DESIGN & ELEKTRONIK
ENTWICKLER FORUM

»Embedded-System-Entwicklung«

5.-6. Juli 2011

Workshop 3

Embedded Systementwicklung auf Basis von Open-Source Tools und Eclipse

about us …

Andreas Unger
Software Engineer
andreas.unger@itemis.de

Axel Terfloth
Head R&D Embedded Systems
axel.terfloth@itemis.de

- work at itemis AG, Germany
- work on model driven development of embedded systems
- work on YAKINDU open source project

about itemis AG ...

Open Source

- Eclipse Strategic Member
- Eclipse Committer – Eclipse Modeling Project (Xpand, Xtext)
- others: YAKINDU, openArchitectureWare, CM3, formax, jmove

eclipse STRATEGIC MEMBER

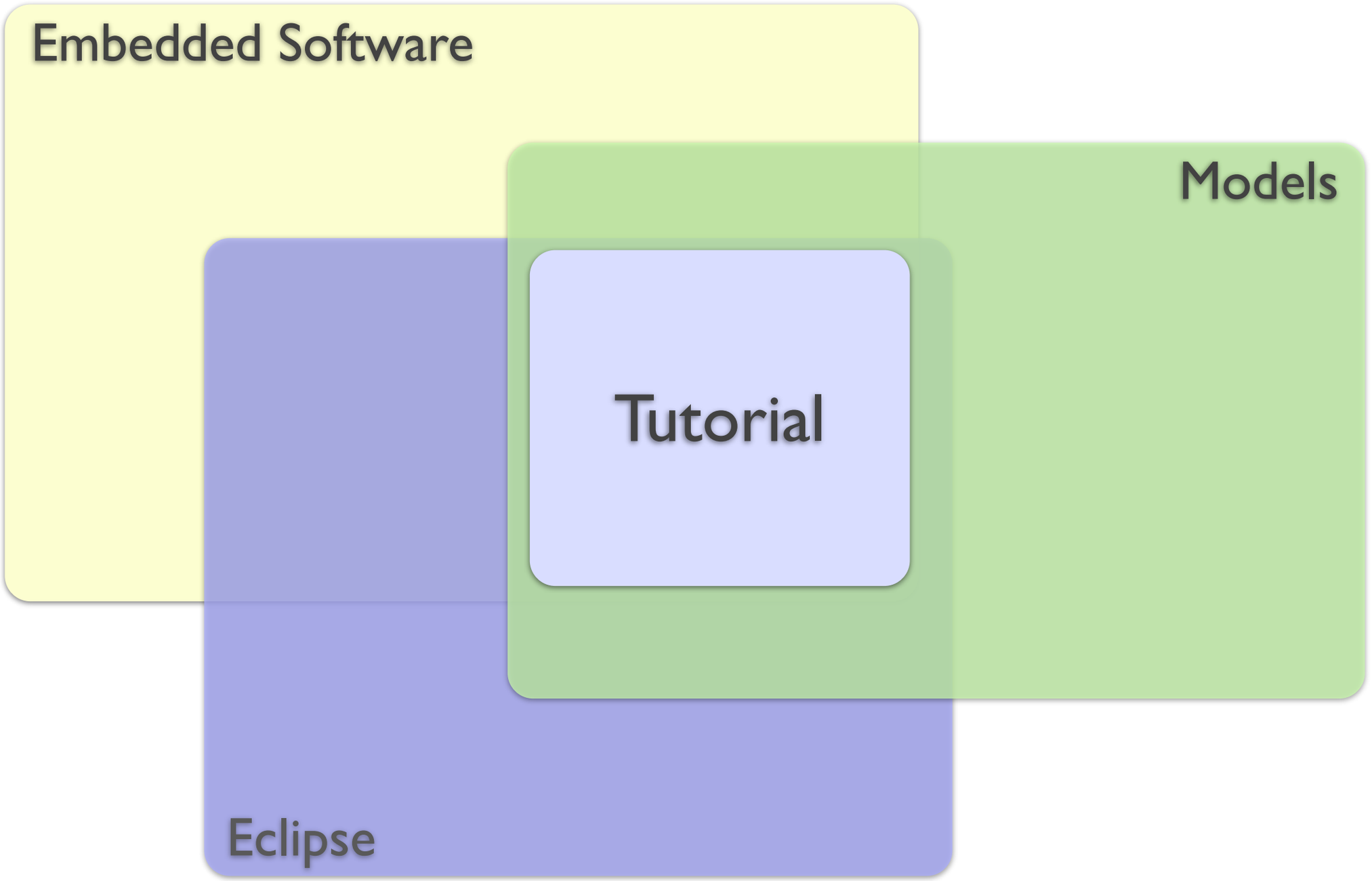
founded 2003, ± 140 people

Training, Coaching, Consulting

- Model Driven Software Development (MDSD)
- Processes, Methodology, & Tools

- Embedded Systems, Mobile Apps, Enterprise Systems

Tutorial Context

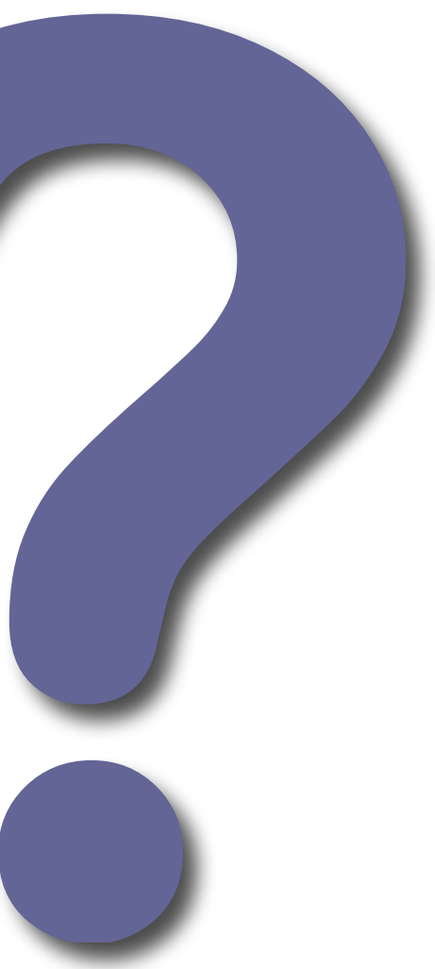


Goals…

- See how Eclipse can support Embedded Software Development beyond coding
- See how models can help dealing with increasing software complexity
- Understand how state machines work
- Understand how block diagrams work
- Understand the role of Eclipse technologies

Agenda

- What is Eclipse good for ?
- What is Yakindu ?
- Model driven development
- Block diagtrams
- Statecharts
- Textual models
- Integrating all in an example application
- Generating code



What is Eclipse good for ?

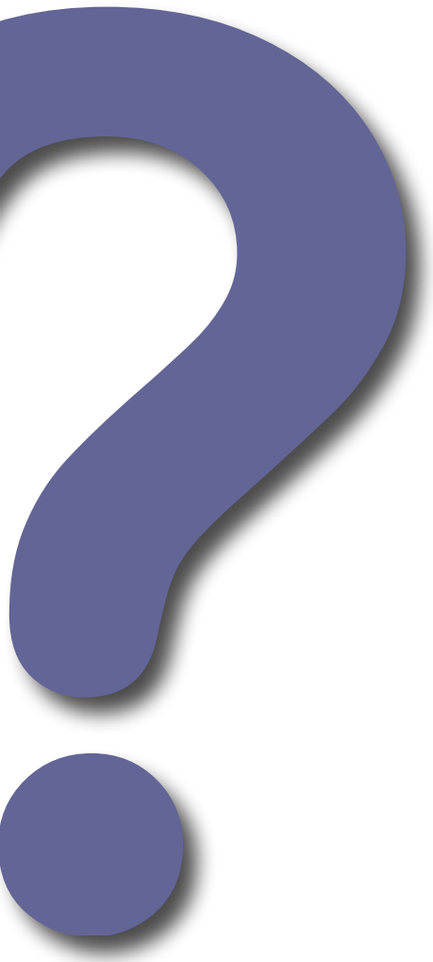
Known as Integrated Development Environment

Eclipse

- Application / Tool Platform
- Open Architecture
- Designed for Extensibility
- Strong Modeling Infrastructure
- Open Source
- Reduced Costs

Eclipse

- Eclipse Modeling Framework
- Graphical Editing/Modeling Framework
- Xpand - Generator Framework
- Xtext - DSL Toolkit
- UML2 support
- …



What is YAKINDU ?

YAKINDU

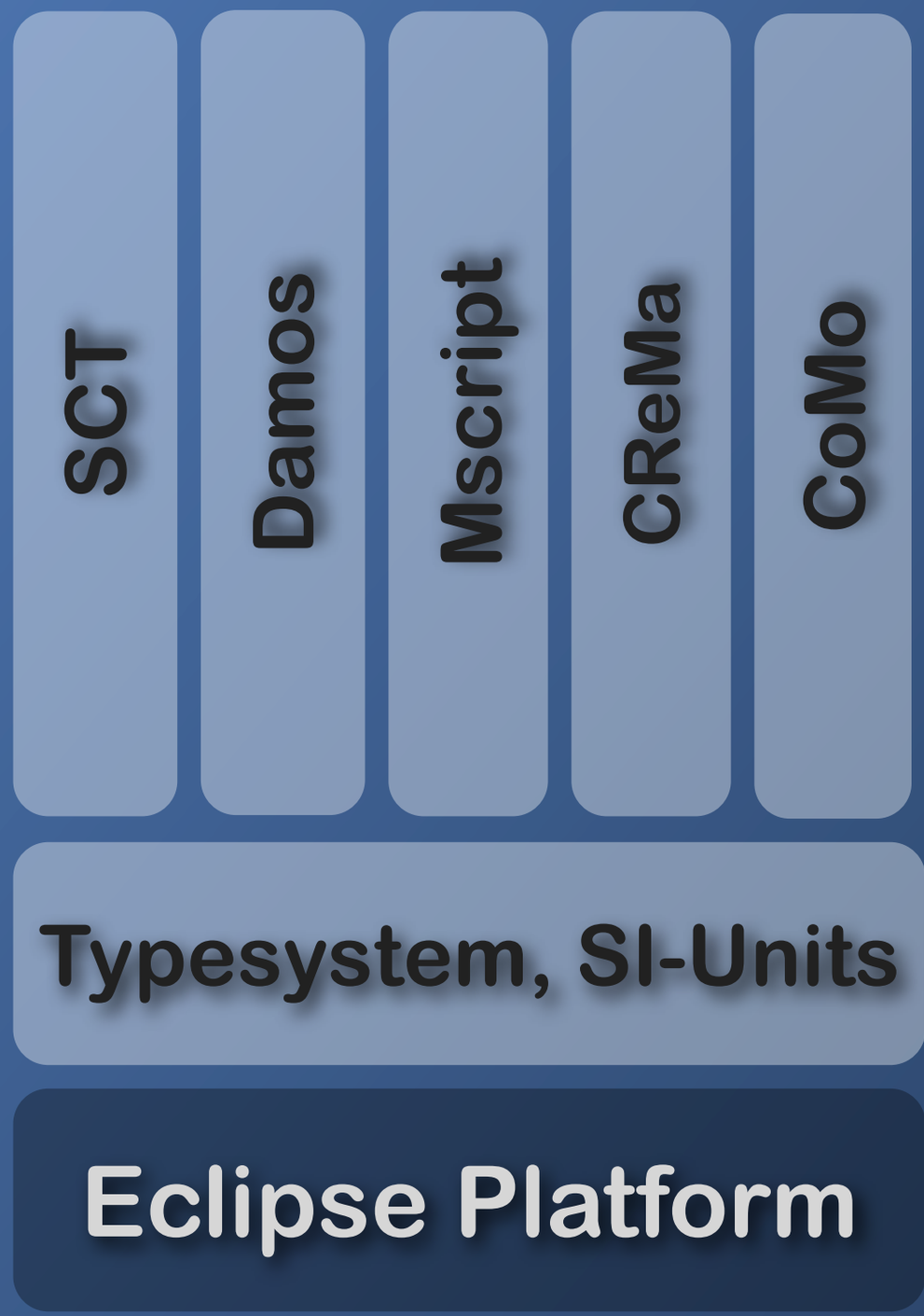
YAKINDU
is a ***modular toolkit***
for ***model driven development***
of embedded systems

