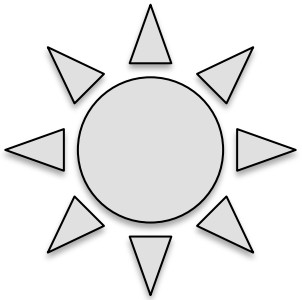
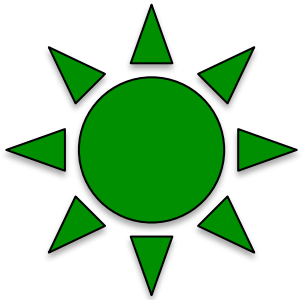
Extended Modelling Notations
&
Experiment

(Modal Sequence Diagrams
&
Timed Automata)

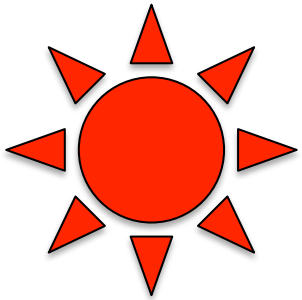
Example: Light bulb



OFF

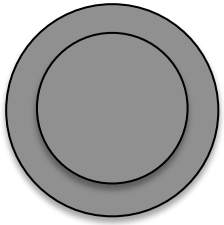


Green & ON

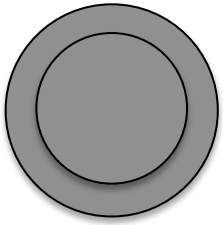


Red & Blinking

Triggered by a user pressing a light button

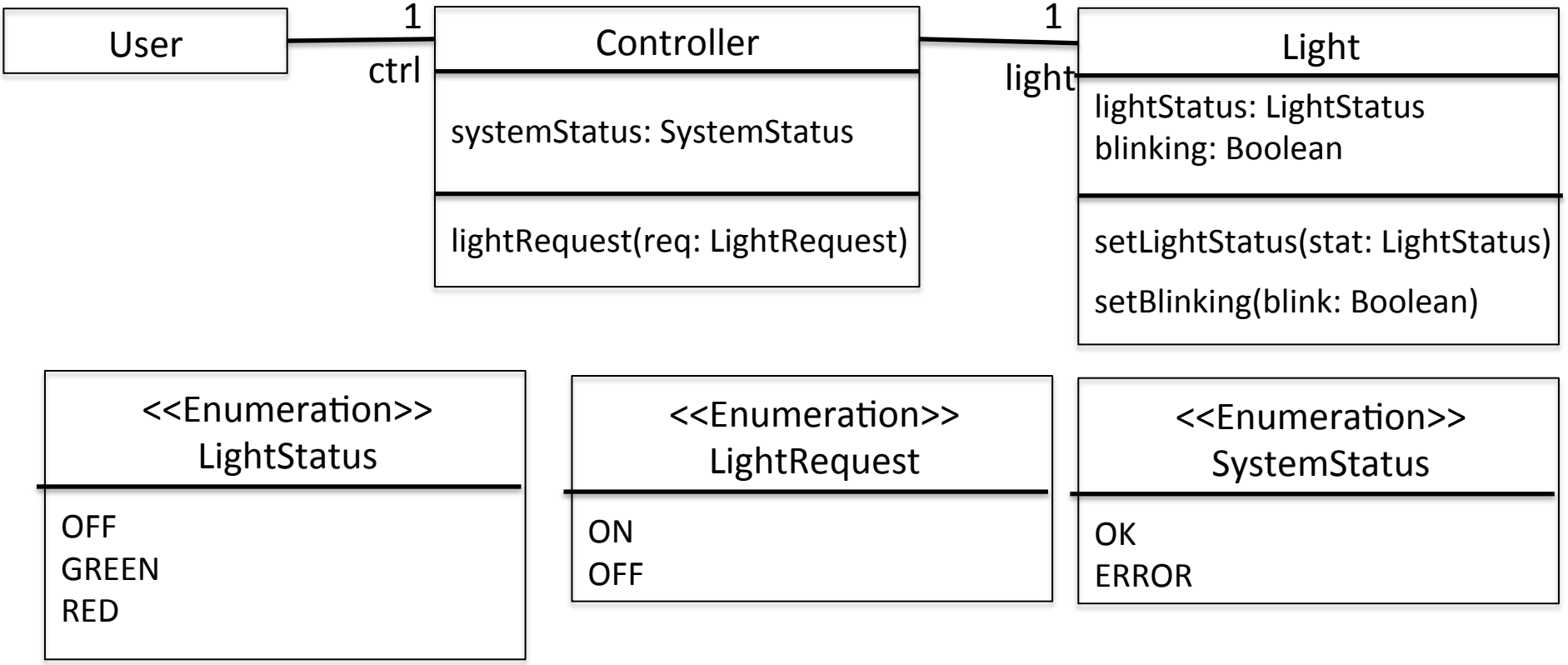


On button



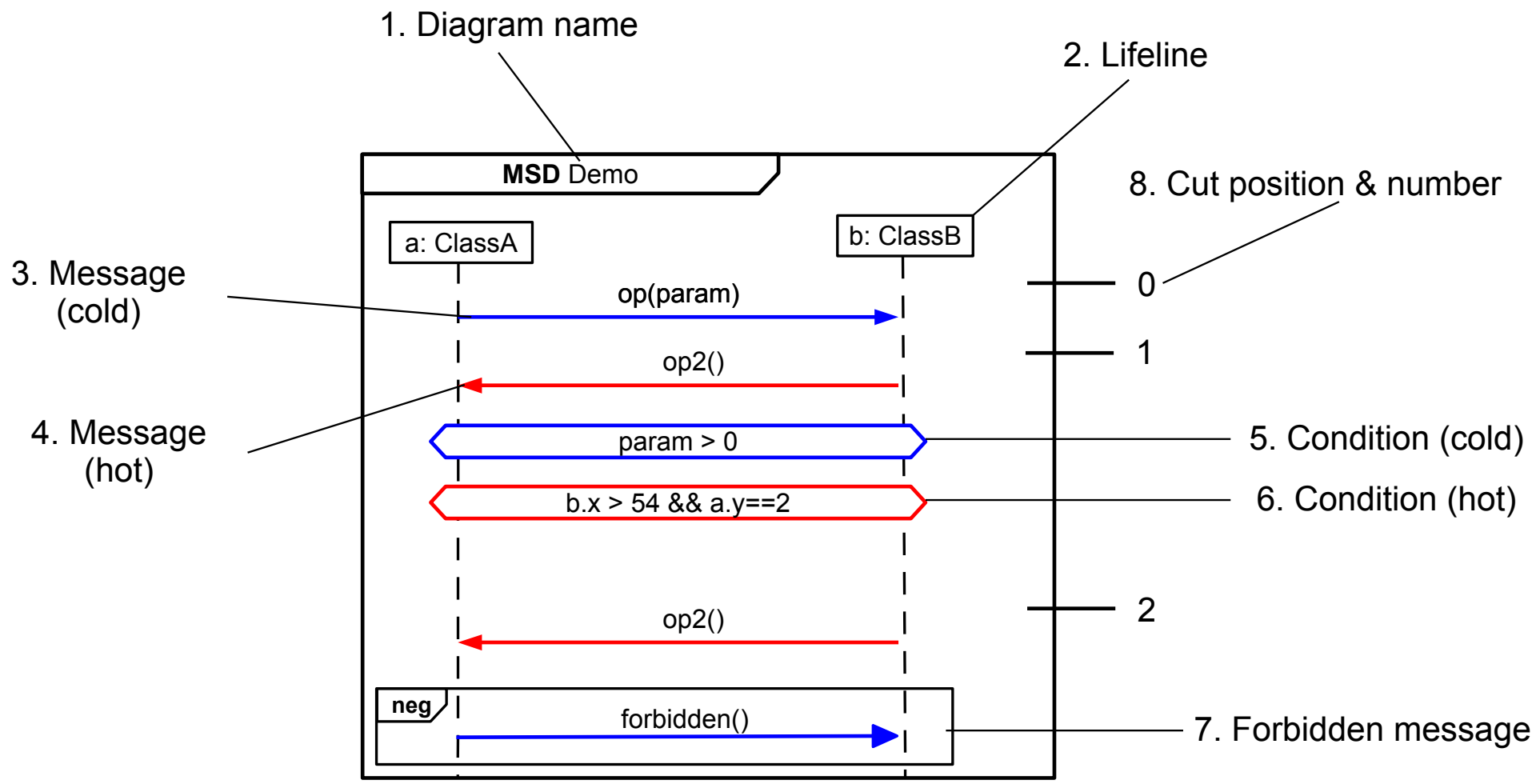
Off button

Class Diagram:

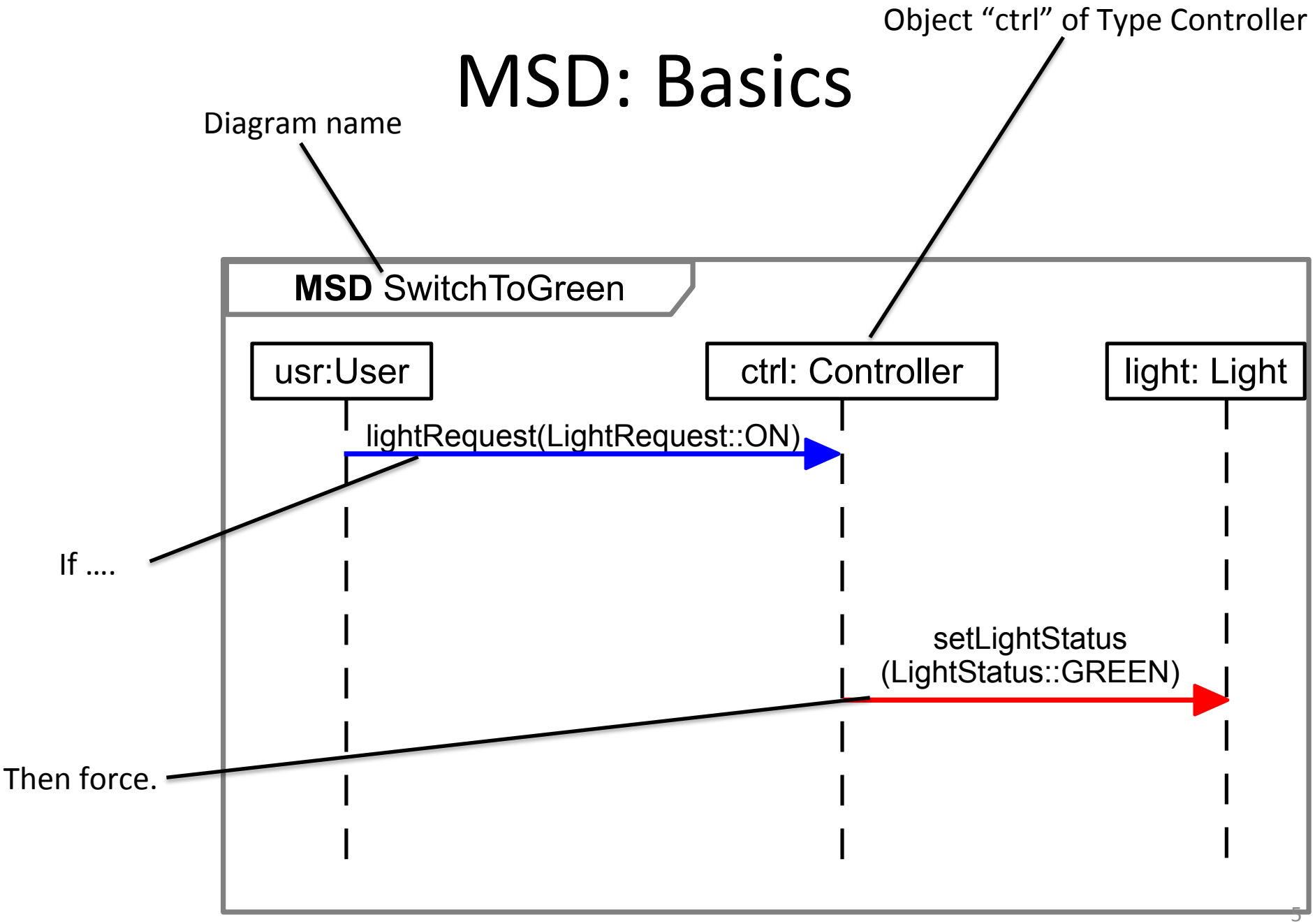


Object Diagram:

