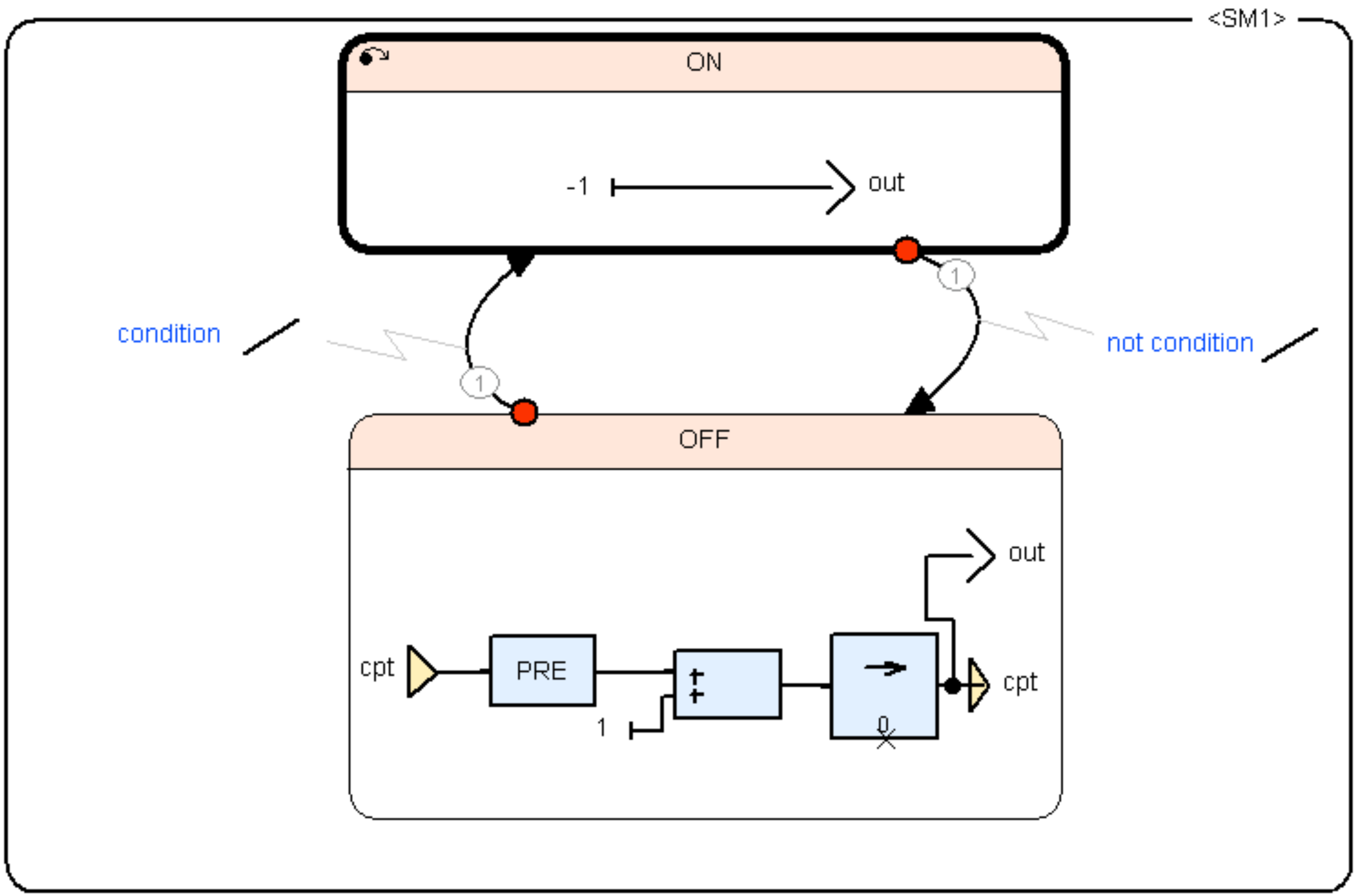


# Scade - State machines

- How to the flows inside a state behave when that state is reached ?
  - Depends on the incoming edge:
    - By default: "*restart*" edge: flows are reset
    - Otherwise: "*resume*" edge: flows keep their last values