Yakindu Language Modules

- SCT - statecharts
- Damos - data-flow oriented modeling
- Mscript - math oriented scripting
- CReMa - (requirements) traceability
- CoMo - component model (upcoming)

language modules support

- editor
- simulator & validation
- code generator



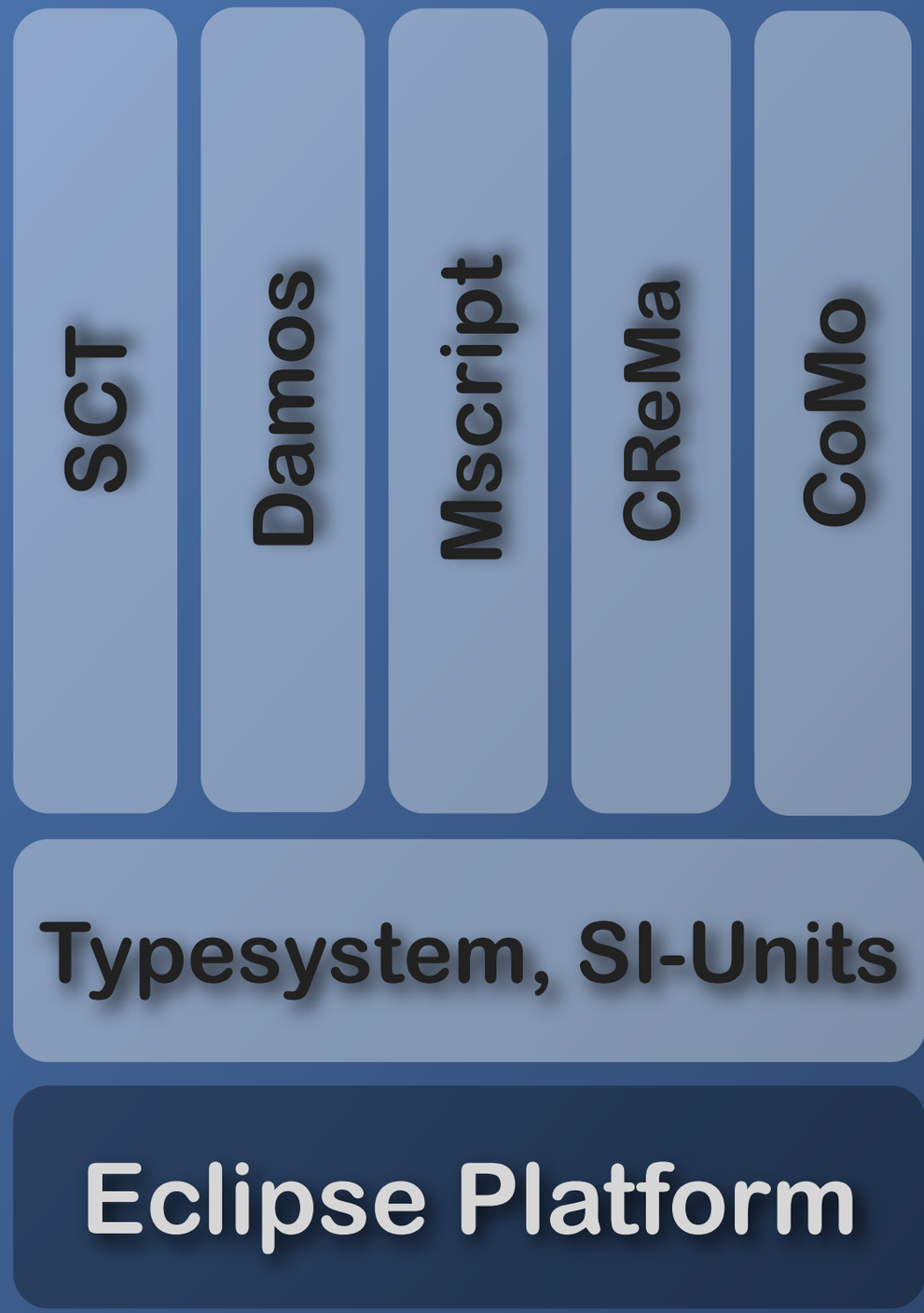
Yakindu is …

- built on Eclipse
- open source
- available at Eclipse Labs

http://eclipselabs.org/p/yakindu

http://yakindu.org

Eclipse Project Proposal: 2011



Model Driven Development

Model Based Software Engineering

makes systematic use of (formal) engineering models as primary engineering artifacts throughout the overall engineering life-cycle

Source: Nyß09

Model Driven Software Development (MDSD)

is a generic term for techniques, that transform formal models into executable software.

Source: Stahl/Völter

Model Driven Development (MDD)

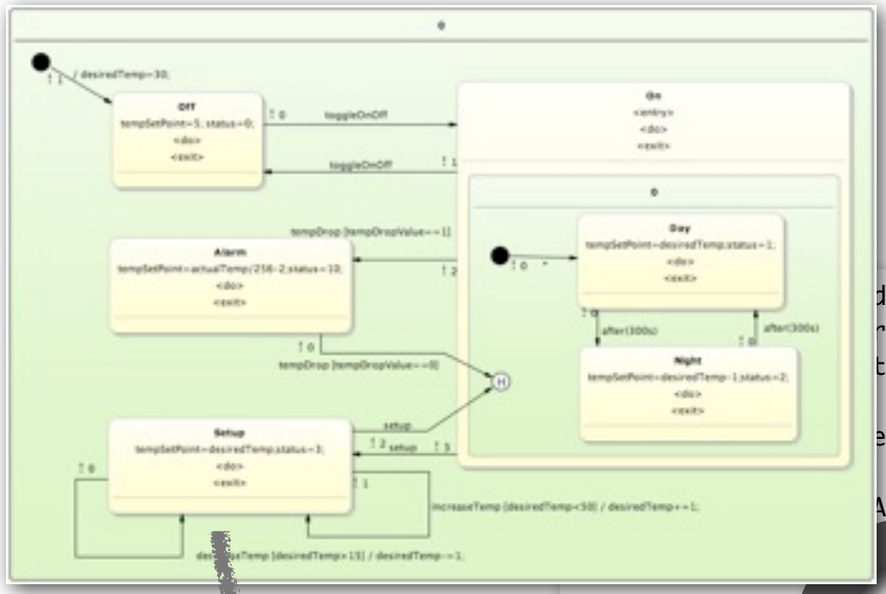
Simplification by abstraction

- hide complexity
- focus on domain relevant aspects
- Generator encapsulates technical details

Transformation to target platform using generators (or interpreters)

- avoid redundancy (DRY: Don‘t Repeat Yourself)
- enforce architecture and design guidelines
- improve quality / reduce error rate / fix errors once
- improve productivity
- more flexibility for variants

Abstraction Levels



generate

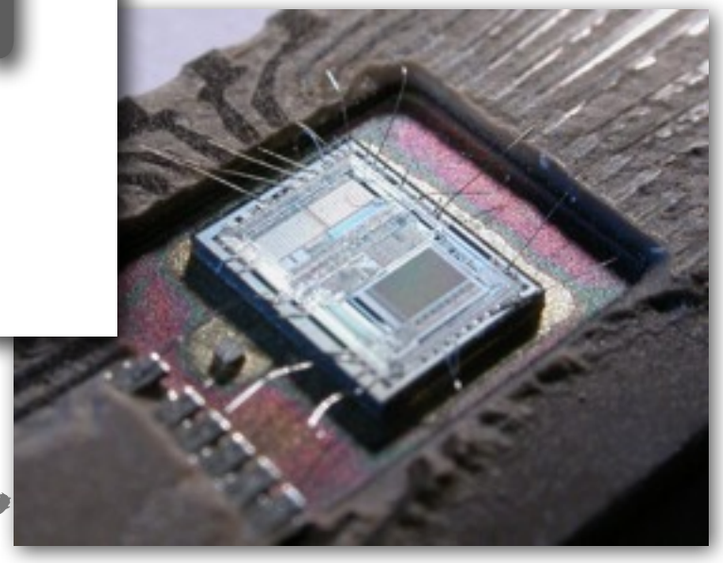
```
dler(void) {
rage = 0;
ter = ADC_COUNTER;

e;

ASE, 0);
ASE, 0, &value);

average += va

if (--adcCoun == 0) {
  average = co rtADCtoT erature(
                F -ROM_INT
(average),
                FIX_FROM_INT(ADC_COUNTER)));
  sensor_actualTemperature_send(&average);
  average = 0;
  adcCounter = ADC_COUNTER;
 }
}
```

compile

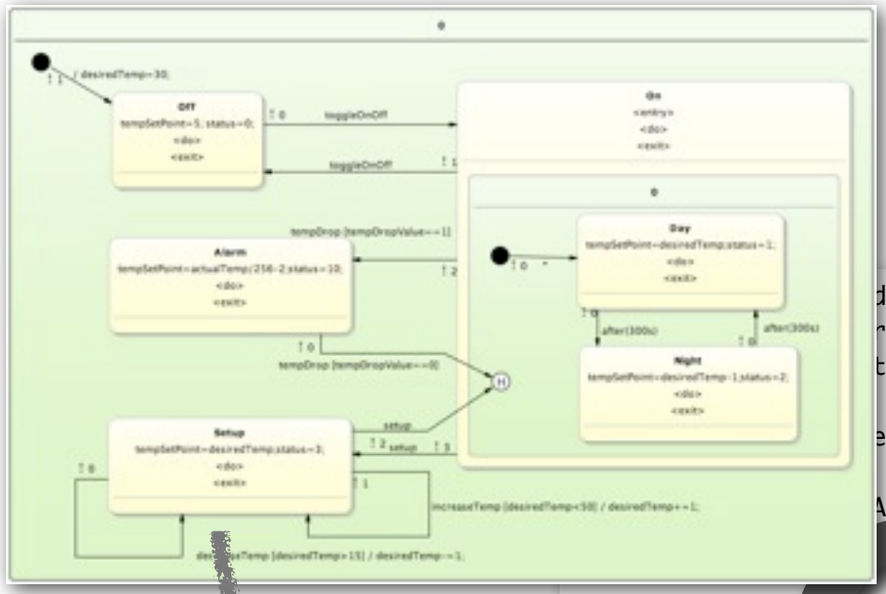
```
r3, #:lower16:.LANCHOR0
r3, #:upper16:.LANCHOR0
[r4, r5, r6, r7}
r2, [r0, #4]
r5, [r0, #0]
r4, [r3, #0]
r6, [r0, #4]
r0, #:lower16:274779
r5, r2
lsl     r4,
movt    r0, #:upper16:274877907
smull   r7, r5, r0, ip
subs    r6, r2, r6
mov     r7, #2048000
mul     r6, r7, r6
asr     ip, ip, #31
lsls    r7, r2, #10
```

Asm

C code as an intermediate artifact



Abstraction Levels



model of

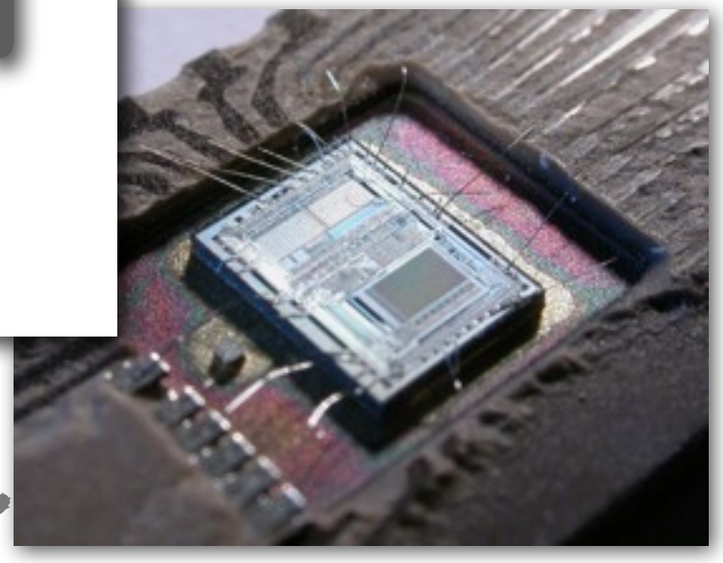
```
dler(void) {
rage = 0;
ter = ADC_COUNTER;

e;

ASE, 0);
ASE, 0, &value);

average += va

if (--adcCoun == 0) {
  average = co rtADCtoT erature(
                F -ROM_INT
(average),
                FIX_FROM_INT(ADC_COUNTER)));
  sensor_actualTemperature_send(&average);
  average = 0;
  adcCounter = ADC_COUNTER;
 }
}
```

model of

```
r3, #:lower16:.LANCHOR0
r3, #:upper16:.LANCHOR0
[r4, r5, r6, r7}
r2, [r0, #4]
r5, [r0, #0]
r4, [r3, #0]
r6, [r0, #4]
r0, #:lower16:274779
r5, r2
lsl     r4,
movt    r0, #:upper16:274877907
smull r7, r5, r0, ip
subs    r6, r2, r6
mov     r7, #2048000
mul     r6, r7, r6
asr     ip, ip, #31
lsls    r7, r2, #10
```