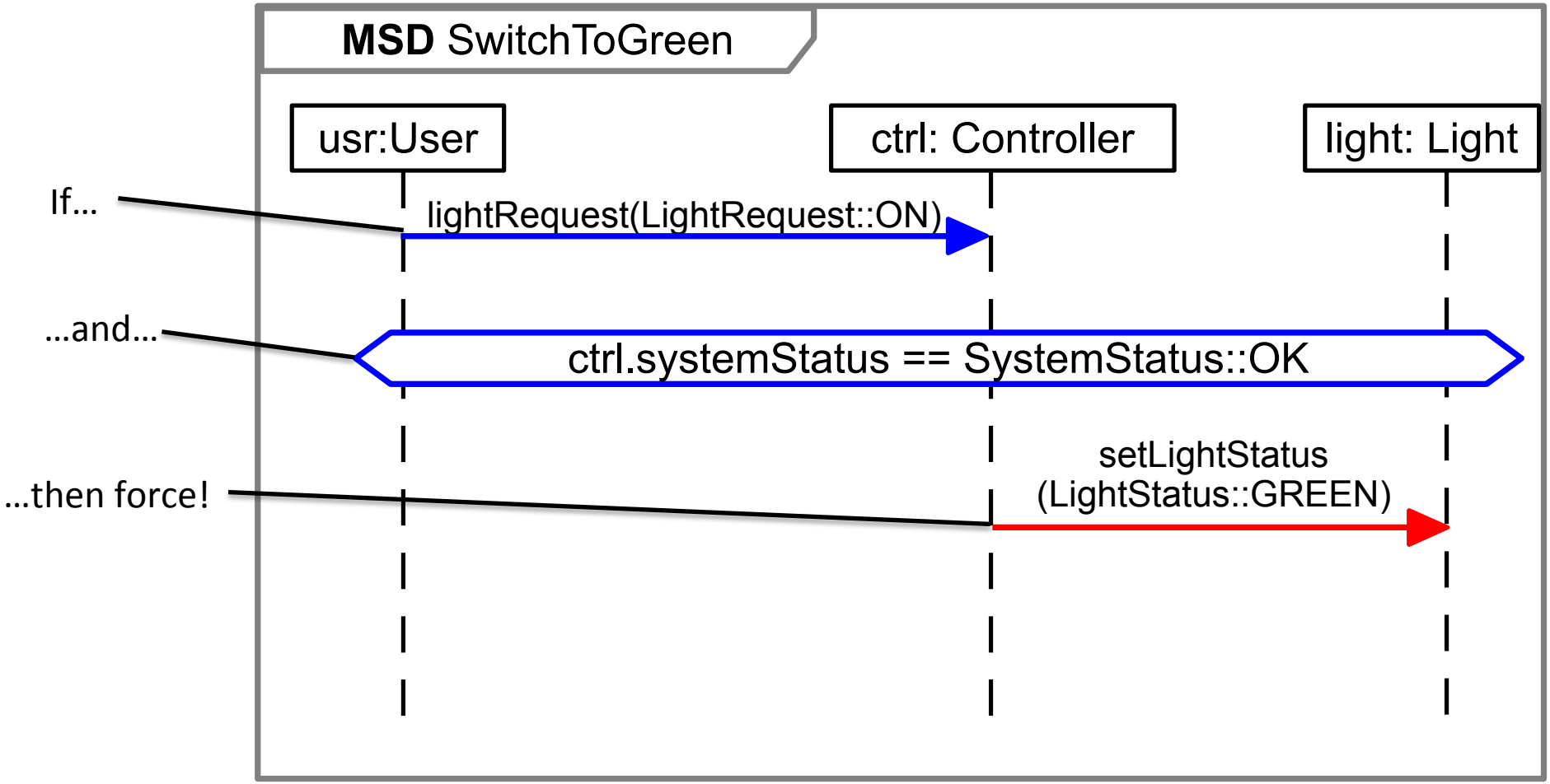
Modal Sequence Diagrams (MSDs)



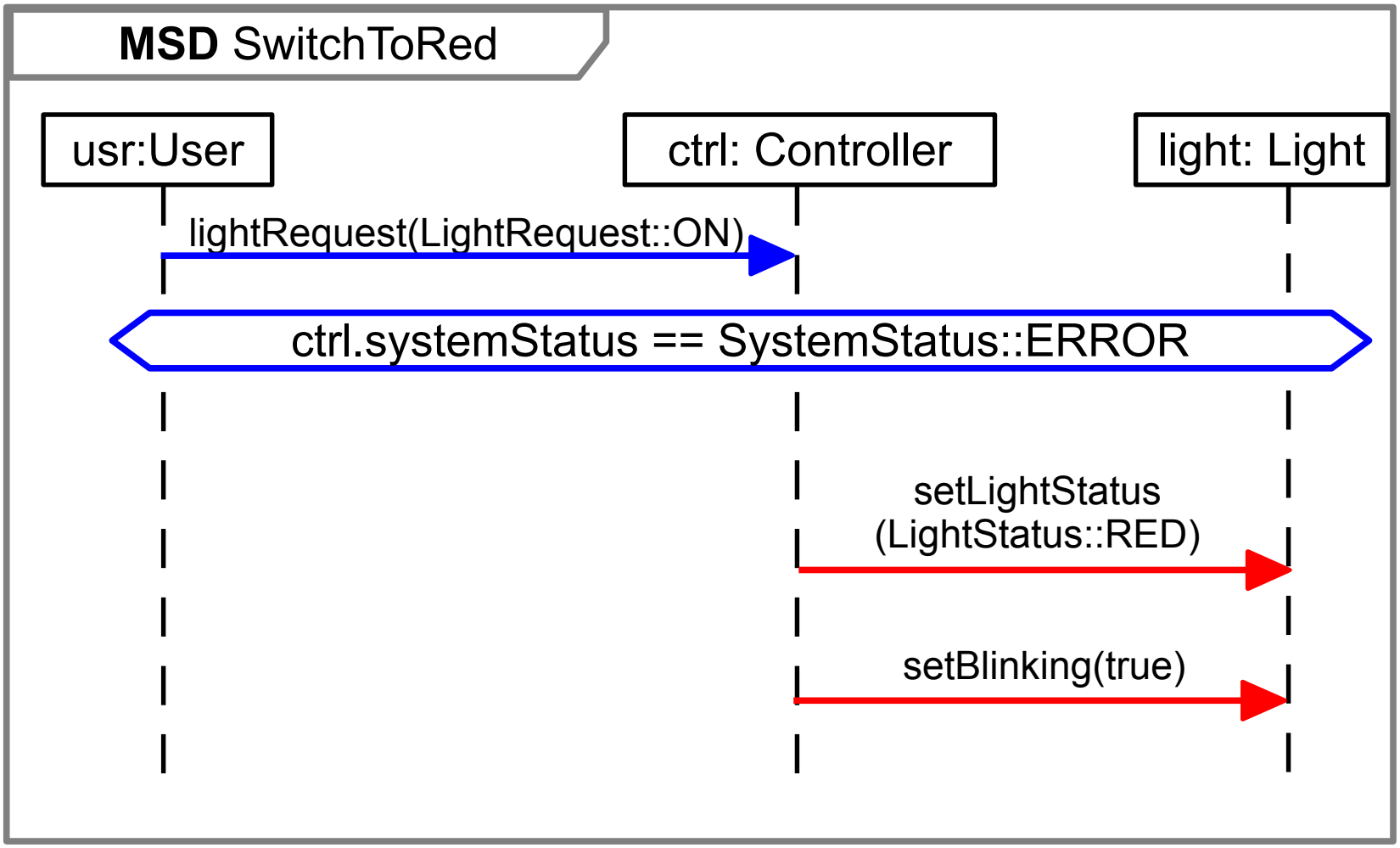
MSD: Basics


Object “ctrl” of Type Controller
Diagram name
MSD SwitchToGreen
usr:User
ctrl: Controller
light: Light
lightRequest(LightRequest::ON)
If ….
setLightStatus
(LightStatus::GREEN)
Then force.