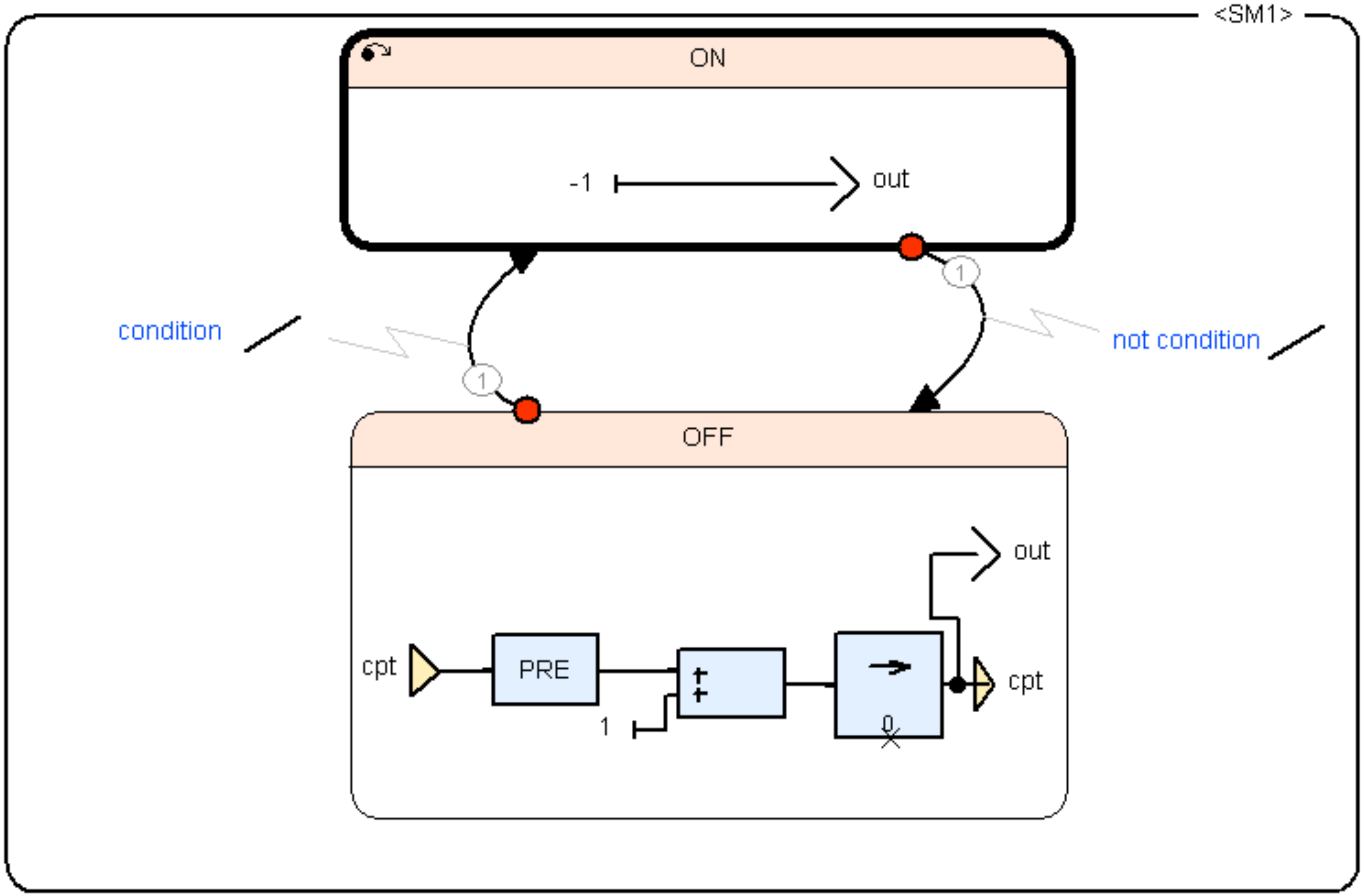
# Scade - State machines

- Example: state with a counter



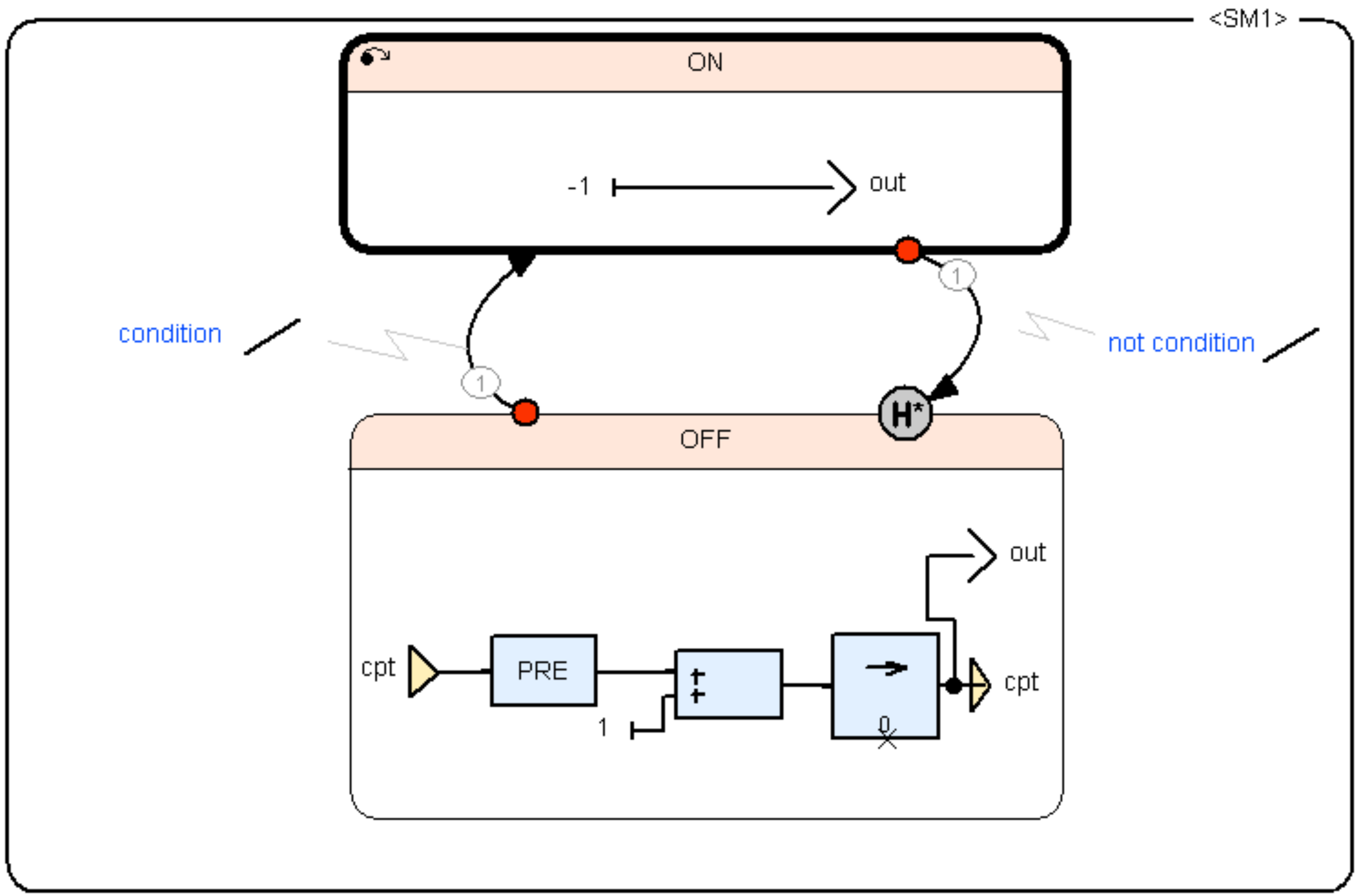
# Scade - State machines

- If the transition entering OFF is a *restart*, the counter is reset each time we enter the state



# Scade - State Machines

- If the transition entering OFF is a *resume*, the counter will keep its last value



# Scade - State Machines

- Démo... (fichier "DiffRestartResume")

# Example - gas burner

- Let us assume to the burner takes some delay before actually switching on, and stays in a transitory «ignition» state before moving to ON