Asm

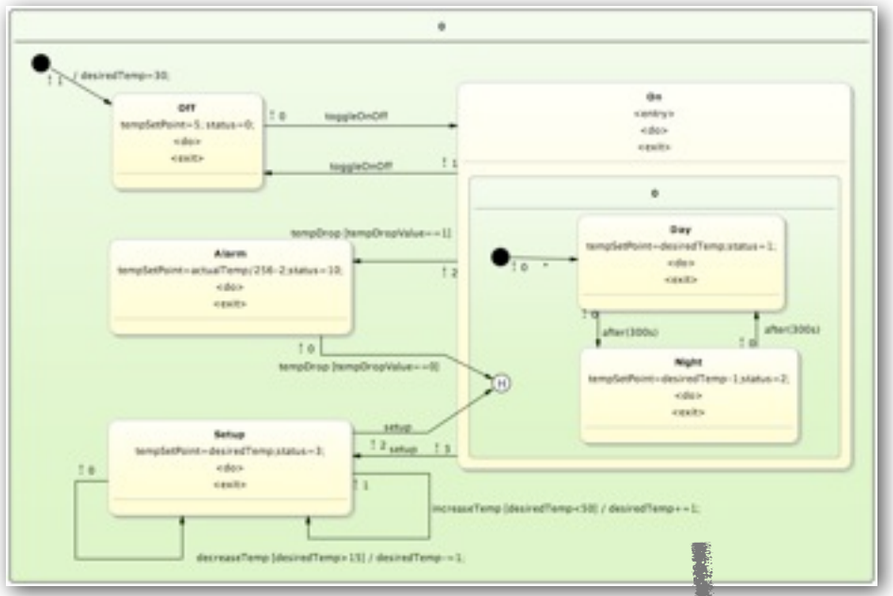
model of



C code as an intermediate artifact

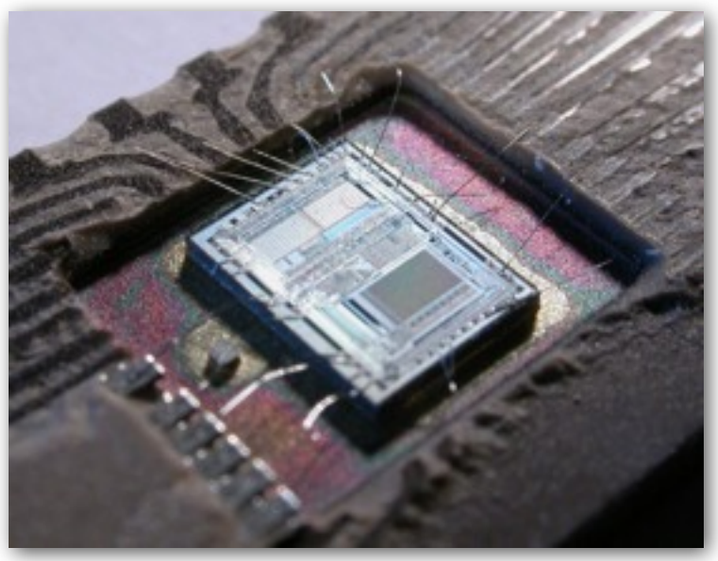
Abstraction Levels

model and c code for disjoint aspects




```
void adcSequence0Handler(void) {
  static int32_t average = 0;
  static int adcCounter = ADC_COUNTER;

  unsigned long value;

  ADCIntClear(ADC0_BASE, 0);
  ADCSequenceDataGet(ADC0_BASE, 0, &value);

  average += value;

  if (--adcCounter == 0) {
    average = convertADCToTemperature(
                  FIX_FROM_INT
(average),
                  FIX_FROM_INT(ADC_COUNTER));
    sensor_actualTemperature_send(&average);
    average = 0;
    adcCounter = ADC_COUNTER;
```

model of



a model is code

code is a model

When is MDD applicable?

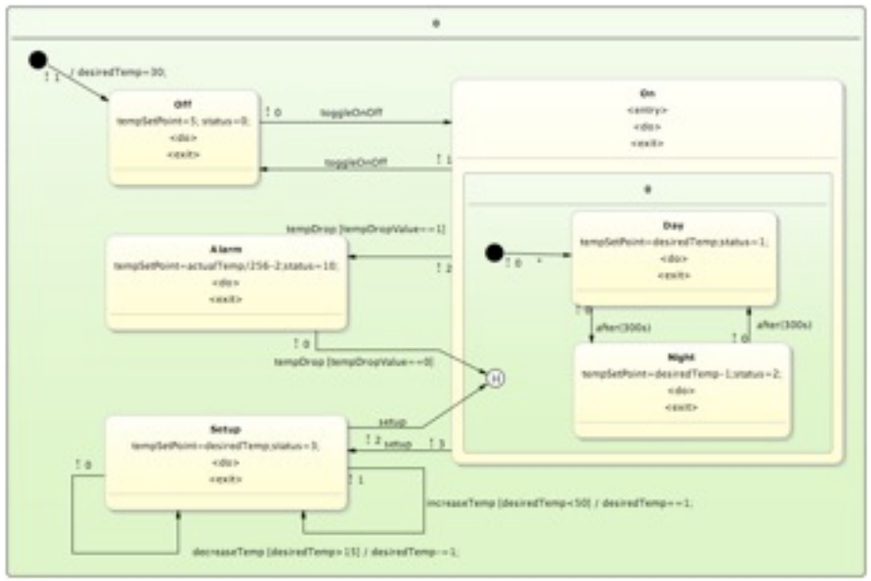
- same solution for the same problem
- schematic technical implemenation
- differences can be specified in the model
- less effort for specification than for implementation

- critical mass will be reached
  - extent of models
  - frequency of change
  - number of target systems
  - many variants

We‘ll take a look at …

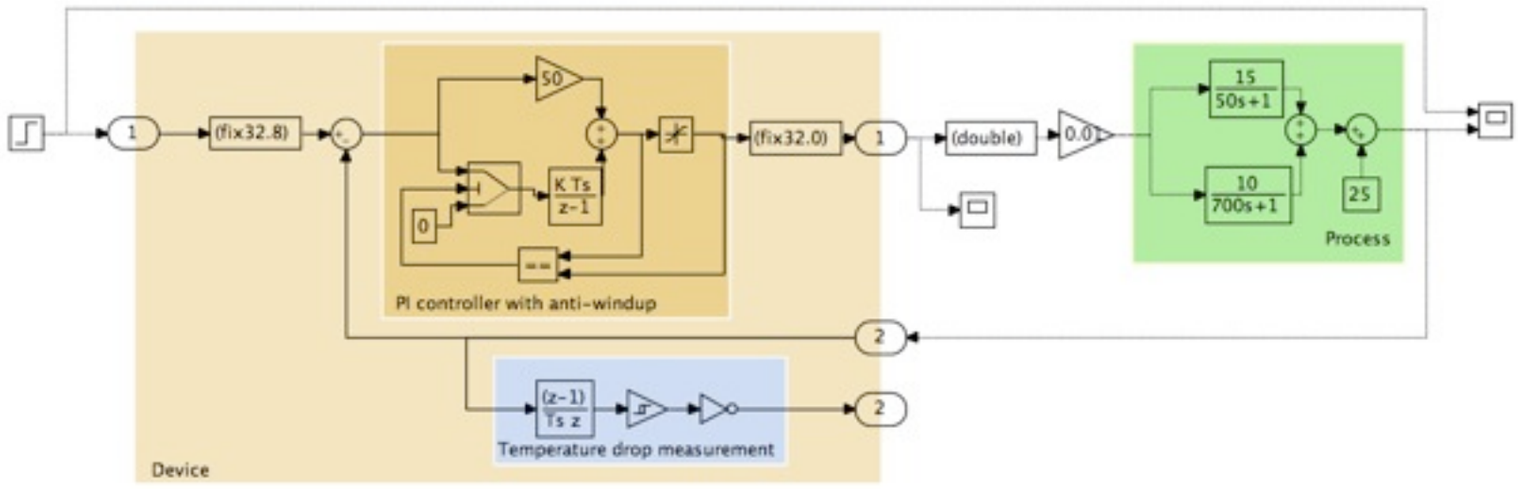
We'll take a look at …

statecharts



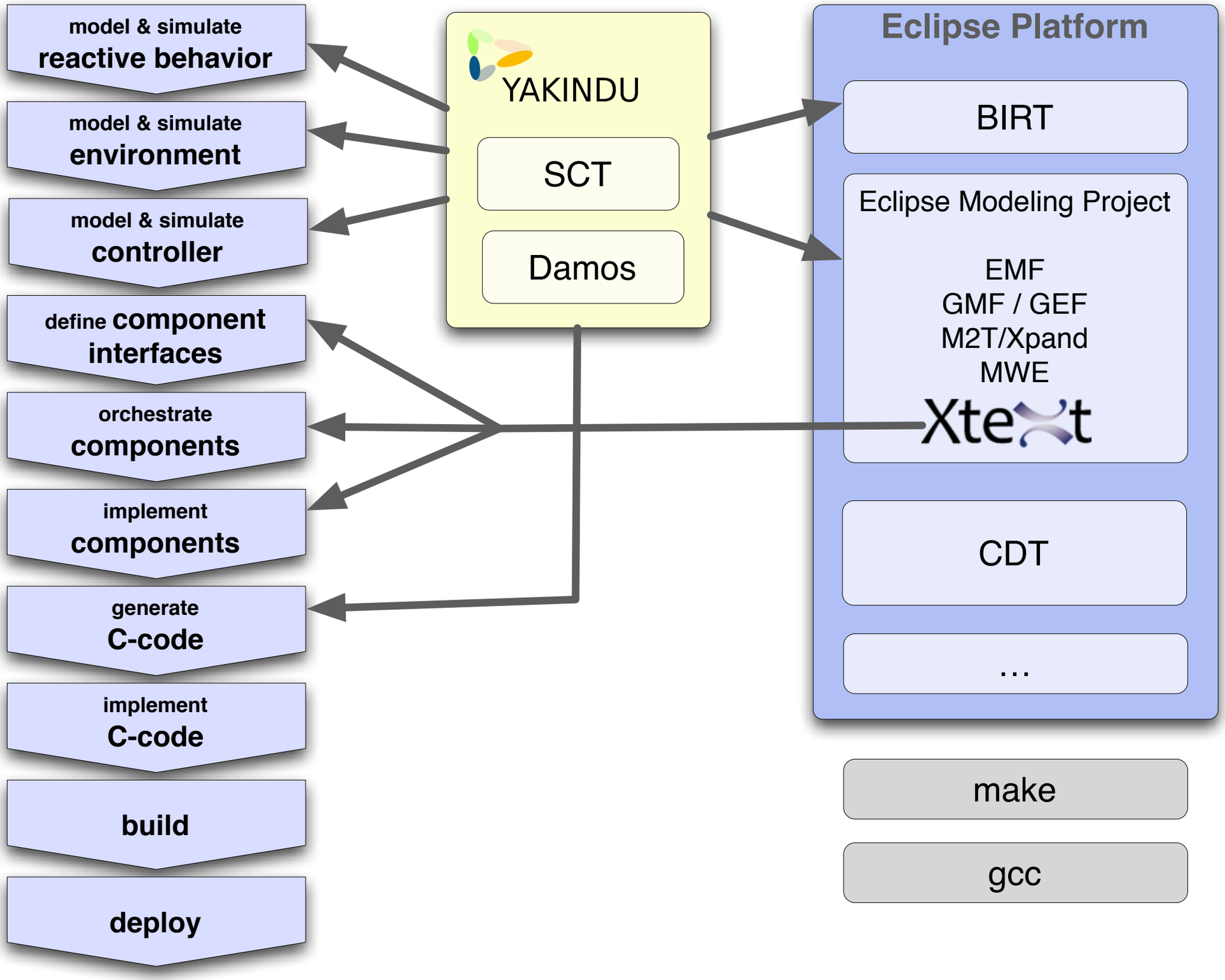

```
//
// define components
//
component Heater {
    in power (queued);
}

component Fan {
    in power (queued);
}

component TemperatureSensor {
    out actualTemperature mapped to isr;
}

component ButtonPanel {
    out select   mapped to isr;
    out up     mapped to isr;
    out down   mapped to isr;
    out left   mapped to isr;
    out right mapped to isr;
}
```

textual modeling



block diagrams

Activities and Tools

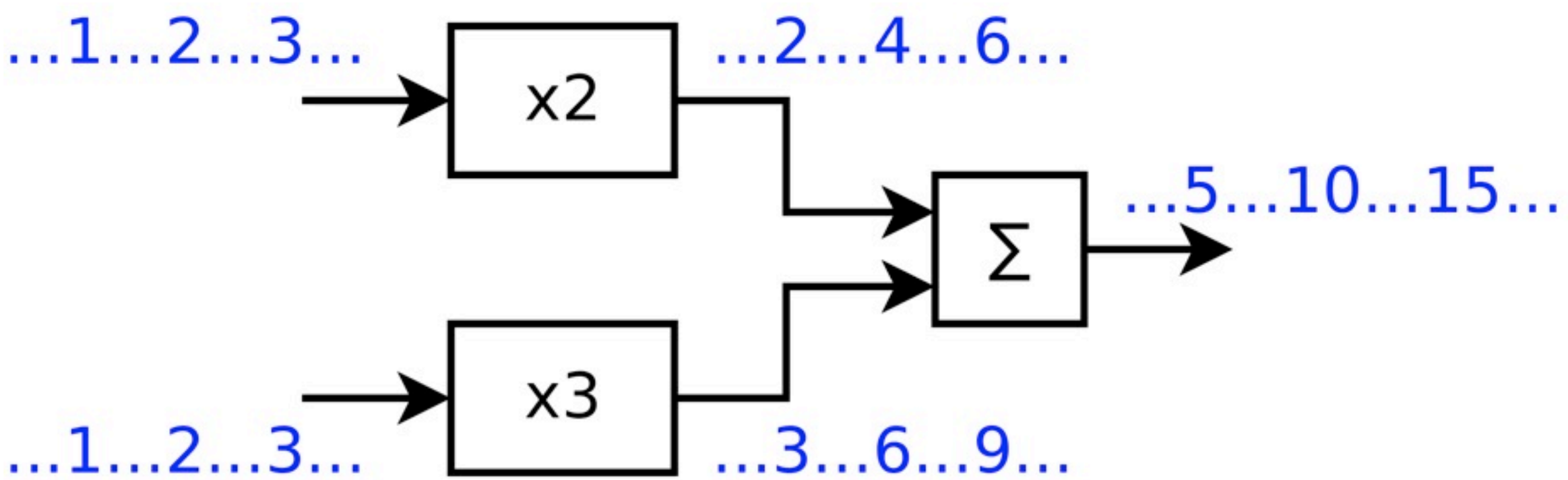


Damos Block Diagrams

Data Flow-Oriented Modeling

- *Data* as main concept, instead of *state*
- Prevalent notation: Block diagrams
  - Block: System component's transfer function
  - Connection: Data flow (e.g. physical quantities)
- Technical applications: Control systems & digital signal processing