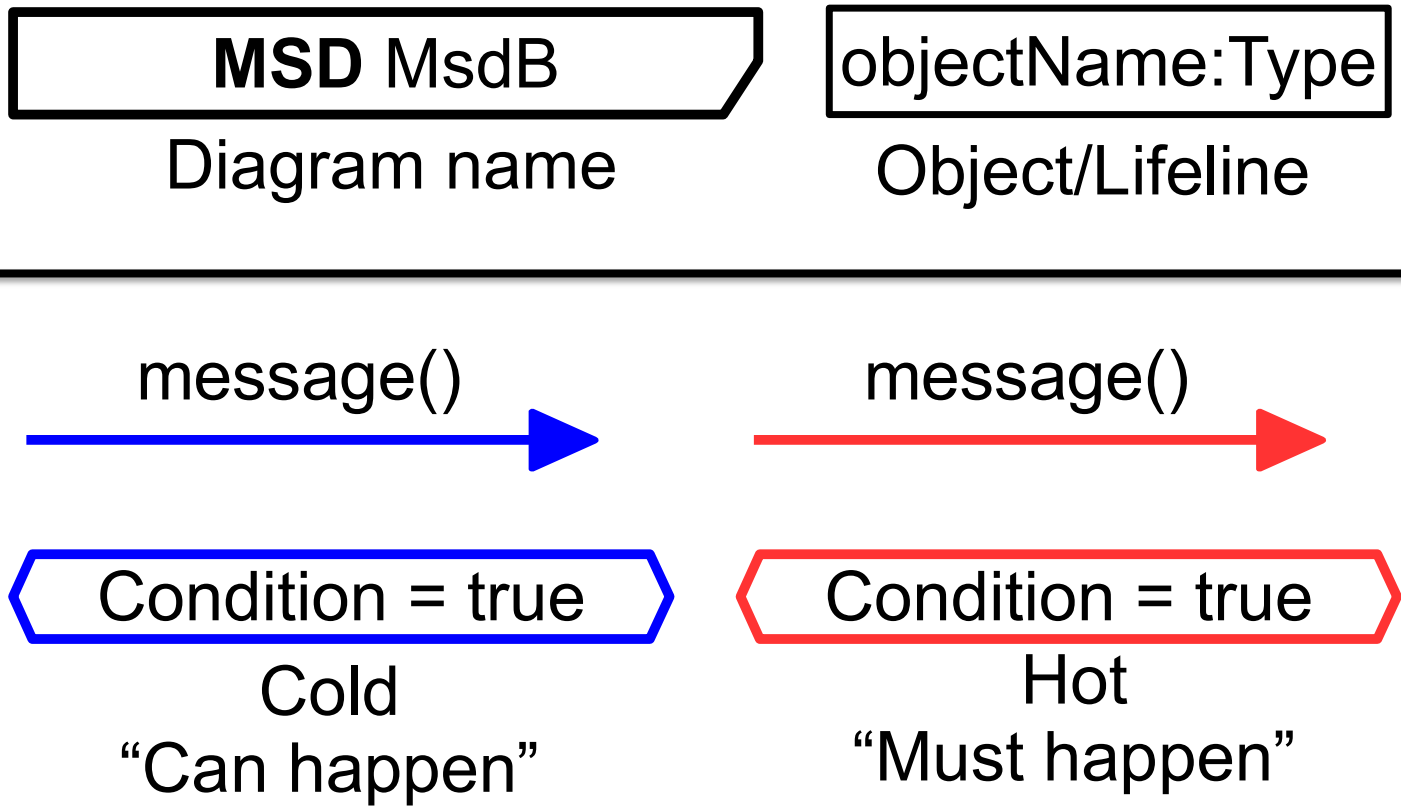
MSD: Conditions



Example


MSD SwitchToRed
usr:User
ctrl: Controller
light: Light
lightRequest(LightRequest::ON)
ctrl.systemStatus == SystemStatus::ERROR
setLightStatus
(LightStatus::RED)
setBlinking(true)

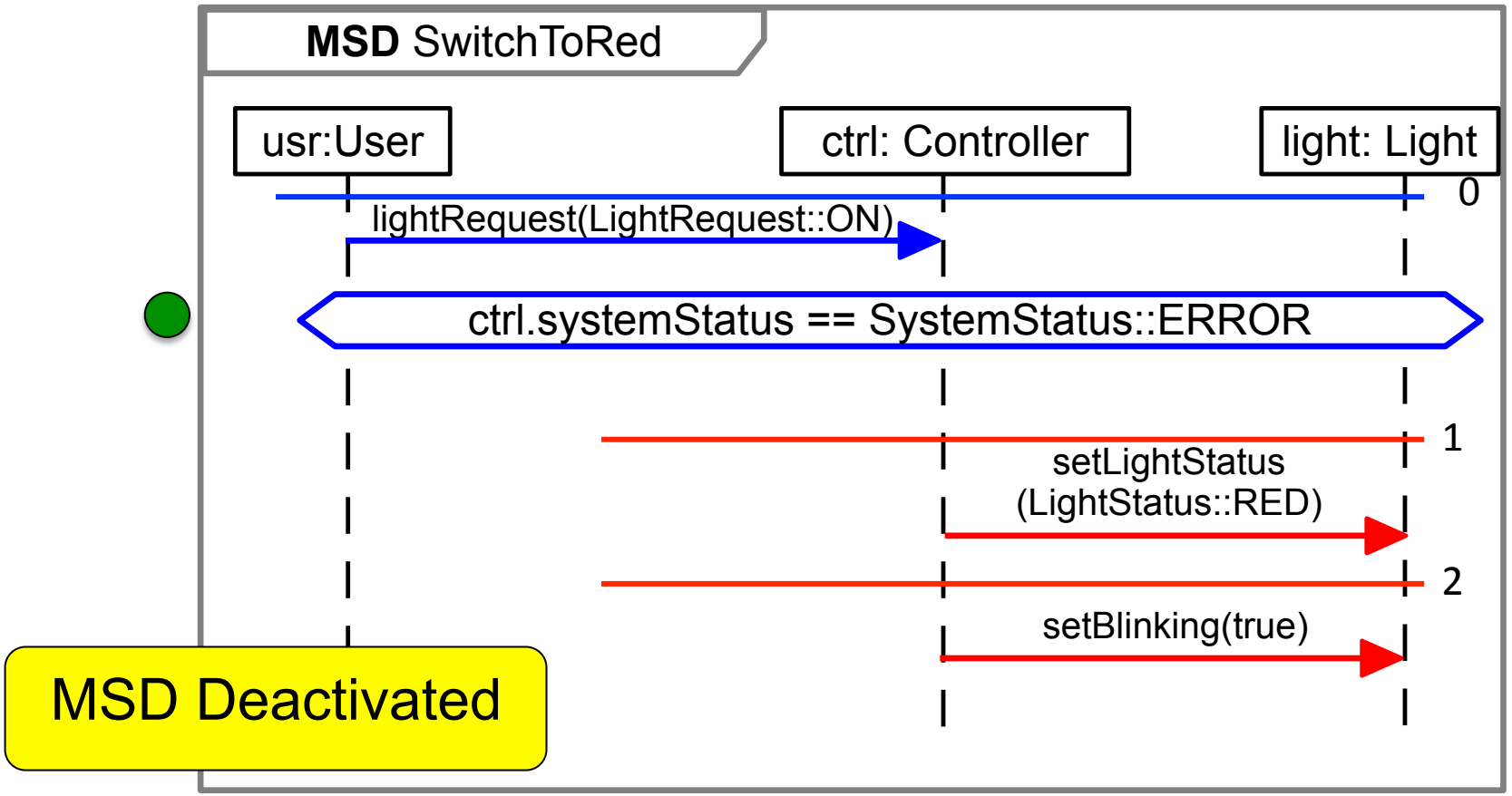
Hot/Cold


MSD MsdB
Diagram name
objectName:Type
Object/Lifeline
message()
message()
Condition = true
Condition = true
Cold
“Can happen”
Hot
“Must happen”

Execution: The cut

1. usr: lightRequest(LightRequest::ON)
2. ctrl: setLightStatus(LightStatus::RED)
3. ctrl: setBlinking(true)

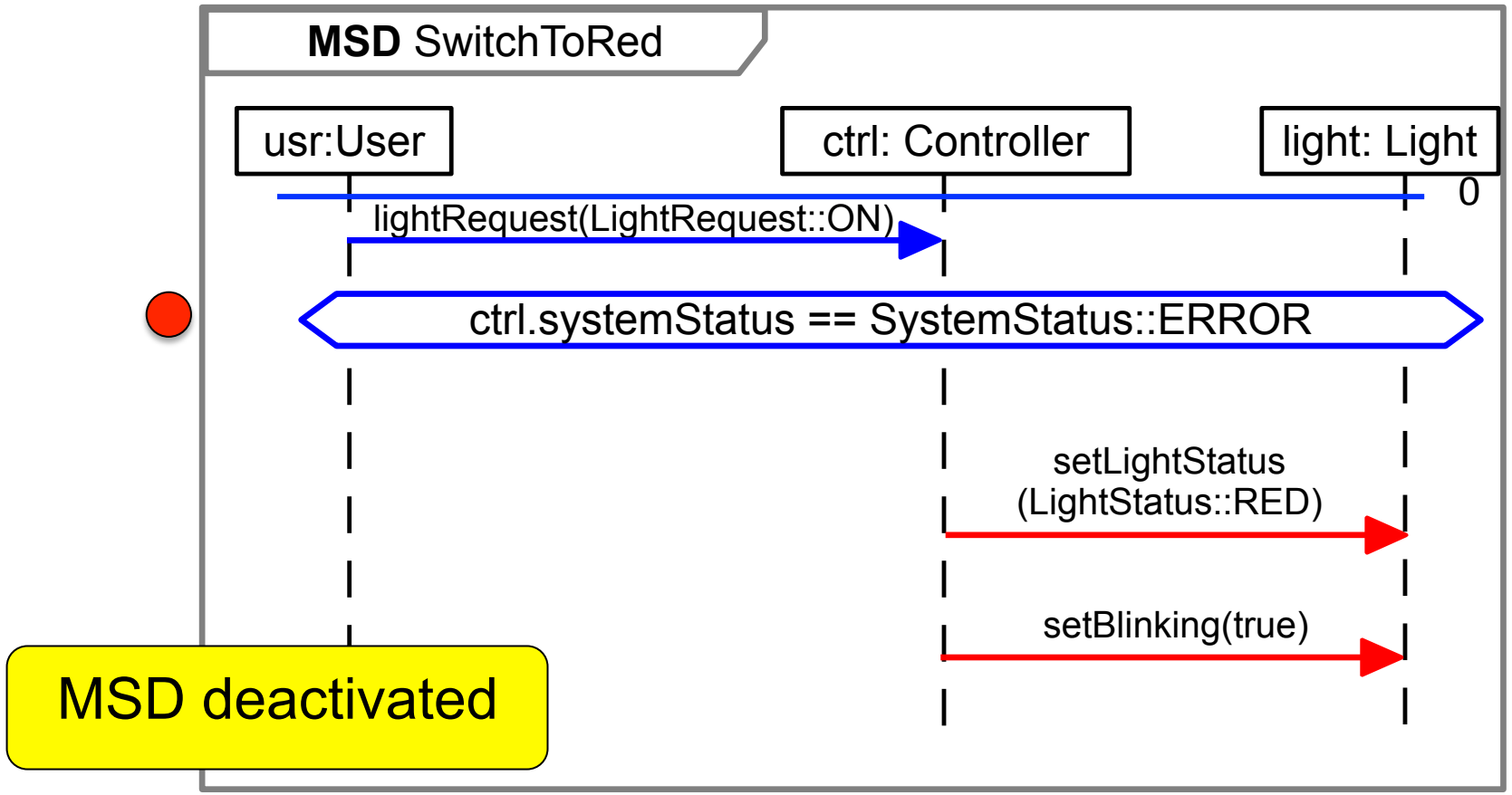
MSD activated



Execution: The cut

1. usr: lightRequest(LightRequest::ON)