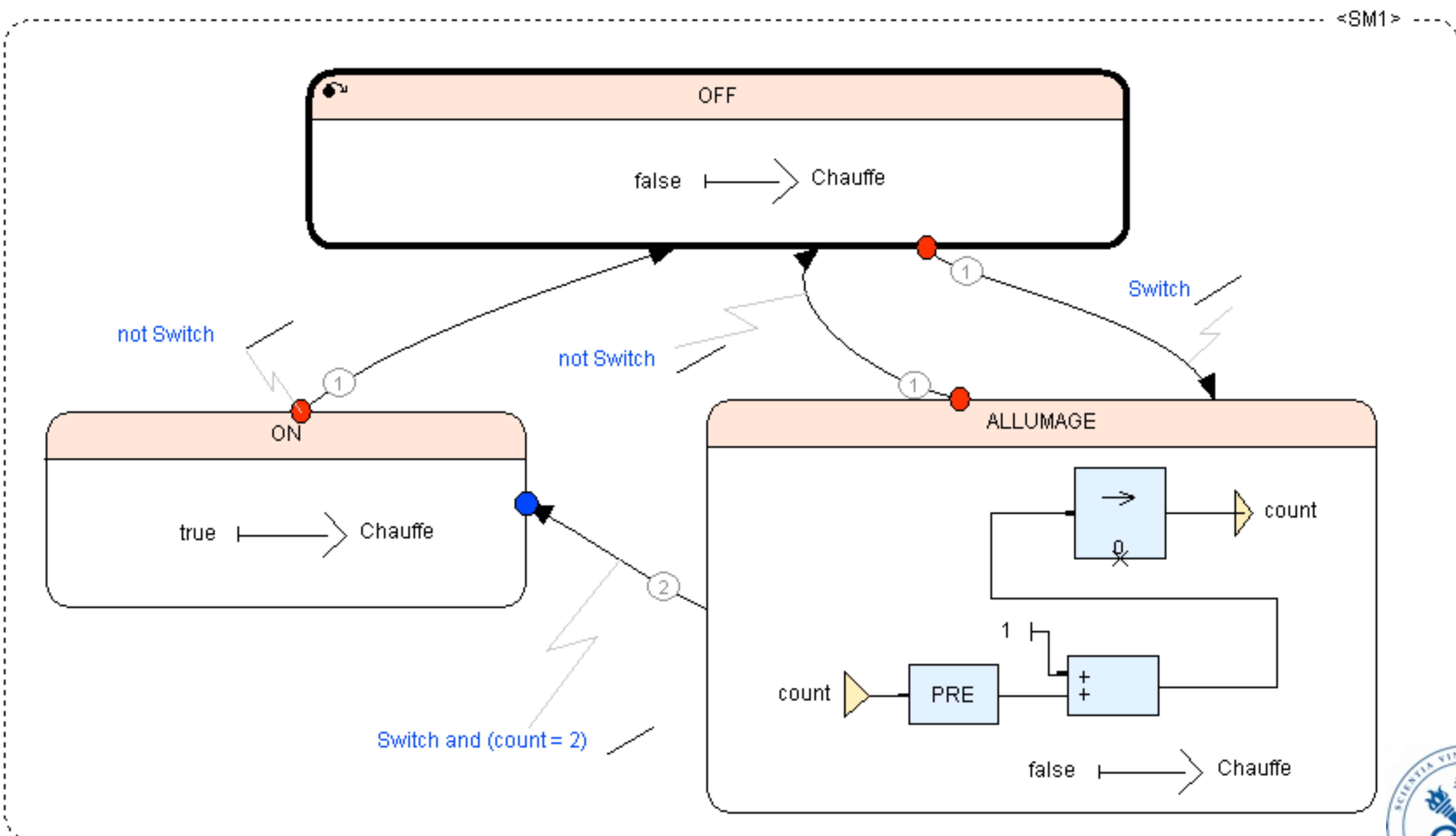
# Example - gas burner

- We can use a 3 states machine:
  - OFF (initial)
  - ignition: the burner does not heat. This state is left after a fixed number of cycles and the machine moves to:
  - ON: burner heats the tank

# Example - gas burner

- In states ON and OFF, we set the output flow to the right value (`true/false`)
- In the transitory state «ignition», a counter, allows to jump to ON after a given number of cycles, to simulate ignition time.

# Example - gas burner



# Example - gas burner

- As expected, this delay introduces a flaw in the control:
  - One cannot guarantee anymore that the temperature stays $\geq$ 50 °c if one switches the burner on at 50 °c
  - During the delay the temperature falls under 50°c

# Example - gas burner

- The verification step allows to detect this problem, and creates a scenario (execution trace).
- Demo (file bruleur-state.avi)

# Example - gas burner

- The controller is modified to switch on the burner when the temperature reaches 53°c
    - Remarks: the initial value of the counter is 0, we thus need three cycles to reach value 2.
- With this new control policy, SCADE can prove the property.

# Example - gas burner

- Demo.... (fichier “cours”)

# Scade - more...

- Advanced datatypes can be defined
    - struct-like datatypes
    - enum-like datatypes
- Scade also provides several libraries of operators
- ...