Block Composition



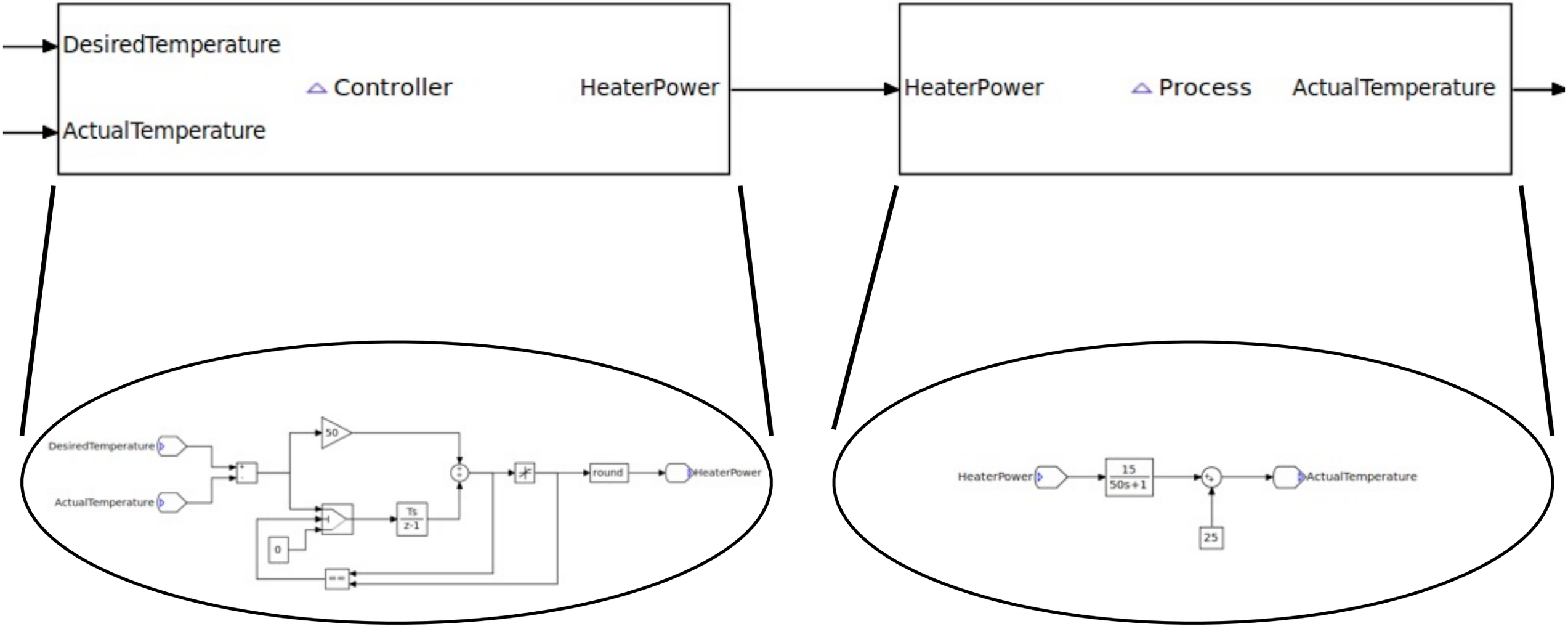
What is Damos?

- Data flow-oriented modeling environment
- Eclipse-based
- Open source
- Comes with
  - Block diagram editor
  - Simulator
  - C99 compliant C code generator
  - Base block library

Structuring Models

- System components are divided into three categories
  - Device components (e.g. digital controller)
  - Environmental components (e.g. process)
  - Simulation interface components (e.g. step functions and scopes)
- Damos supports „two-dimensional“ structuring
  - Hierarchal structuring using subsystems
  - Cross-cutting structuring using system fragments

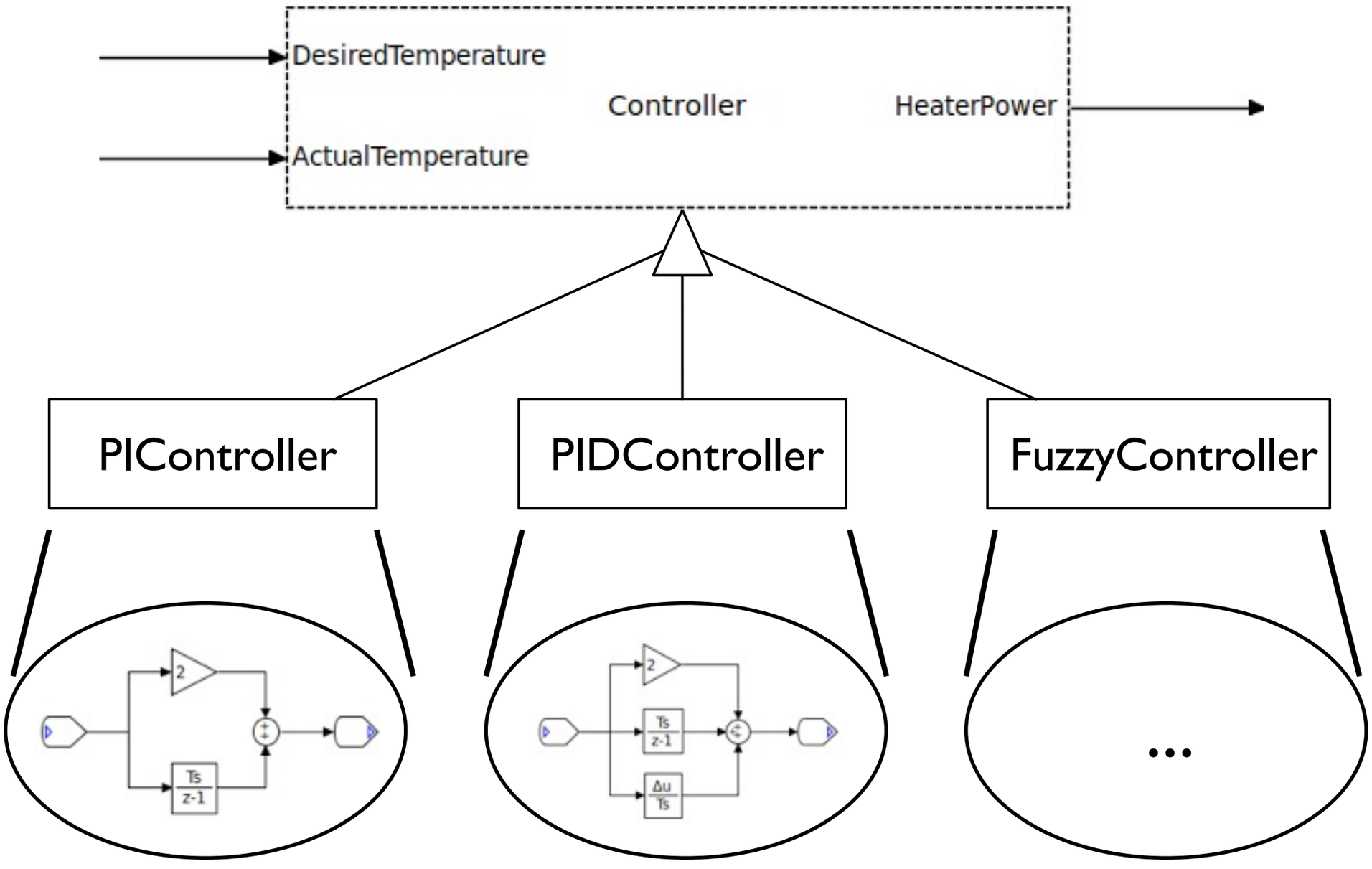
Subsystems


DesiredTemperature
ActualTemperature
Controller
HeaterPower
HeaterPower
Process
ActualTemperature
DesiredTemperature
ActualTemperature
50
0
Ts
z-1
round
HeaterPower
HeaterPower
15
50s+1
25
ActualTemperature

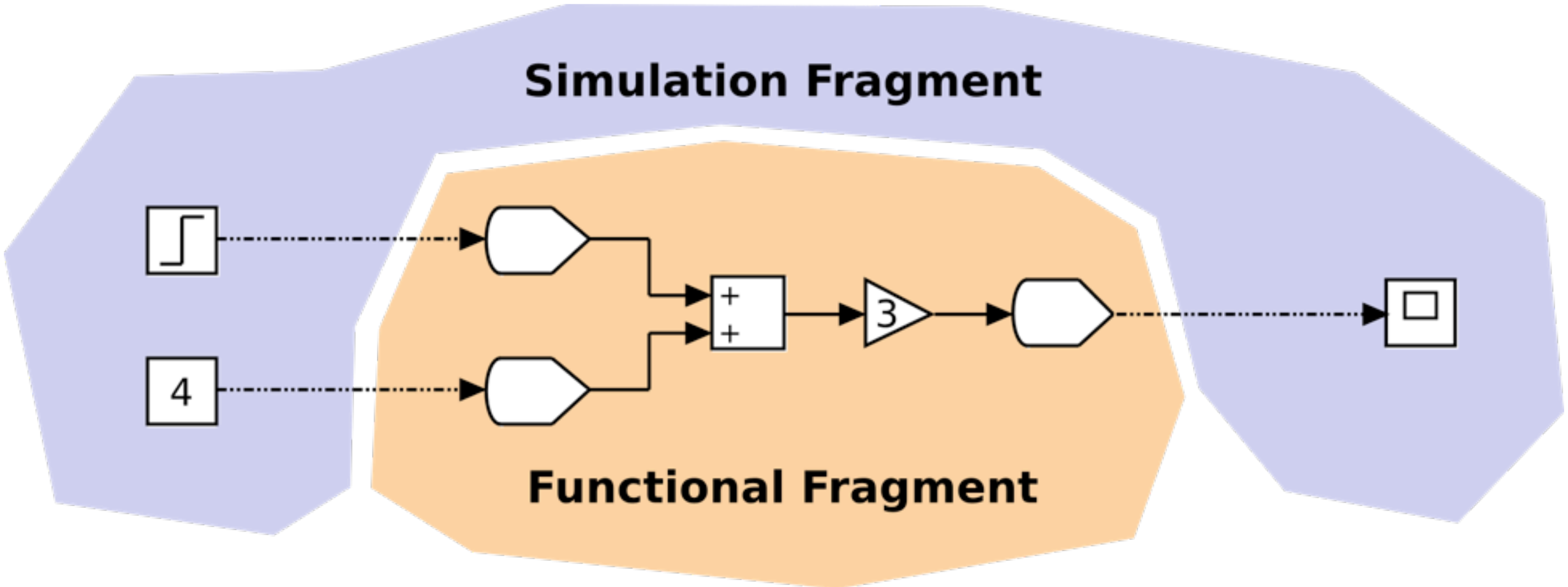
System Interfaces

- Defines system inlets and outlets, and their data types
- Subsystems specify their *provided interface*
- Realizing system is specified with *subsystem realization* model element
- Allows for specifying a subsystem without specifying a realization
- Can be used in product line engineering