MSD activated

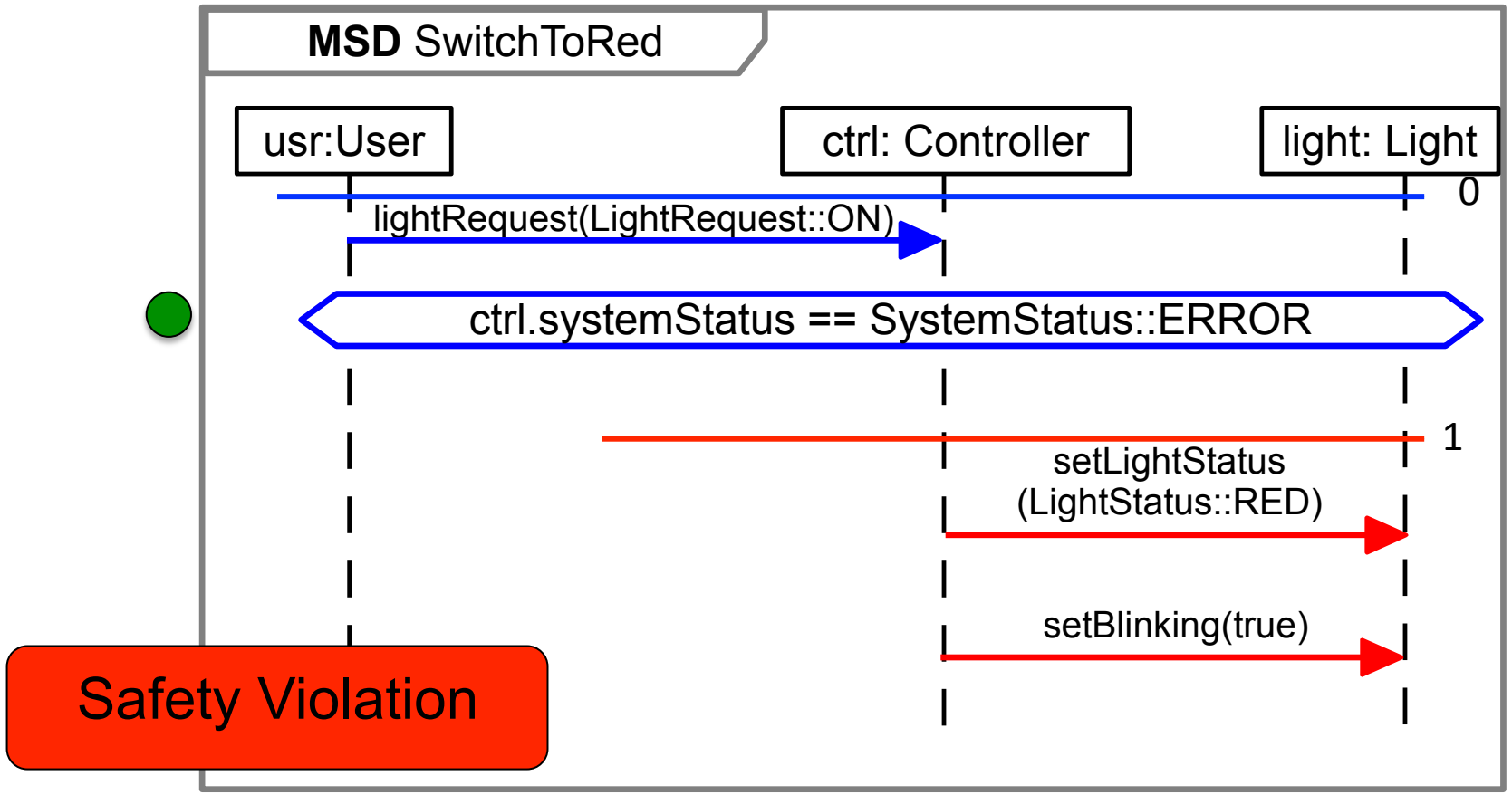


MSD deactivated

Execution: The cut

1. usr: lightRequest(LightRequest::ON)
2. ctrl: setBlinking(true)

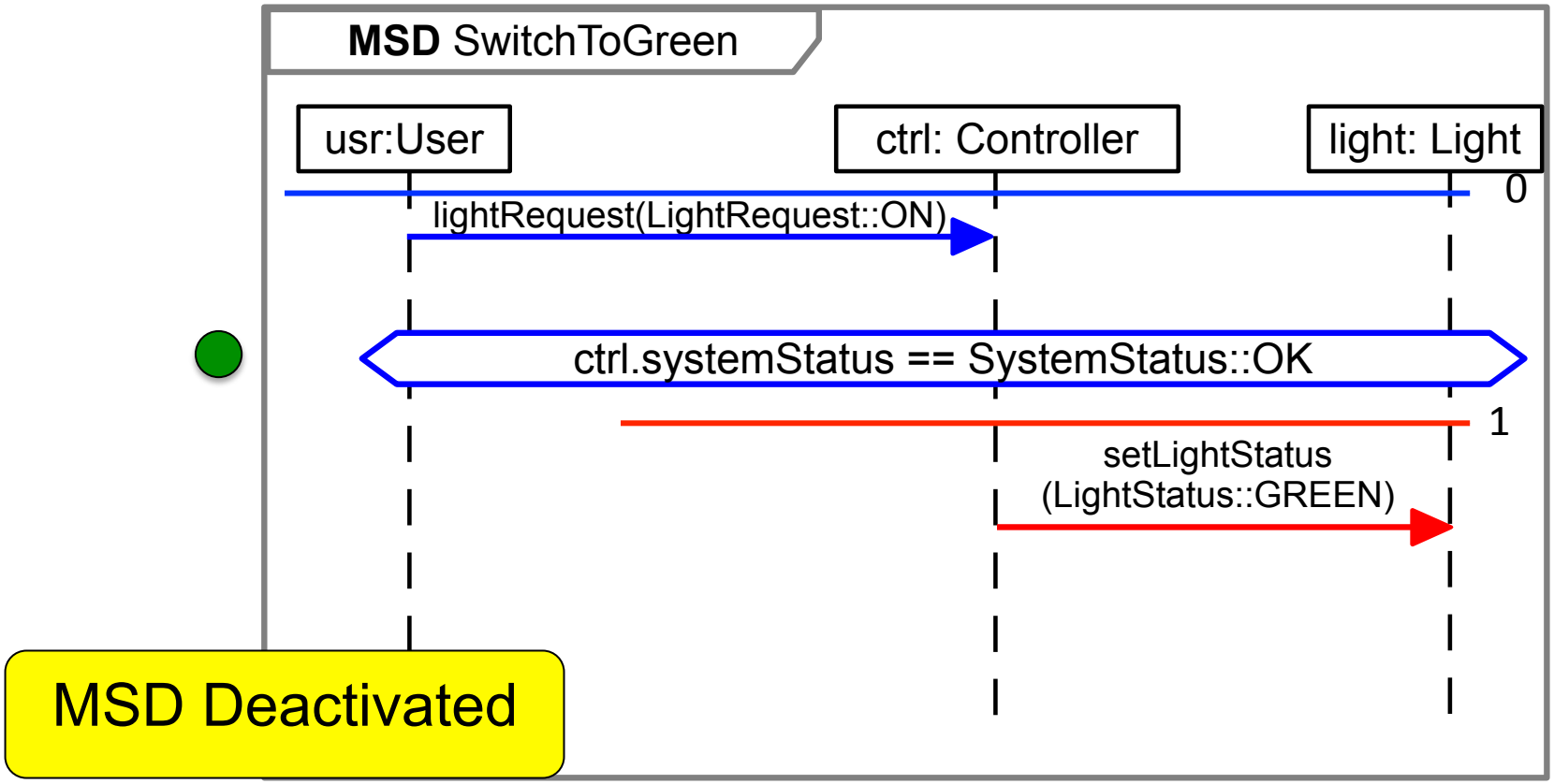
MSD activated

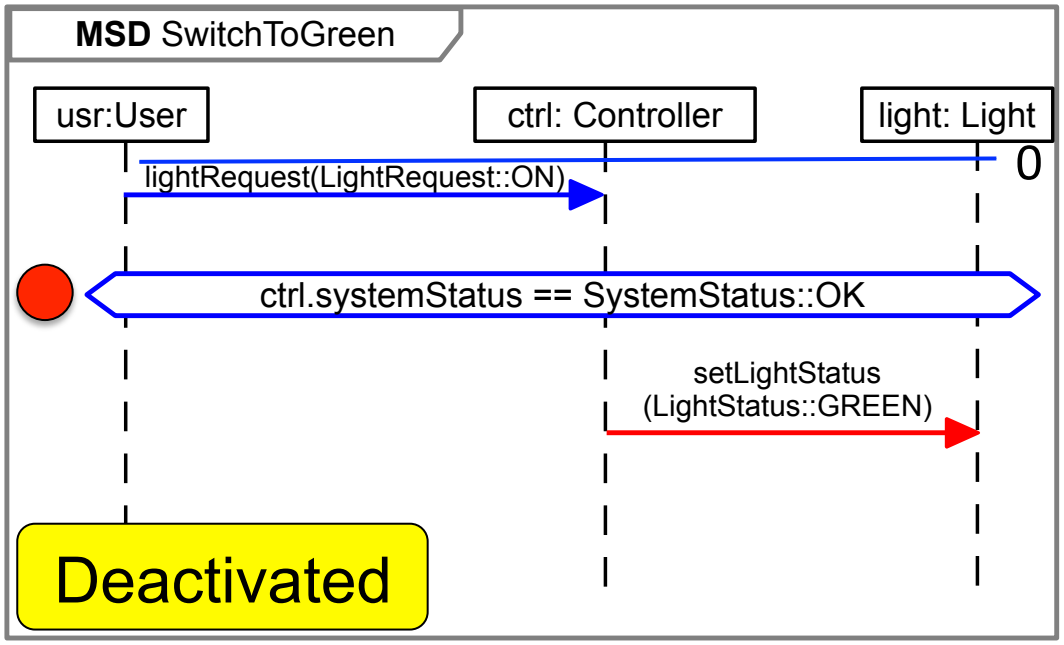


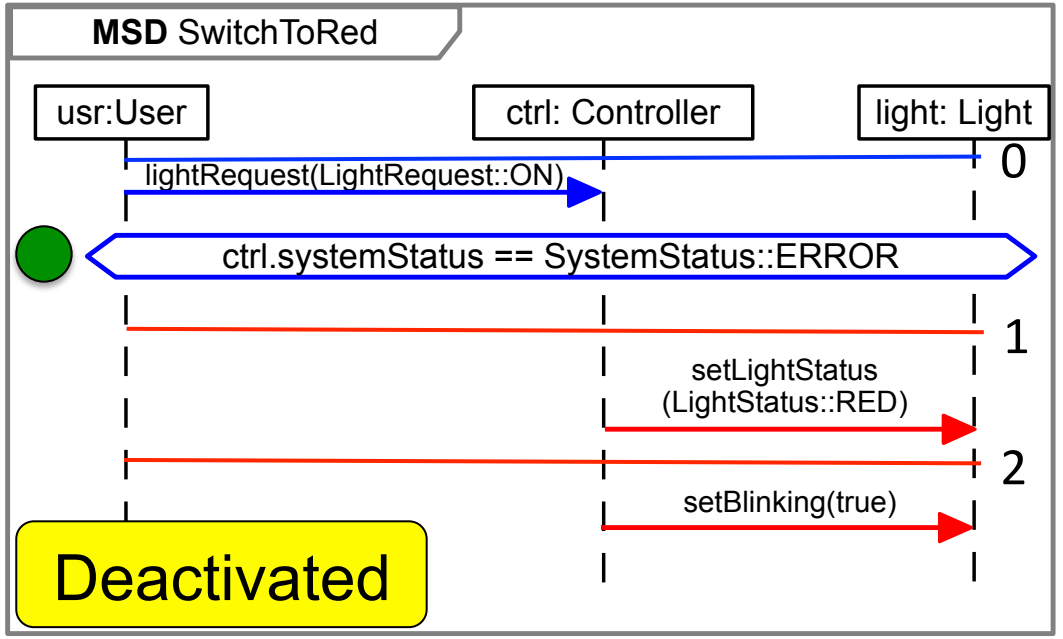
Execution: The cut

1. usr: lightRequest(LightRequest::ON)
2. ctrl: setBlinking(true)
3. ctrl: setLightStatus(LightStatus::GREEN)