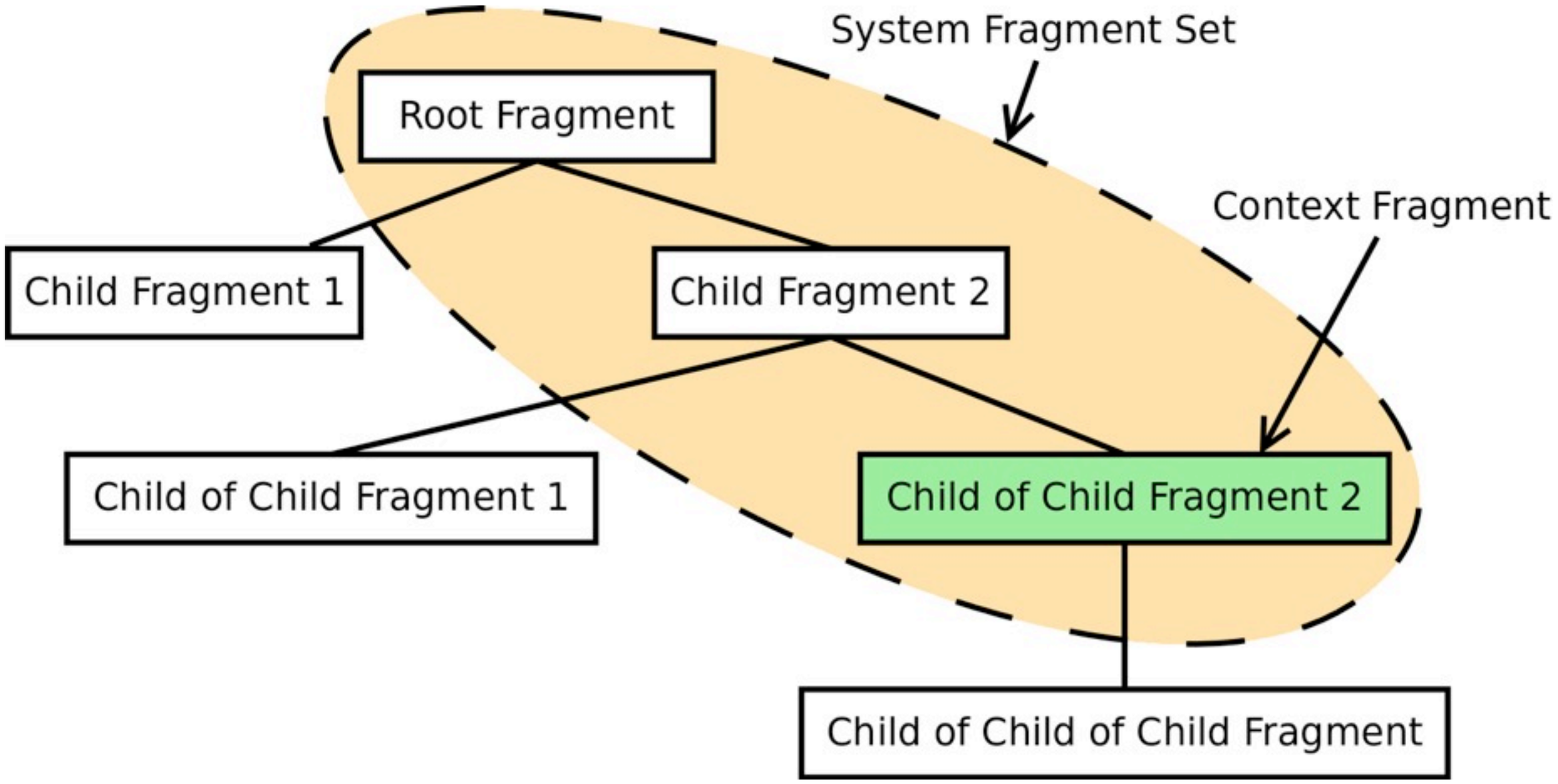
Subsystem Realizations



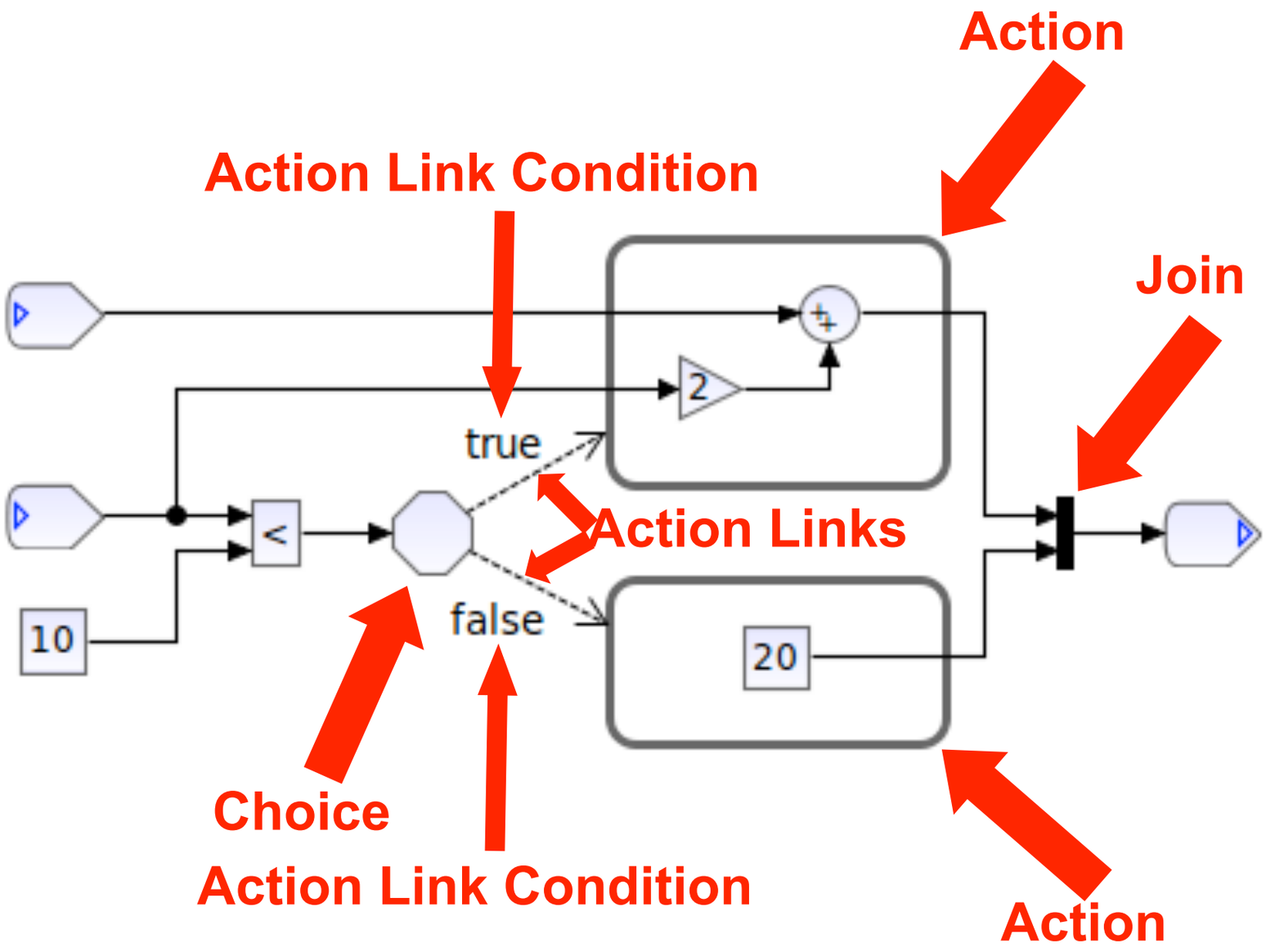
System Fragments



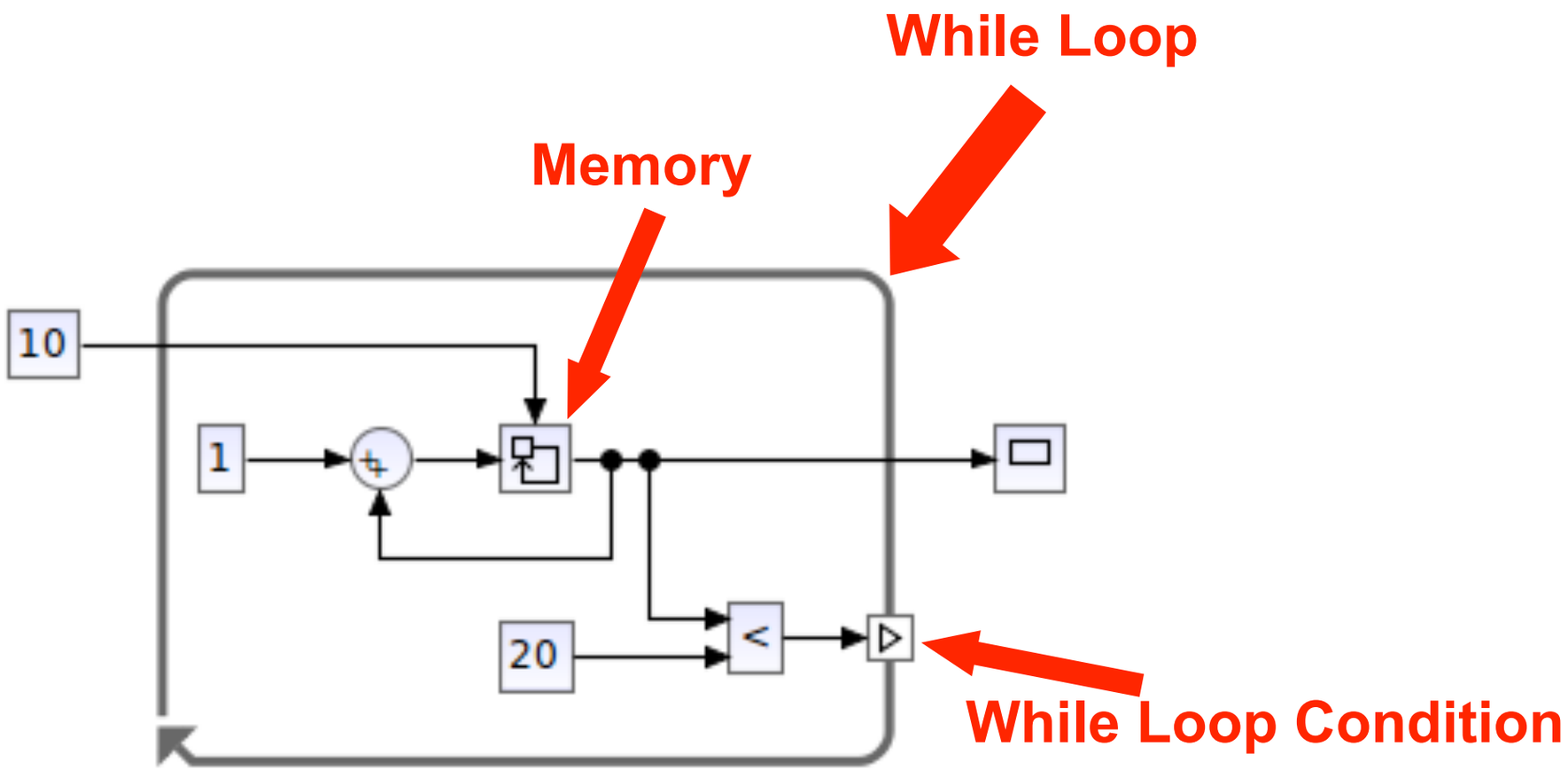
System Fragment Hierarchy



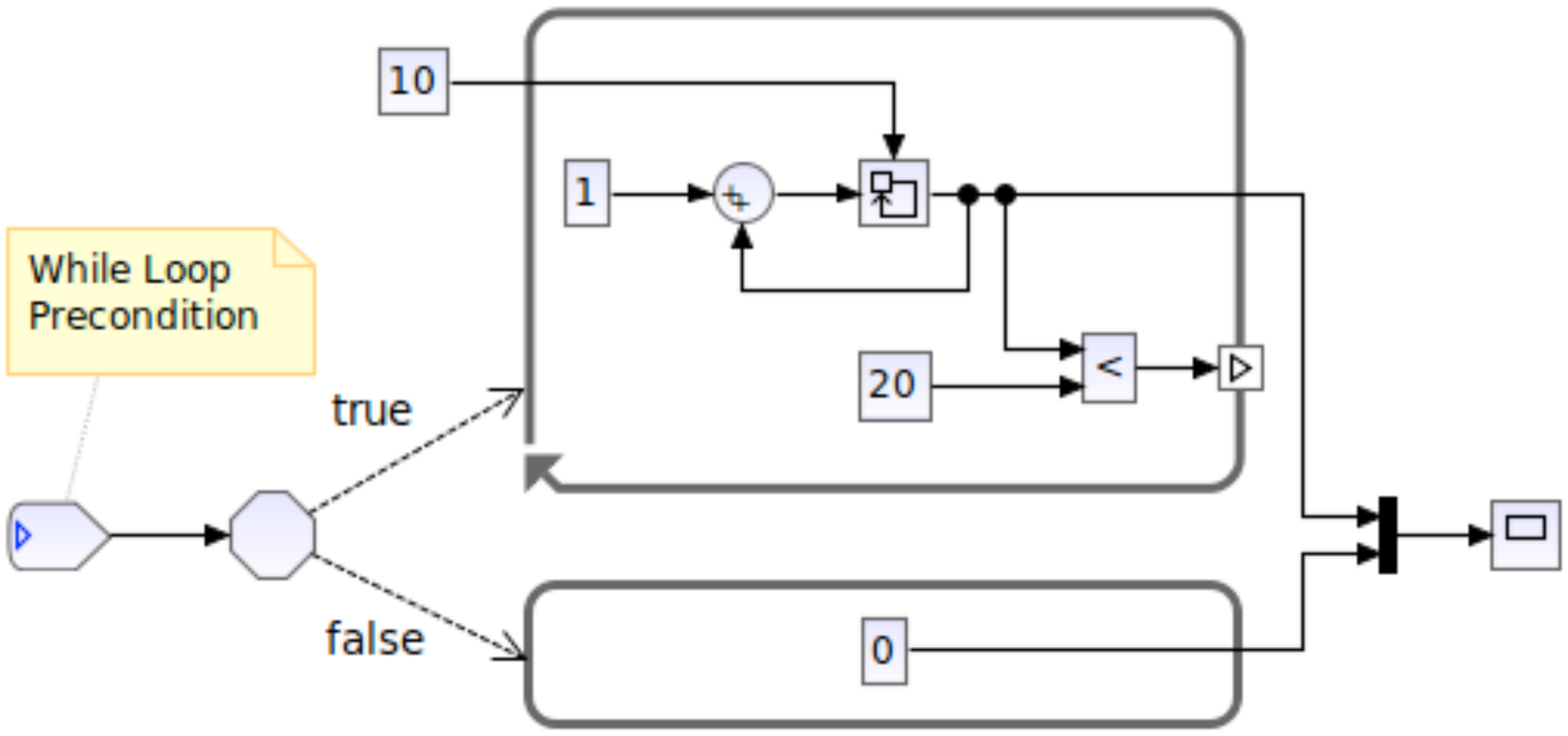
Action Compounds



While Loops

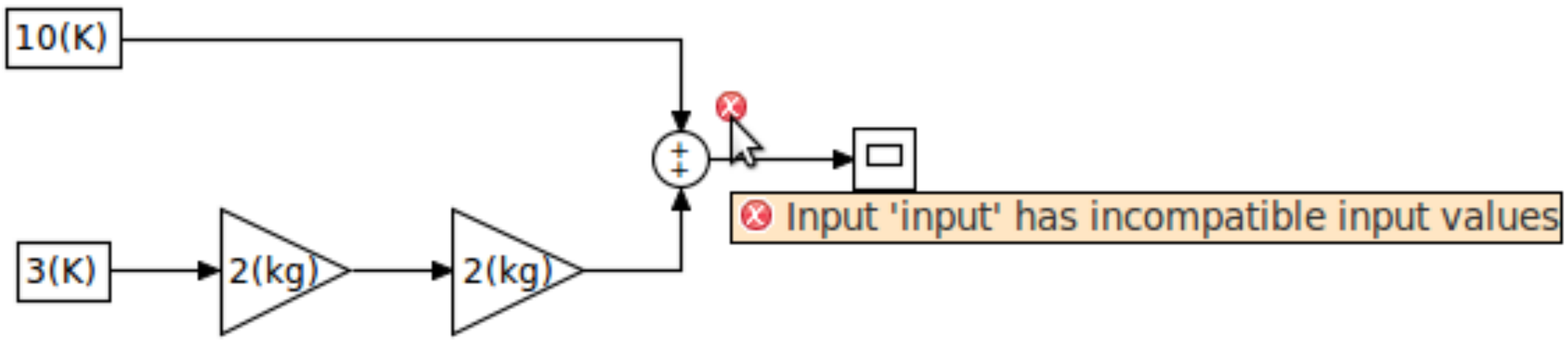


While Loops (cont.)



Units of Measurement

- Numeric data types incorporate unit of measurement
- When no unit is specified, dimensionless value is assumed
- Used for model validation



Defining Block Behavior with Mscript

- Math-oriented declarative programming language
- Relationship between inputs and outputs are defined using equations
- Functions can be stateful
  - Previous values (inputs, outputs, or state variables) can be referenced
  - Equations are difference equations in stateful functions
- Numeric data types incorporate units of measurement
- Used for simulation and code generation in Damos

Mscript Example

- Discrete derivative:

$$y[n] = \frac{x[n] - x[n-1]}{T_s}$$


```
stateful func discreteDerivative<xinit, Ts>(x) -> y {
        x(-1) = xinit;
        y(n) = (x(n) - x(n-1)) / Ts;
}
```

Computation Model

- Mapping from logical data type (e.g. real) to machine data type (e.g. IEEE 754 binary64)
- Supports fixed point data types
- Defines computation algorithm of arithmetic operations
- Specifies intermediate word size and saturation for fixed-point arithmetic operations
- Used by the fixed-point overflow detection mechanism

Mscript-based Code Generation

```
func Product(a, b) -> y {
        y = a * b;
        }
```

Floating Point

```
void Sys_execute(
      const Sys_Input *input,
      Sys_Output *output) {

   double Sys_Product_y;

   /* Product */
   {
      Sys_Product_y = input->a * input->b;
   }

   /* Y */
   {
      output->y = Sys_Product_y;
   }

}
```

Fixed Point

```
void Sys_execute(
      const Sys_Input *input,
      Sys_Output *output) {

   int32_t Sys_Product_y;

   /* Product */
   {
      Sys_Product_y =
        (int32_t) (((int64_t) (input->a))
        * (((int64_t) (input->b))) >> 16);
   }

   /* Y */
   {
      output->y = Sys_Product_y;
   }

}
```

Mscript and Damos

- Damos incorporates Mscript's type system
- Block behavior of discrete blocks are defined using Mscript functions
  - Block parameters map to Mscript template parameters
  - Block inputs map to Mscript input parameters
  - Block outputs map to Mscript output parameters
  - For multi-port inputs and outputs, an array is passed to the Mscript function

Mscript and Damos (cont.)

- Damos simulator uses Mscript function to simulate blocks
  - No Java code has to be provided for those blocks
- Damos code generator uses Mscript function to generate C code
  - No code template has to be provided for those blocks
- Simulation algorithm match generated C code algorithm
- Damos uses Mscript's computation model and interpreter for simulation
  - Mscript's Fixed-point overflow detection is integrated into Damos

Code Optimization

```