MSD activated

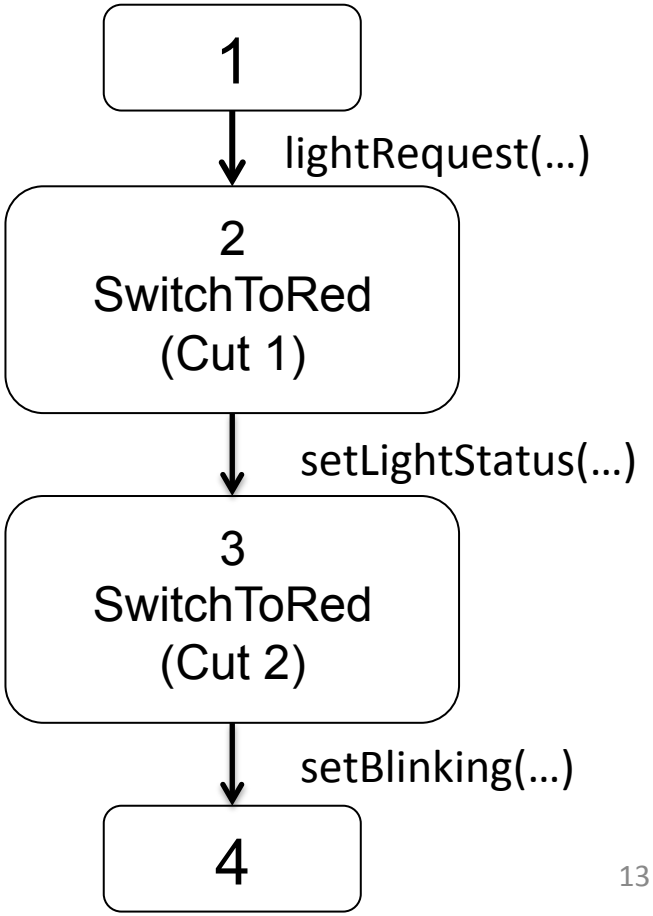






ctrl.systemStatus == SystemStatus.ERROR

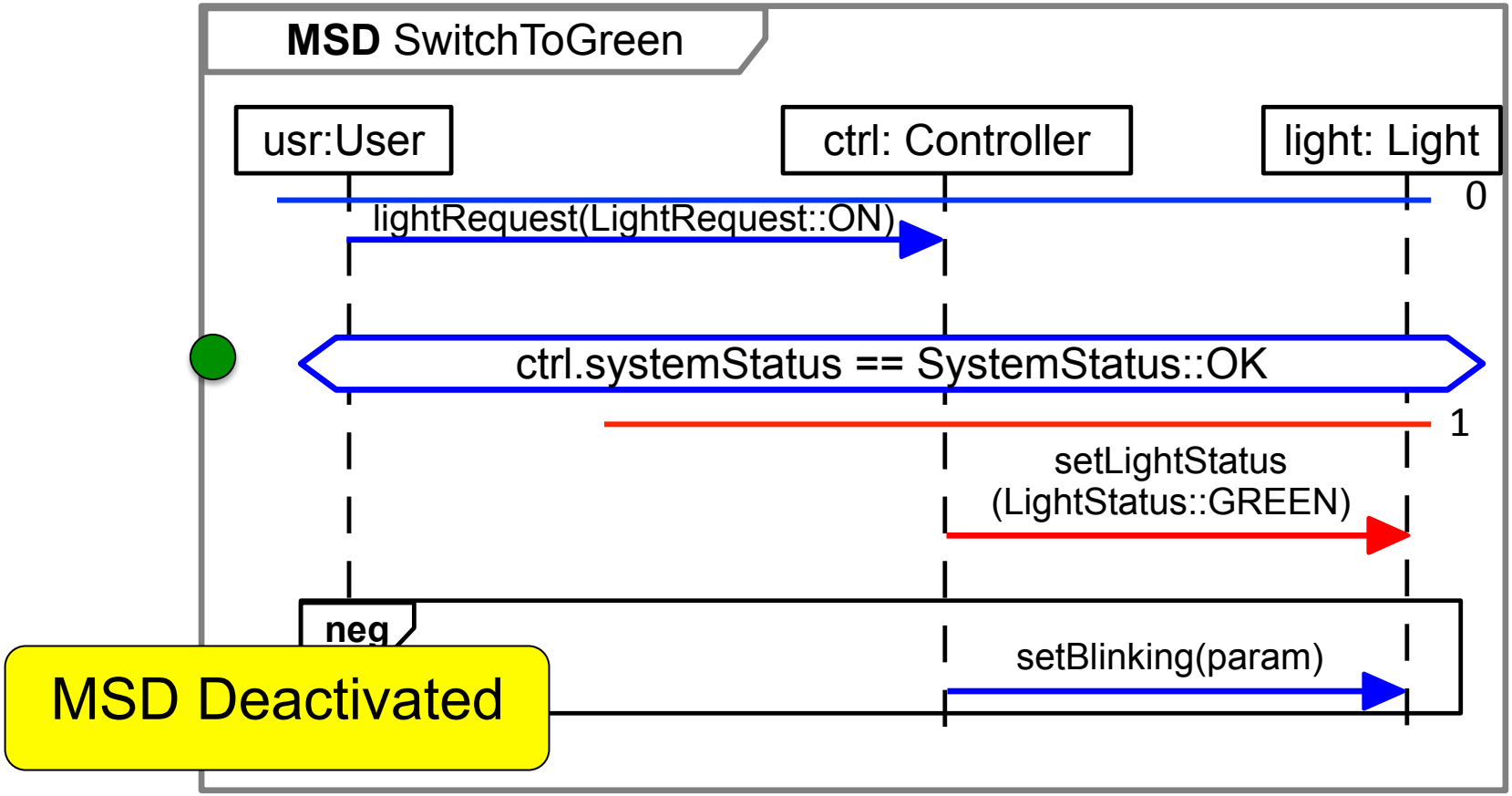
1. usr: lightRequest(LightRequest::ON)
2. ctrl: light.setLightStatus(LightStatus::RED)
3. ctrl: light.setBlinking(true)



Execution: Forbidden messages

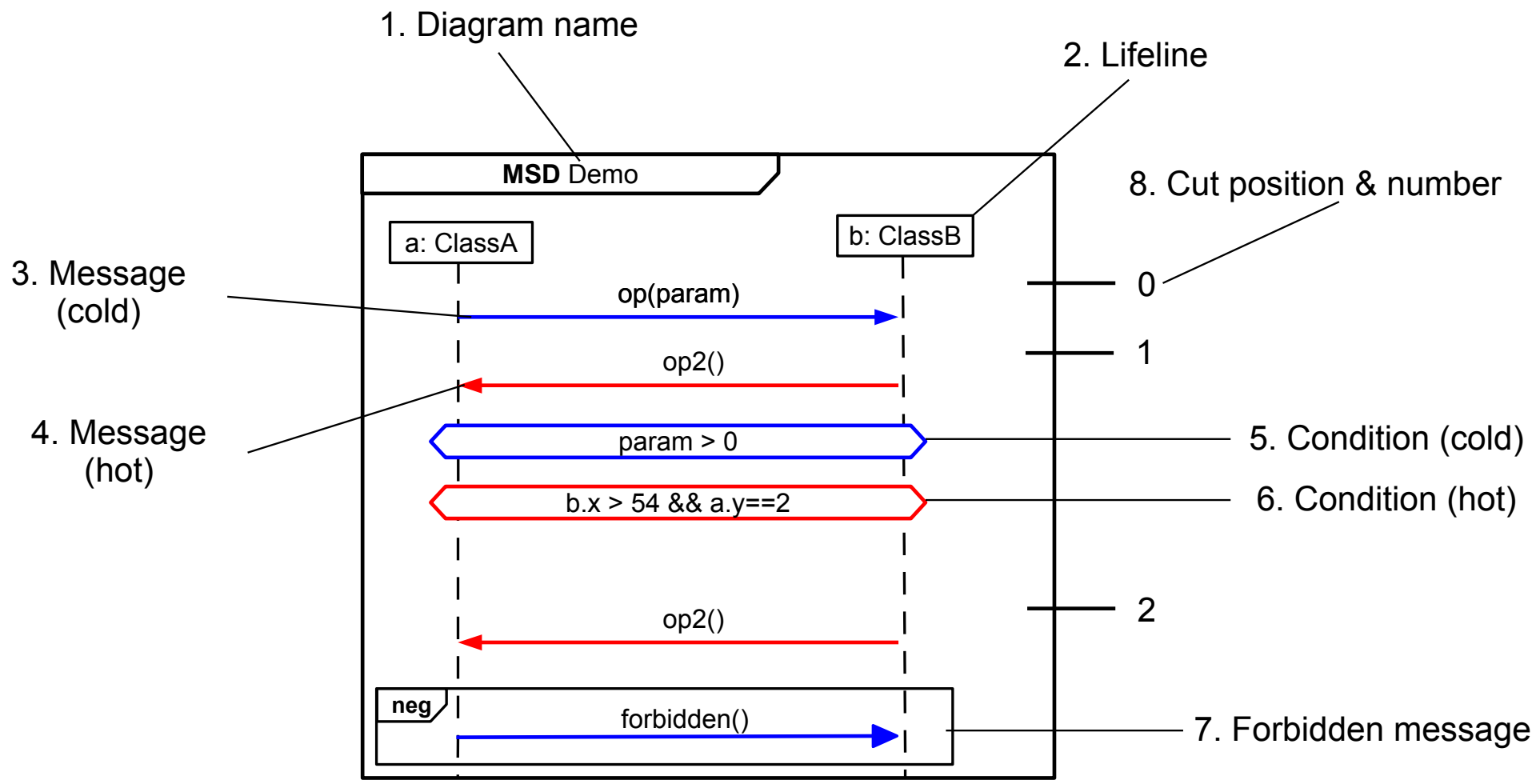
1. usr: lightRequest(LightRequest::ON)
2. ctrl: setBlinking(true)

MSD activated

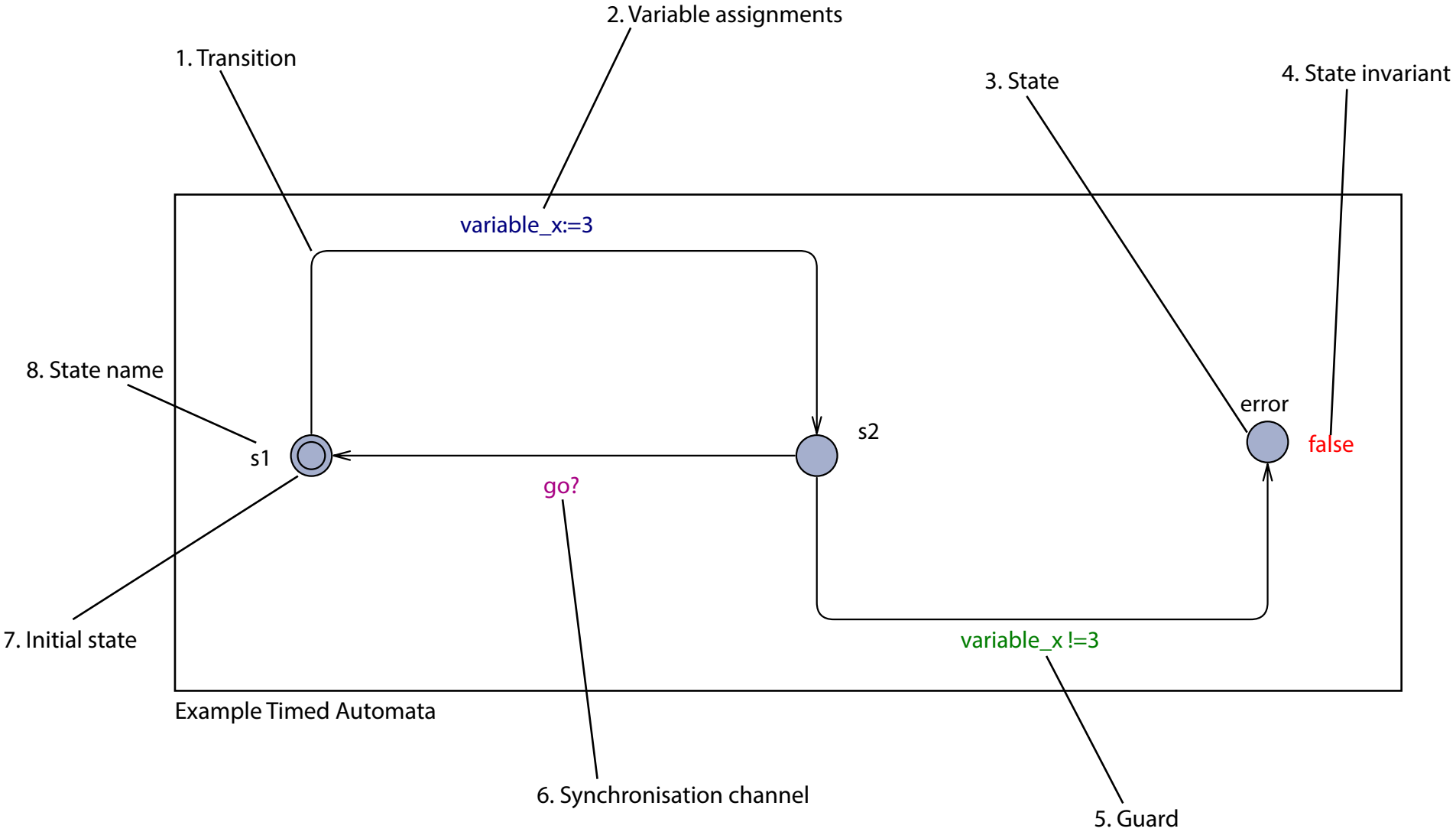


MSD Deactivated

Modal Sequence Diagrams (MSDs)

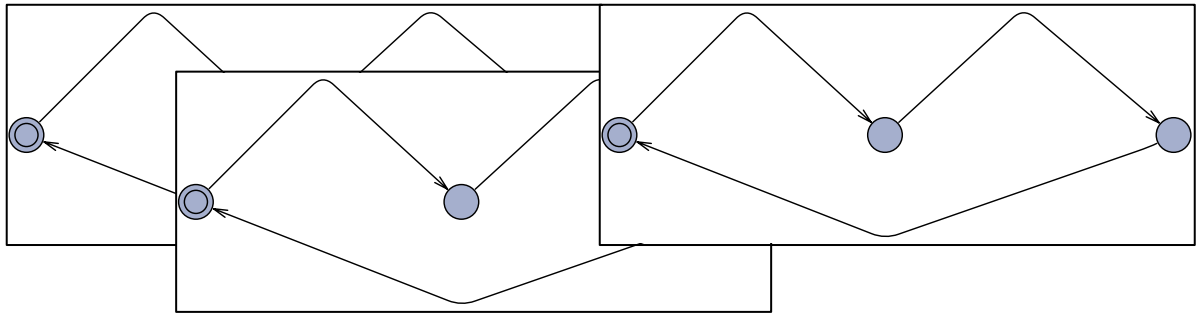


Timed Automata (TA)



TA: Variables

- Timed Automata model consists of:
  - Diagrams



  - Declarations

```
//Enum definitions
typedef int[0,1] SystemStatusValues;
const int SystemStatus_ERROR = 0;
const int SystemStatus_OK = 1;

typedef int[0,1] LightRequestValues;
const int LightRequest_ON = 0;
const int LightRequest_OFF = 1;

typedef int[0,2] LightStatusValues;
const int LightStatus_OFF = 0;
const int LightStatus_GREEN = 1;
const int LightStatus_RED = 2;

const int NO_REQUEST = -1;
//Broadcast Channels to trigger state transitions
broadcast chan LightRequest;
broadcast chan SetLightStatus;
broadcast chan SetBlinking;
```

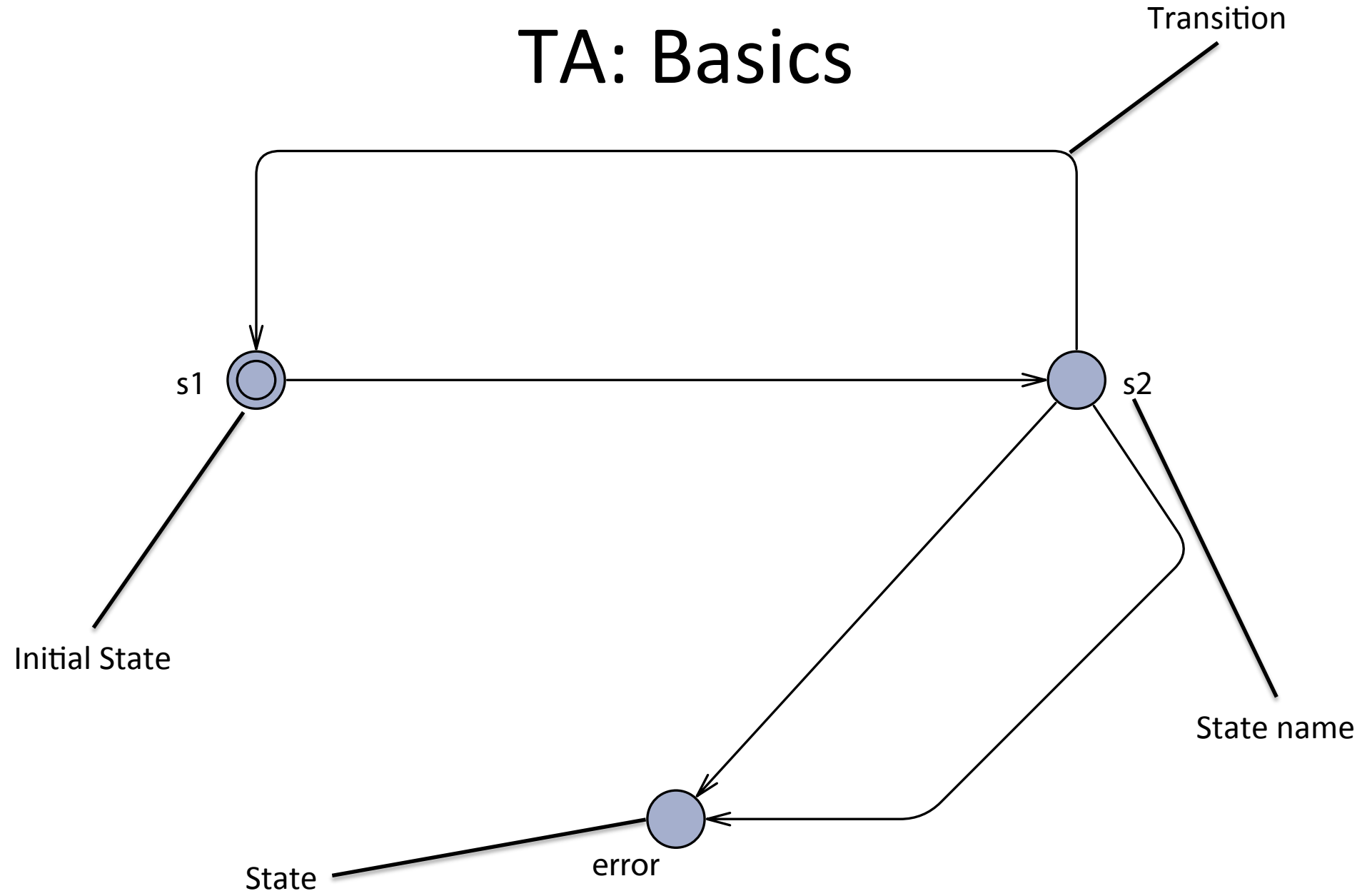
TA: Variables

- C Syntax: primitive datatypes (& functions)