BlockDiagram_execute:
    push    {r7}
    sub     sp, sp, #60
    add     r7, sp, #0
    str     r0, [r7, #4]
    str     r1, [r7, #0]
    mov     r3, #0
    str     r3, [r7, #12]
    mov     r3, #0
    str     r3, [r7, #16]
    mov     r3, #0
    str     r3, [r7, #20]
    mov     r3, #0
    str     r3, [r7, #28]
    mov     r3, #0
    str     r3, [r7, #32]
    movw    r3, #:lower16:BlockDiagram_DiscreteIntegrator_sum
    movt    r3, #:upper16:BlockDiagram_DiscreteIntegrator_sum
    ldr     r3, [r3, #0]
    lsl     r2, r3, #8
    movw    r3, #:lower16:274877907
    movt    r3, #:upper16:274877907
    smull   r1, r3, r3, r2
    asr     r1, r3, #17
    asr     r3, r2, #31
    rsb     r3, r3, r1
    str     r3, [r7, #8]
    ldr     r3, [r7, #8]
    str     r3, [r7, #36]
    ldr     r2, [r7, #32]
    ldr     r3, [r7, #36]
    add     r3, r2, r3
    str     r3, [r7, #32]
    ldr     r3, [r7, #4]
    ldr     r3, [r3, #0]
    ldr     r2, [r7, #16]
    add     r3, r2, r3
    str     r3, [r7, #16]
    ldr     r3, [r7, #4]
    ldr     r3, [r3, #4]
    ldr     r2, [r7, #16]
    rsb     r3, r3, r2
    str     r3, [r7, #16]
    ldr     r3, [r7, #16]
    str     r3, [r7, #40]
    ldr     r3, [r7, #40]
    lsl     r3, r3, #10
    asr     r3, r3, #8
    str     r3, [r7, #12]
    ldr     r3, [r7, #40]
    lsl     r3, r3, #10
    asr     r3, r3, #8
    str     r3, [r7, #20]
    ldr     r3, [r7, #40]
    str     r3, [r7, #28]
    ldr     r3, [r7, #20]
    str     r3, [r7, #44]
    ldr     r2, [r7, #32]
    ldr     r3, [r7, #44]
    add     r3, r2, r3
    str     r3, [r7, #32]
    movw    r3, #:lower16:BlockDiagram_DiscreteDerivative_previousValue
    movt    r3, #:upper16:BlockDiagram_DiscreteDerivative_previousValue
    ldr     r3, [r3, #0]
    ldr     r2, [r7, #28]
    rsb     r3, r3, r2
    str     r3, [r7, #24]
    ldr     r3, [r7, #24]
    mov     r2, #2048000
    mul     r3, r2, r3
    asr     r3, r3, #8
    str     r3, [r7, #24]
    ldr     r3, [r7, #24]
    lsl     r3, r3, #10
    asr     r3, r3, #8
    str     r3, [r7, #24]
    ldr     r3, [r7, #24]
    str     r3, [r7, #48]
    ldr     r2, [r7, #32]
    ldr     r3, [r7, #48]
    add     r3, r2, r3
    str     r3, [r7, #32]
    ldr     r3, [r7, #32]
    str     r3, [r7, #52]
    ldr     r3, [r7, #0]
    ldr     r2, [r7, #52]
    str     r2, [r3, #0]
    movw    r3, #:lower16:BlockDiagram_DiscreteIntegrator_sum
    movt    r3, #:upper16:BlockDiagram_DiscreteIntegrator_sum
    ldr     r2, [r3, #0]
    ldr     r3, [r7, #12]
    add     r2, r2, r3
    movw    r3, #:lower16:BlockDiagram_DiscreteIntegrator_sum
    movt    r3, #:upper16:BlockDiagram_DiscreteIntegrator_sum
    str     r2, [r3, #0]
    movw    r3, #:lower16:BlockDiagram_DiscreteDerivative_previousValue
    movt    r3, #:upper16:BlockDiagram_DiscreteDerivative_previousValue
    ldr     r2, [r7, #28]
    str     r2, [r3, #0]
    add     r7, r7, #60
    mov     sp, r7
    pop     {r7}
    bx      lr
```

Generated PID controller code
CFLAGS: -O0
102 lines of code!