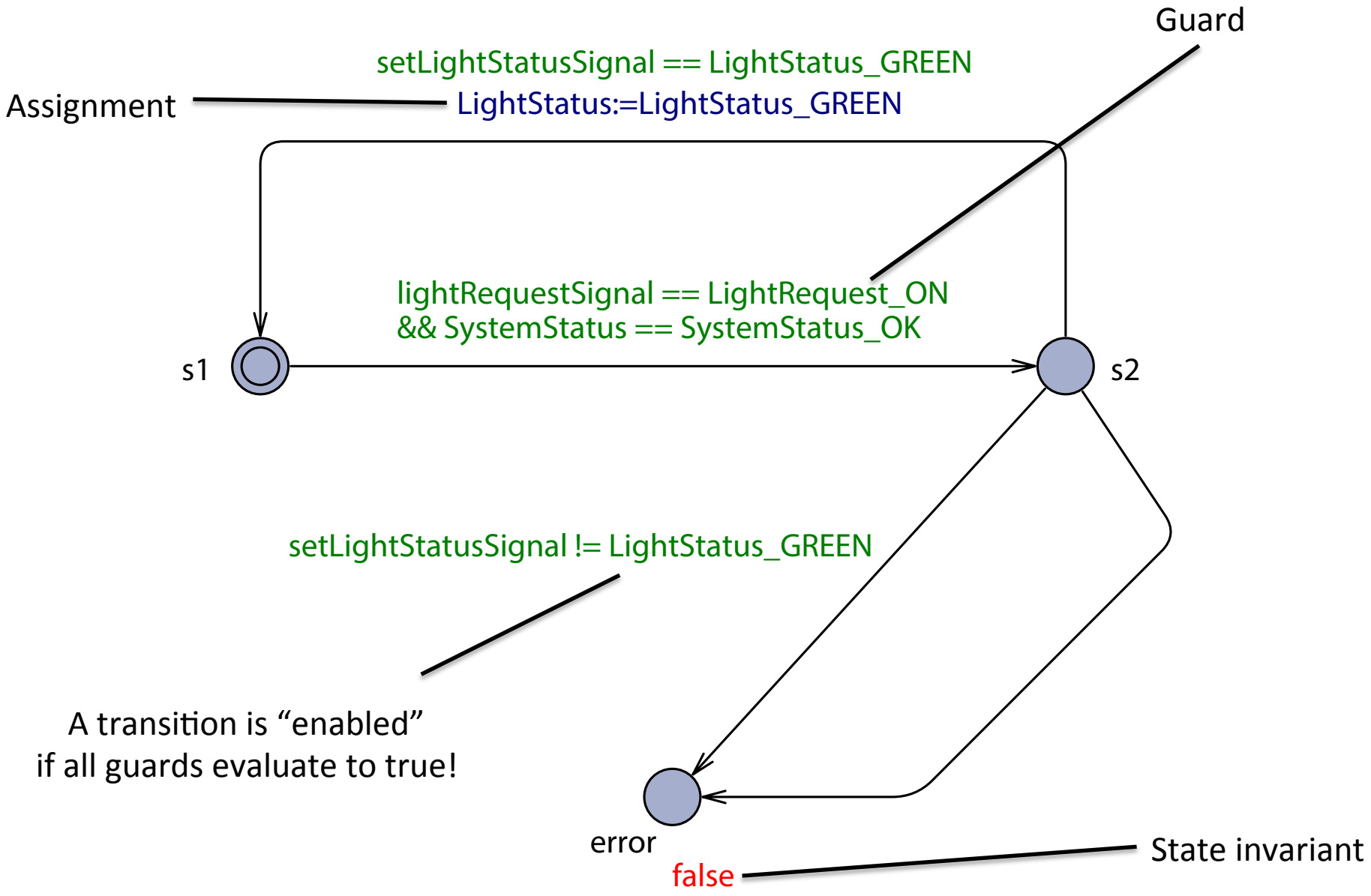
```
//Enum definitions
typedef int[0,1] SystemStatusValues;
const int SystemStatus_ERROR = 0;
const int SystemStatus_OK = 1;

typedef int[0,1] LightRequestValues;
const int LightRequest_ON = 0;
const int LightRequest_OFF = 1;

typedef int[0,2] LightStatusValues;
const int LightStatus_OFF = 0;
const int LightStatus_GREEN = 1;
const int LightStatus_RED = 2;

const int NO_REQUEST = -1;
```

```
//Broadcast Channels to trigger state transitions
broadcast chan LightRequest;
broadcast chan SetLightStatus;
broadcast chan SetBlinking;

//Signal values
int lightRequestSignal = NO_REQUEST;
int setLightStatusSignal = NO_REQUEST;
int setBlinkingSignal = NO_REQUEST;

//System variables
SystemStatusValues        SystemStatus;
LightStatusValues         LightStatus;
bool                      blinking;
```

TA: Basics

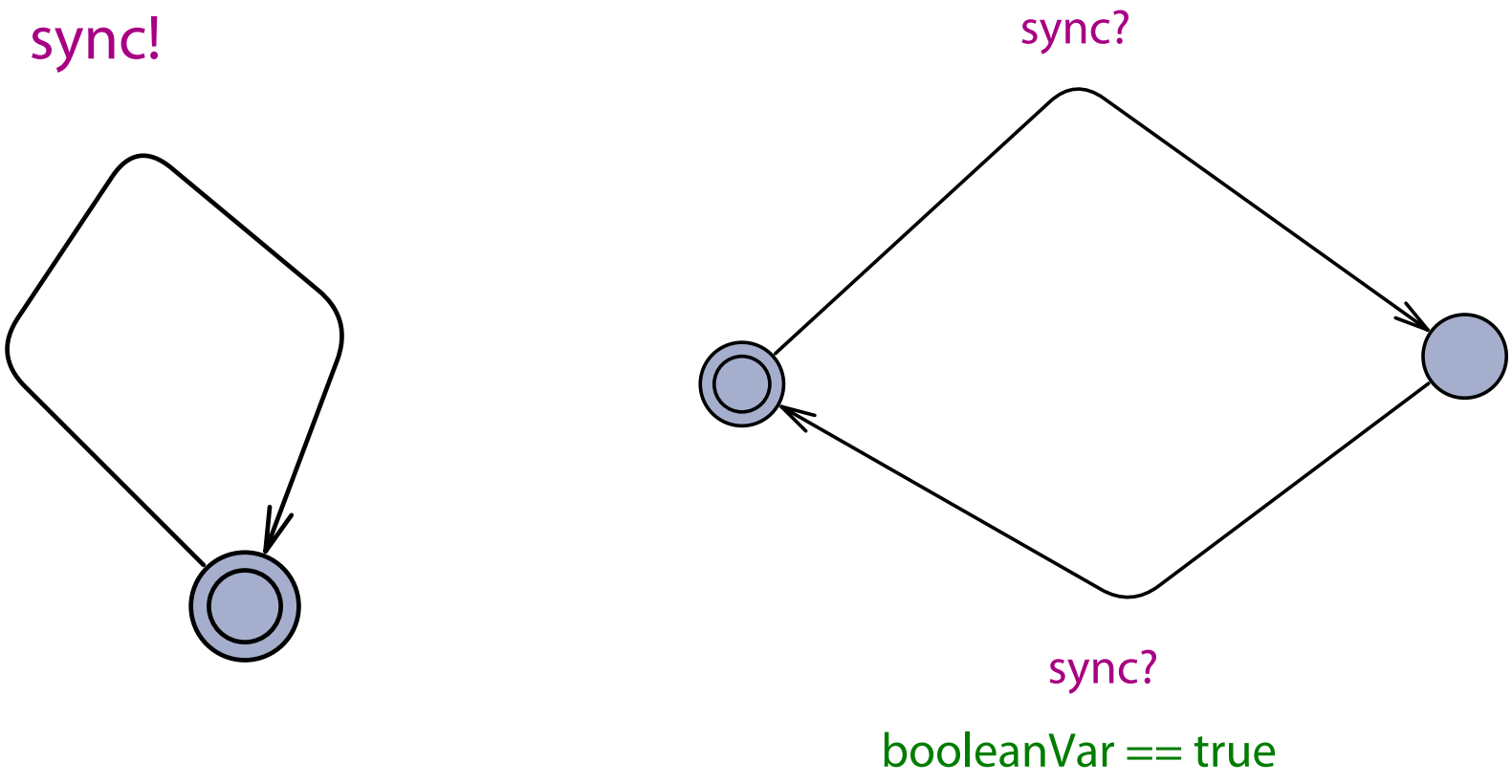


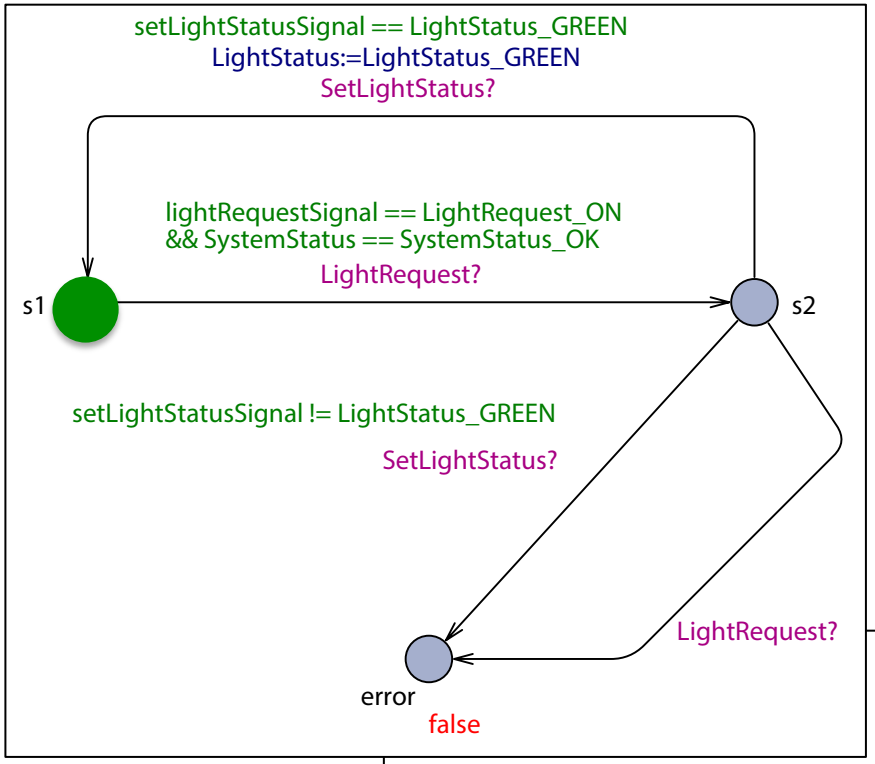
TA: Transitions & Guards



TA: Channels

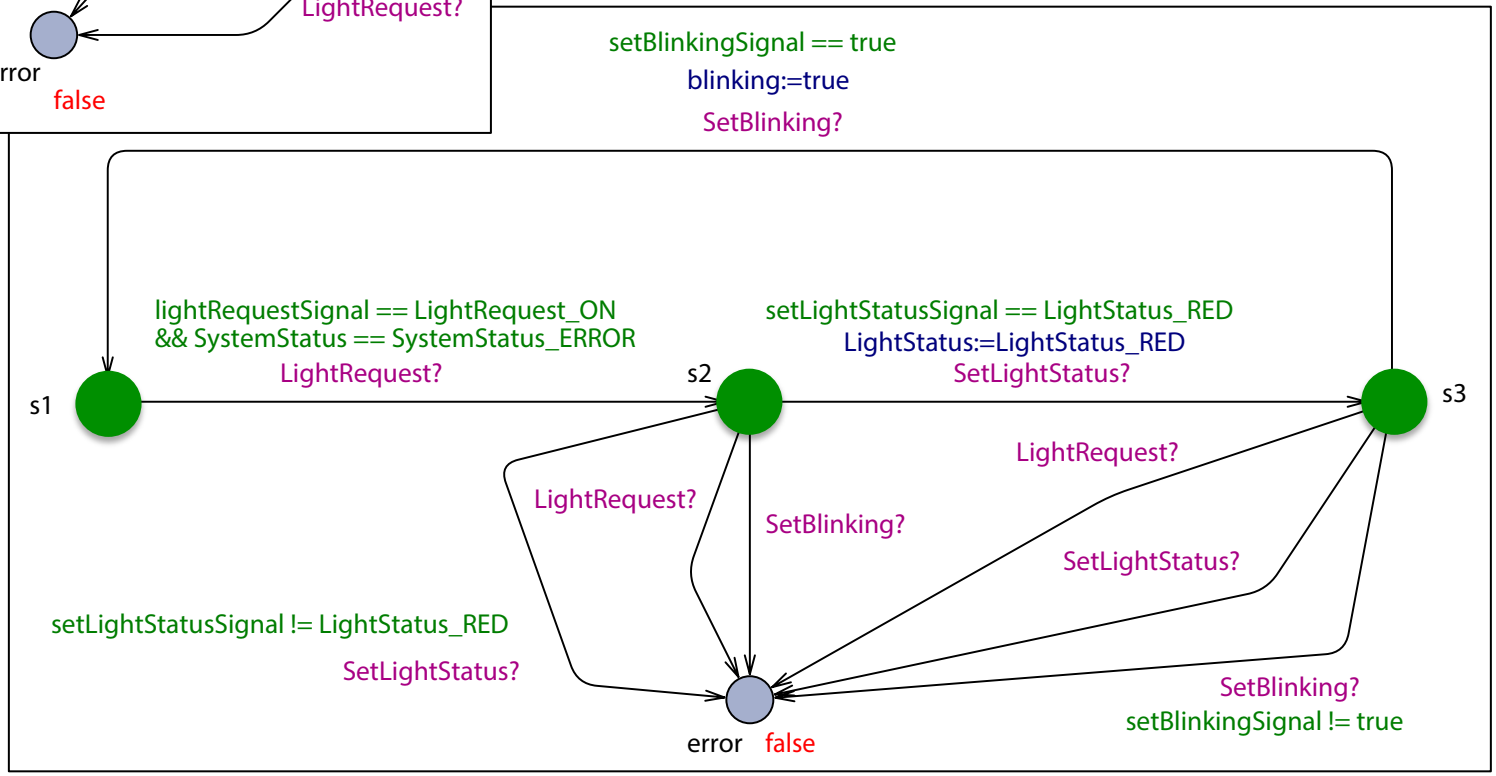
```
//Channels
broadcast chan sync;
```



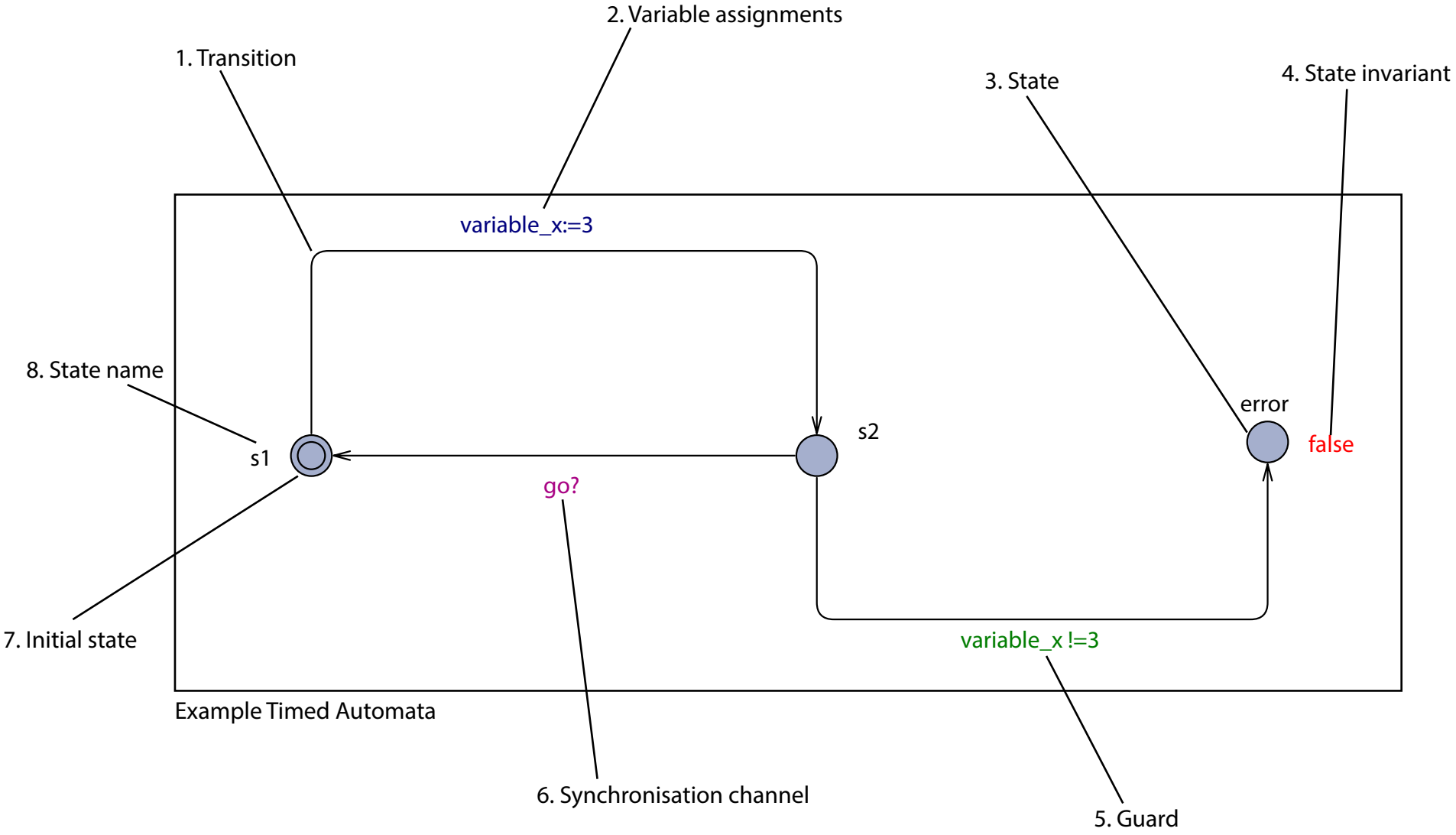


SystemStatus == SystemStatus_ERROR

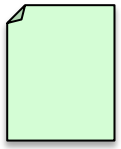
1. LightRequest!, lightRequestSignal:=LightRequest_ON
2. SetLightStatus!, setLightStatusSignal := LightStatus_RED
3. SetBlinking!, setBlinkingSignal := true



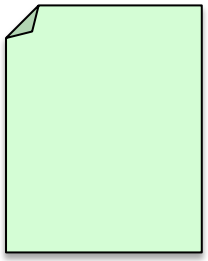
Timed Automata (TA)



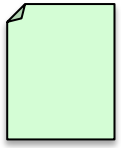
Experiment Setup (Car Wiper)



Pre-Experiment Questionnaire (Background, Experience)
3 minutes



Experiment Questionnaire (Actual Questions)
40 minutes



Post-Experiment Questionnaire (Difficulty, Impression)
2 minutes

Diagrams

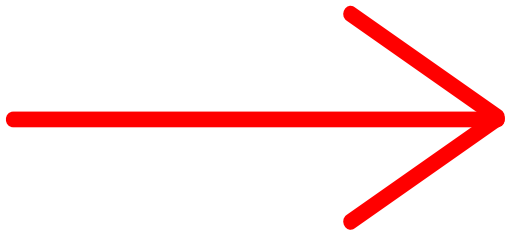
- Five Diagrams:
  Slow activation, fast activation, deactivation, start increase, long increase

- MSD: Class Diagram and Object Diagram
- TA: Declarations (“C code”)

Experiment Questionnaire

- Four blocks of three questions
  - Please note start and end time for each block
  - Don't rush too much
    → % completed is also a relevant measure!

- Arrows wherever you need to fill in something



Question Type I

WiperRequest!, wiperRequestSignal:=WiperRequest_OFF

Precondition:	Actuator_WiperSpeed Actuator_WiperState Wiper_VehicleStatus Wiper_WiperConfiguration	= Constants_SLOW = WiperState_ACTIVE = VehicleStatus_RUNNING = WiperConfig_INSTALLED
1.	*WiperRequest* is triggered, with *wiperRequestSignal* set to *WiperRequest_OFF*	
2.	*SetWiperSpeed* is triggered, with *setWiperSpeedSignal* set to *Constants_OFF*	

Question 1: Does the input scenario violate the specified behaviour?

Answer 1: No ×, Yes, in step: 1 ☐, 2 ☐

Question 2: Which values do the following variables have
- after the execution of the input scenario (if A1 is 'No')
- before the violating step (if A1 is 'Yes')?

Answer 2:

Actuator_WiperSpeed	= **Constants_OFF**
Actuator_WiperState	= **WiperState_ACTIVE**
Wiper_VehicleStatus	= **VehicleStatus_RUNNING**
Wiper_WiperConfiguration	= **WiperConfig_INSTALLED**

Experiment Convention

- `setX(param)`
  Sets variable X of receiving object to param

- addToX(param)
  Adds param to variable X of receiving object

Question Type II

Precondition:	Actuator_WiperSpeed = Constants_SLOW Actuator_WiperState = WiperState_ACTIVE Wiper_VehicleStatus = VehicleStatus_RUNNING Wiper_WiperConfiguration = WiperConfig_INSTALLED
1.	*WiperRequest* is triggered, with *wiperRequestSignal* set to *WiperRequest_OFF*
2.	*SetWiperSpeed* is triggered, with *setWiperSpeedSignal* set to *Constants_FAST*

Question 1: Does the input scenario violate the specified behaviour?

Answer 1: No ☐, Yes, in step: 1 ☐, 2 ✕

Question 2: In which states are the Automata
- after the execution of the input scenario (if A1 is 'No')
- before the violating step (if A1 is 'Yes')?

Answer 2:

ta_slow_activation:	State s1: ✕, s2: ☐, s3: ☐
ta_fast_activation:	State s1: ✕, s2: ☐, s3: ☐
ta_deactivation:	State s1: ☐, s2: ✕, s3: ☐
ta_start_increase:	State s1: ✕, s2: ☐
ta_long_increase:	State s1: ✕, s2: ☐