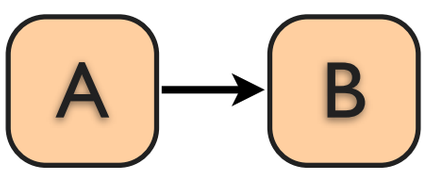
Code Optimization (cont.)

```
BlockDiagram_execute:
    movw        r3, #:lower16:.LANCHOR0
    movt        r3, #:upper16:.LANCHOR0
    push        {r4, r5, r6, r7}
    ldr         r2, [r0, #4]
    ldr         r5, [r0, #0]
    ldr         r4, [r3, #0]
    ldr         r6, [r3, #4]
    movw        r0, #:lower16:274877907
    subs        r2, r5, r2
    lsl         ip, r4, #8
    movt        r0, #:upper16:274877907
    smull       r7, r5, r0, ip
    subs        r6, r2, r6
    mov         r7, #2048000
    mul         r6, r7, r6
    asr         ip, ip, #31
    lsls        r7, r2, #10
    rsb         r5, ip, r5, asr #17
    asr         ip, r7, #8
    add         r5, ip, r5
    sbfx        r0, r6, #6, #24
    adds        r0, r5, r0
    add         r4, ip, r4
    str         r0, [r1, #0]
    str         r2, [r3, #4]
    str         r4, [r3, #0]
    pop         {r4, r5, r6, r7}
    bx                      lr
```

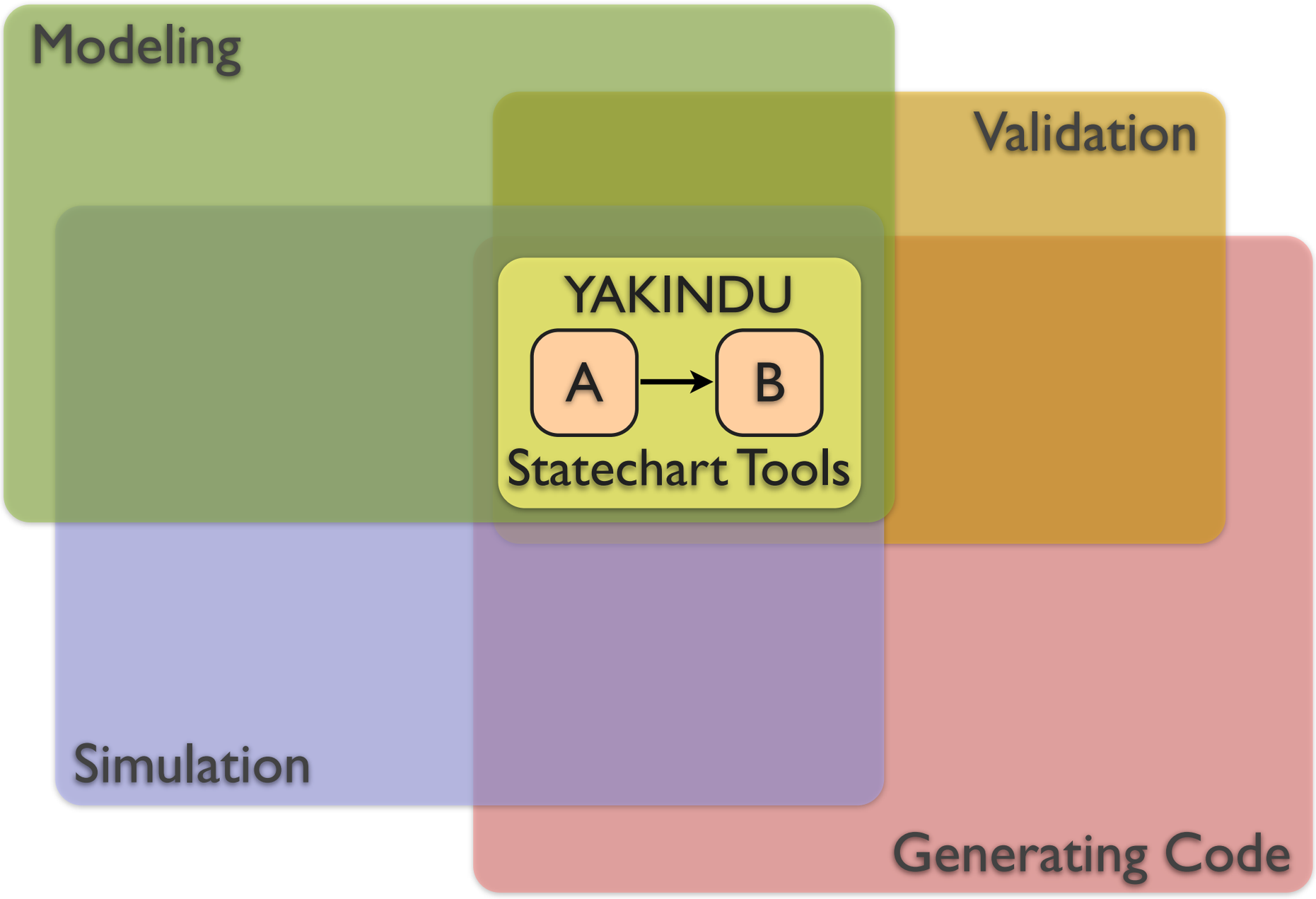
Generated PID controller code
CFLAGS: -O2
28 lines of code!

Statecharts

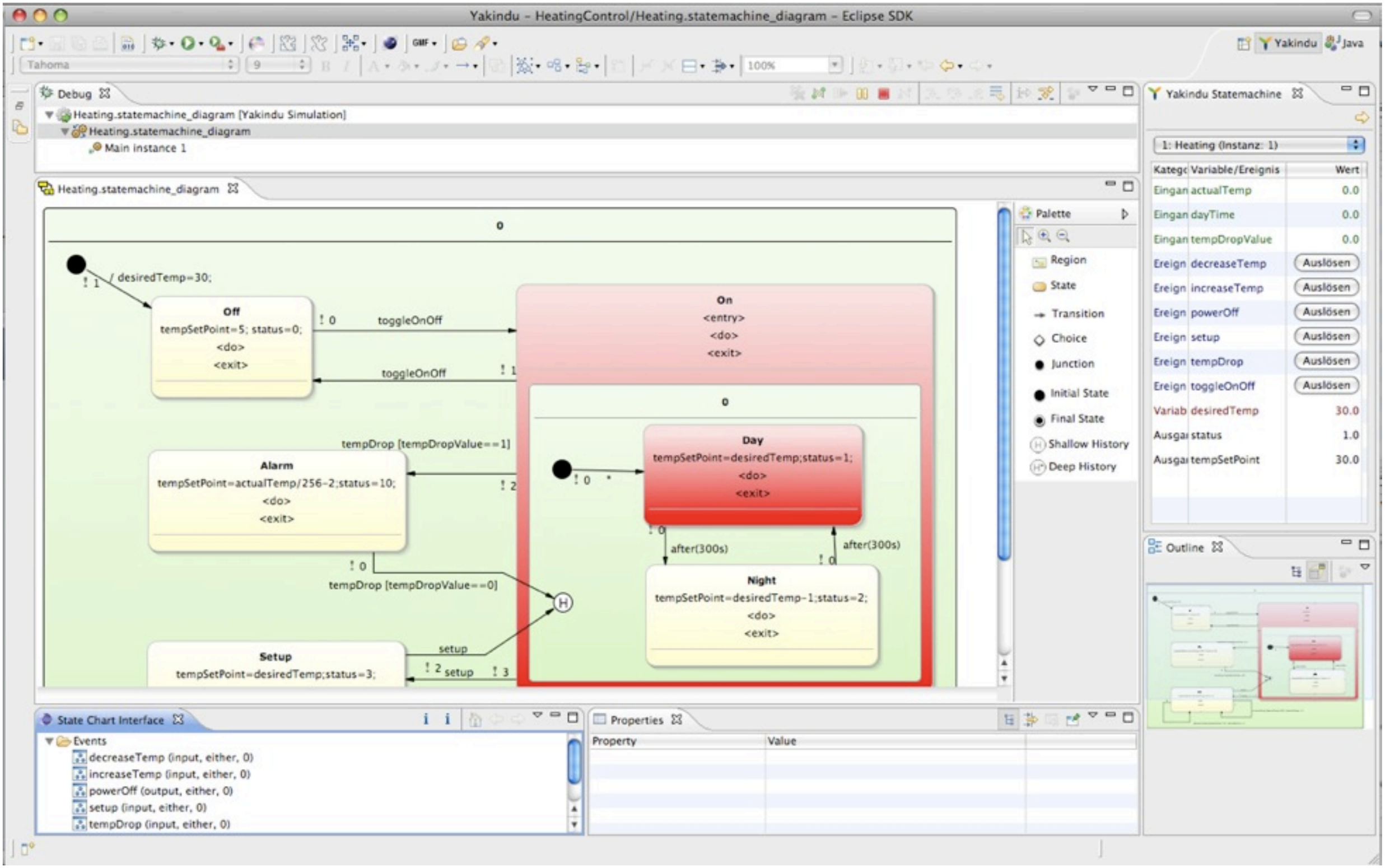
What are YAKINDU Statechart Tools ?



What are YAKINDU Statechart Tools ?



YAKINDU Statechart Tools

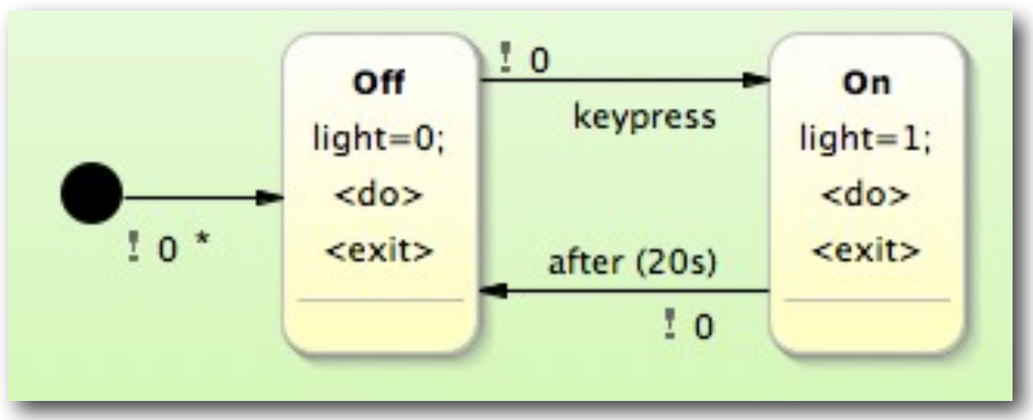


state charts at a glance

- YAKINDU is based on Statecharts as defined by David Harel (like UML)

- well defined formalism
- extension on deterministic finite state machines
- executable

- describes behaviour based on
  - states
  - transitions
  - actions



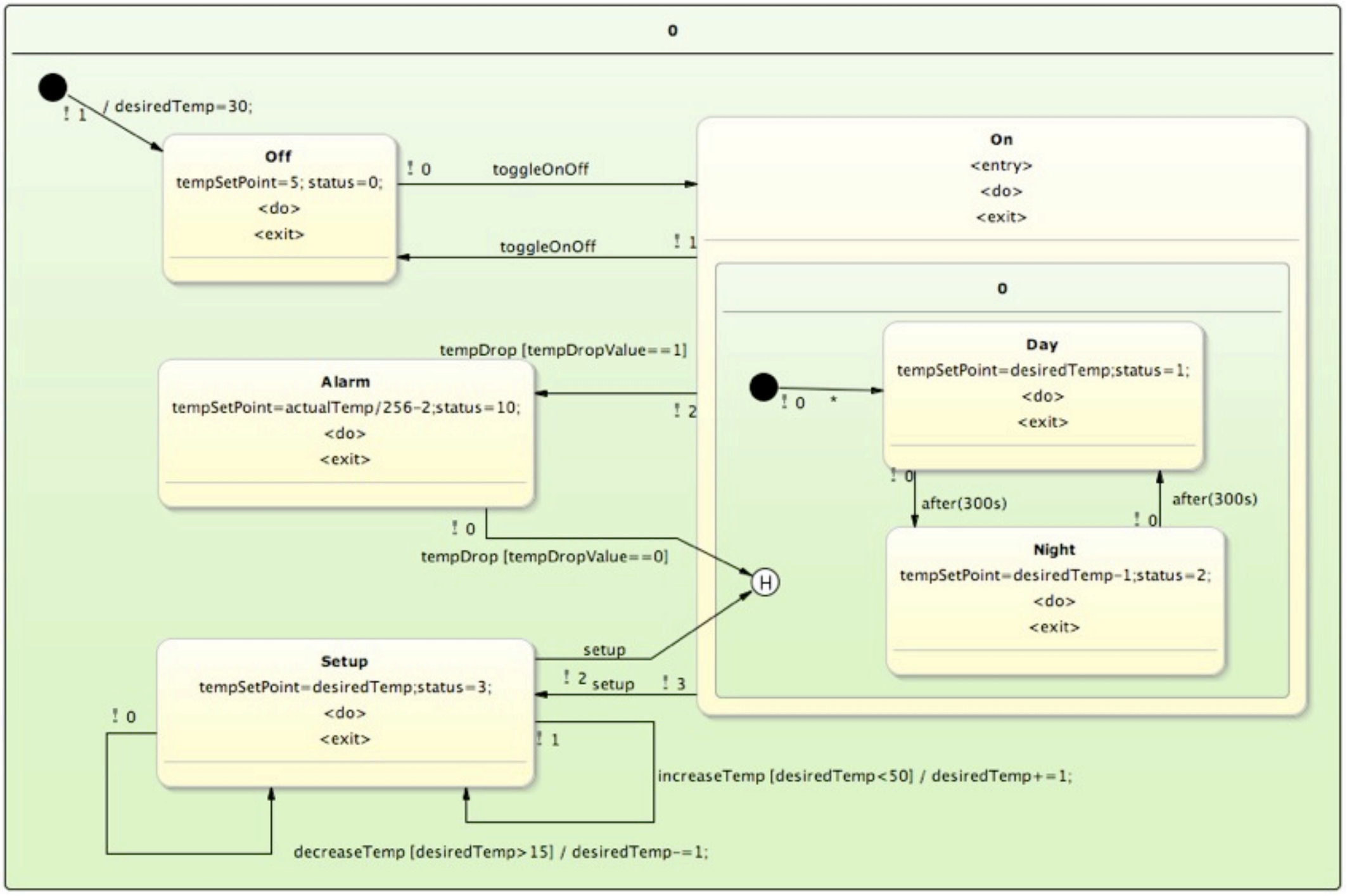
Finite State Machines

- A system is defined by a finite number of states
- The behaviour of the system depends on the current state
  - behaves differently to events depending on the state
- The current state is determined by the history of the state machine

- Mealy machine (1955)
  - Actions are associated with transitions

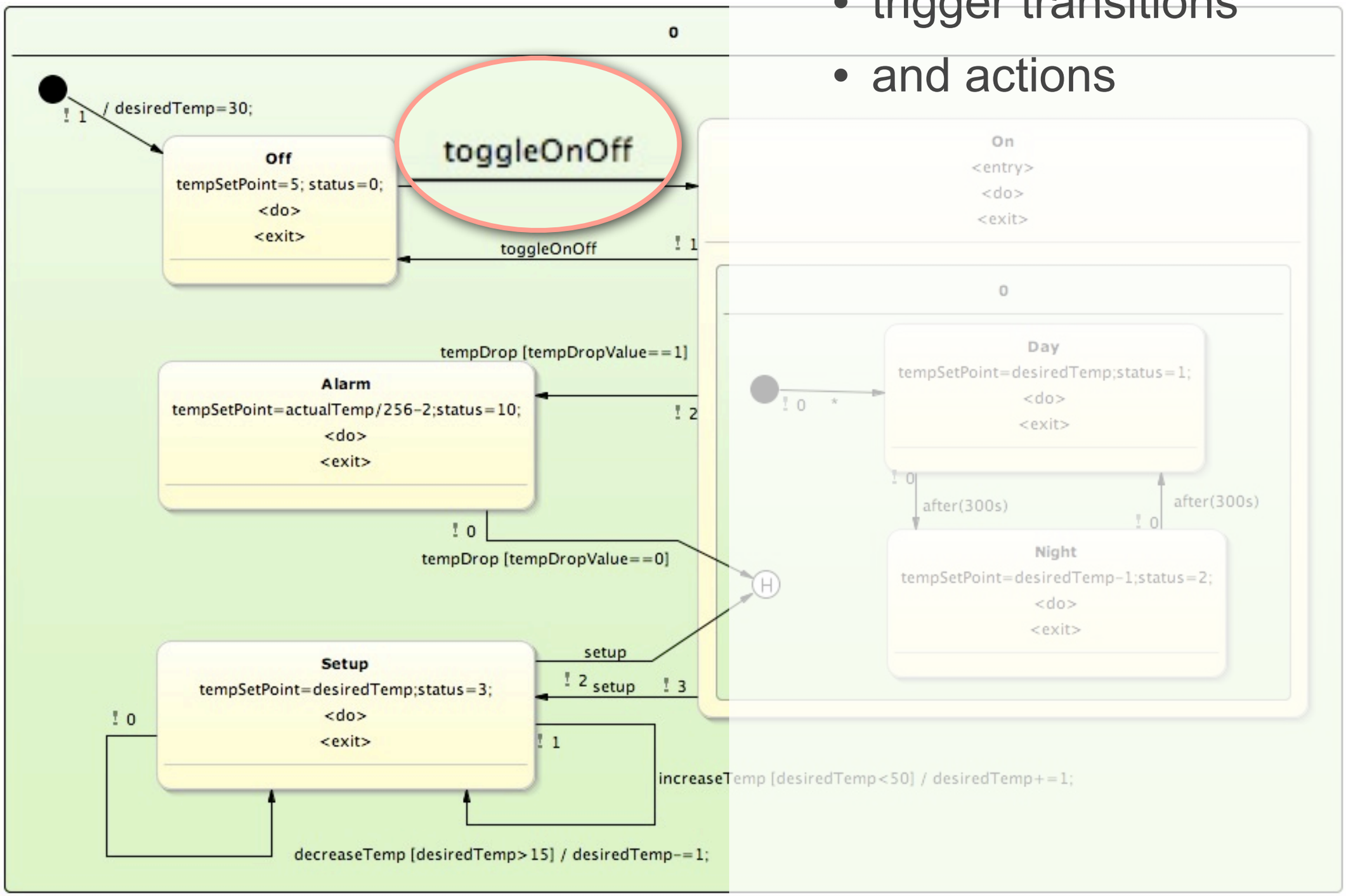
- Moore machine (1956)
  - Actions are associated with states (entries)

an example ...

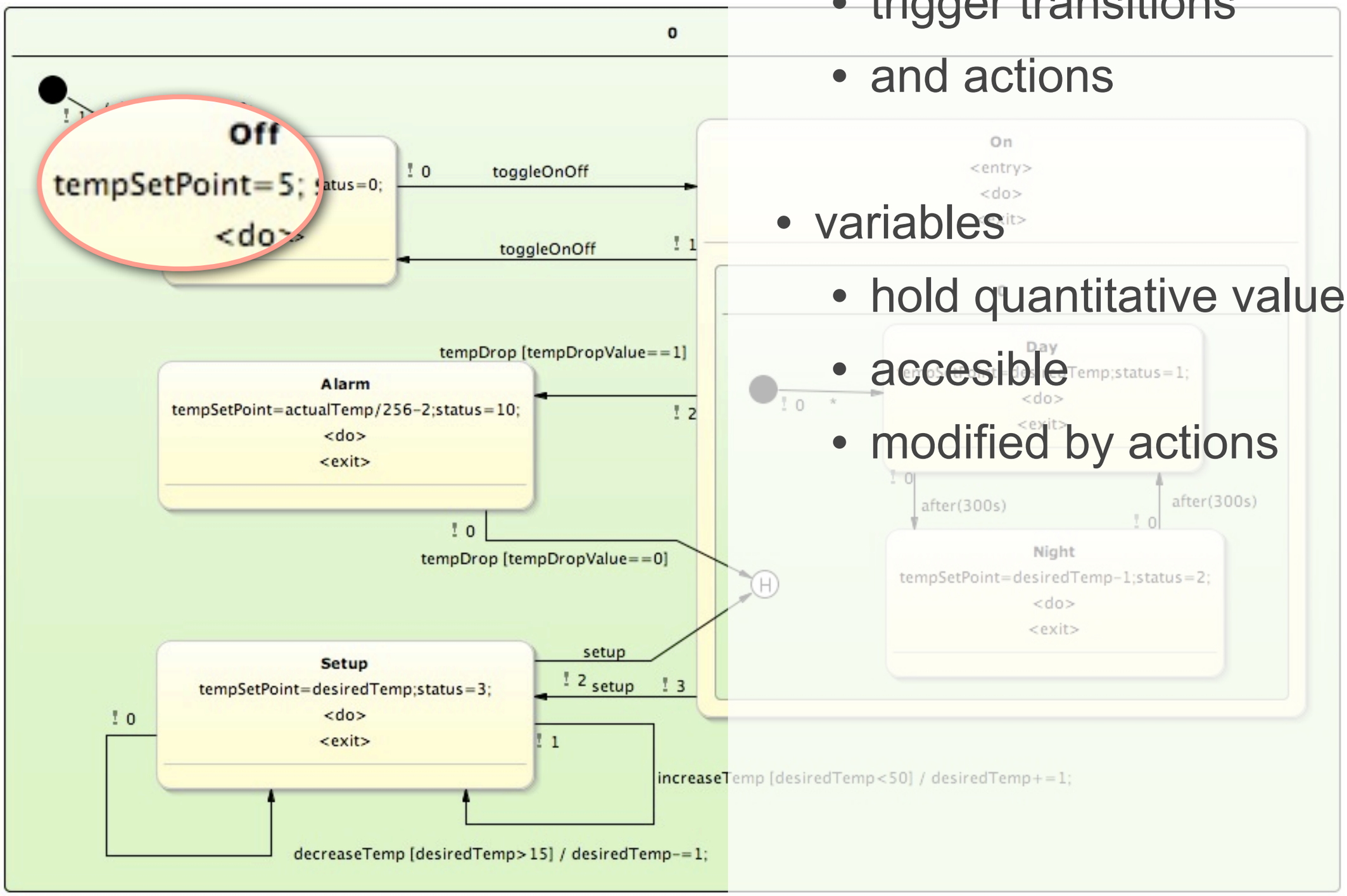

0
! 1
/ desiredTemp=30;
Off
tempSetPoint=5; status=0;
<do>
<exit>
! 0
toggleOnOff
toggleOnOff
! 1
On
<entry>
<do>
<exit>
0
tempDrop [tempDropValue==1]
Alarm
tempSetPoint=actualTemp/256-2;status=10;
<do>
<exit>
! 2
! 0
*
Day
tempSetPoint=desiredTemp;status=1;
<do>
<exit>
! 0
after(300s)
after(300s)
! 0
Night
tempSetPoint=desiredTemp-1;status=2;
<do>
<exit>
! 0
tempDrop [tempDropValue==0]
H
setup
! 2
setup
! 3
Setup
tempSetPoint=desiredTemp;status=3;
<do>
<exit>
! 0
! 1
increaseTemp [desiredTemp<50] / desiredTemp+=1;
decreaseTemp [desiredTemp>15] / desiredTemp-=1;

interface concepts

- events
  - trigger transitions
  - and actions

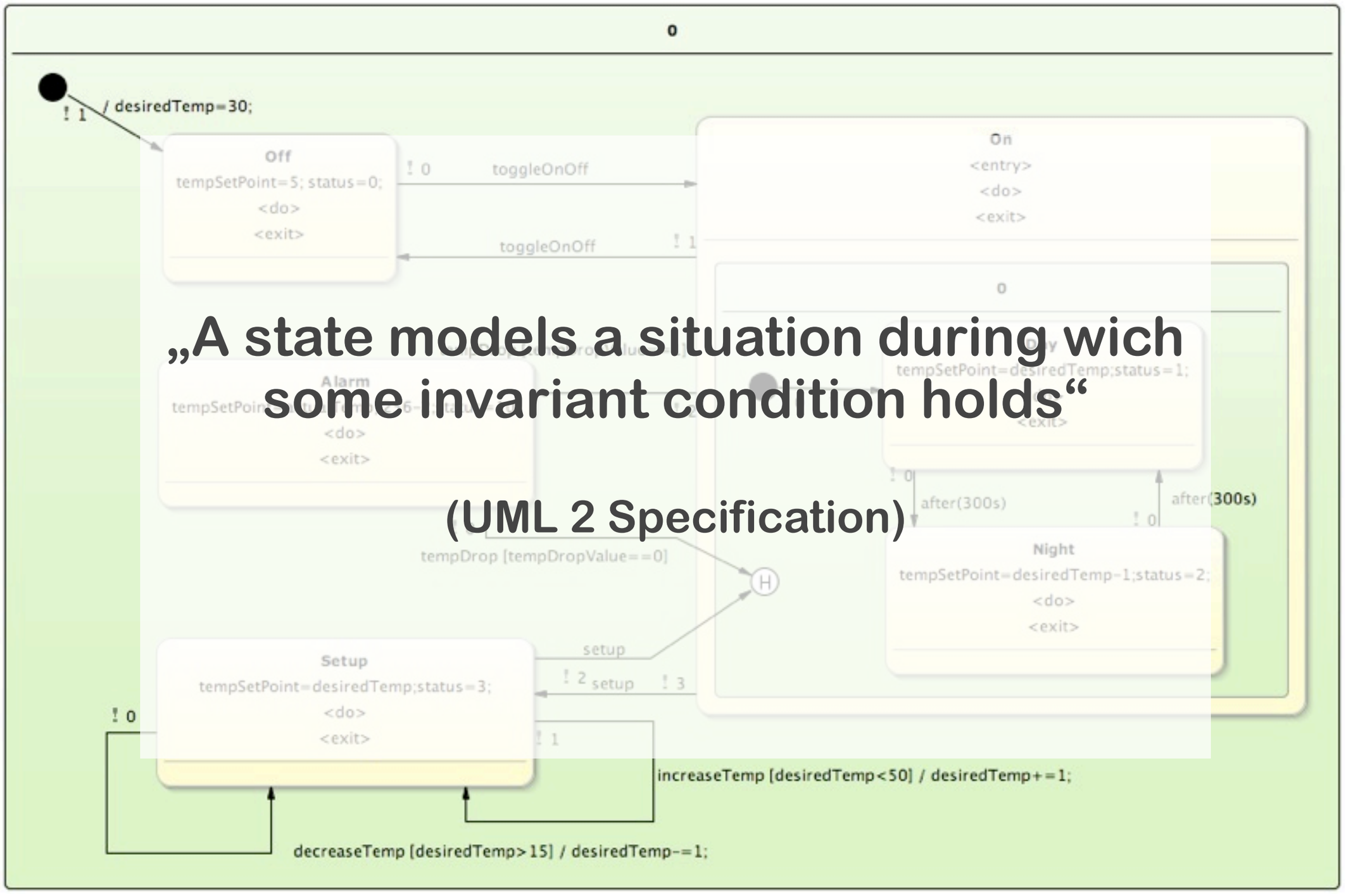


interface concepts



- events
  - trigger transitions
  - and actions
- variables
  - hold quantitative values
  - accesible
  - modified by actions

states and transitions

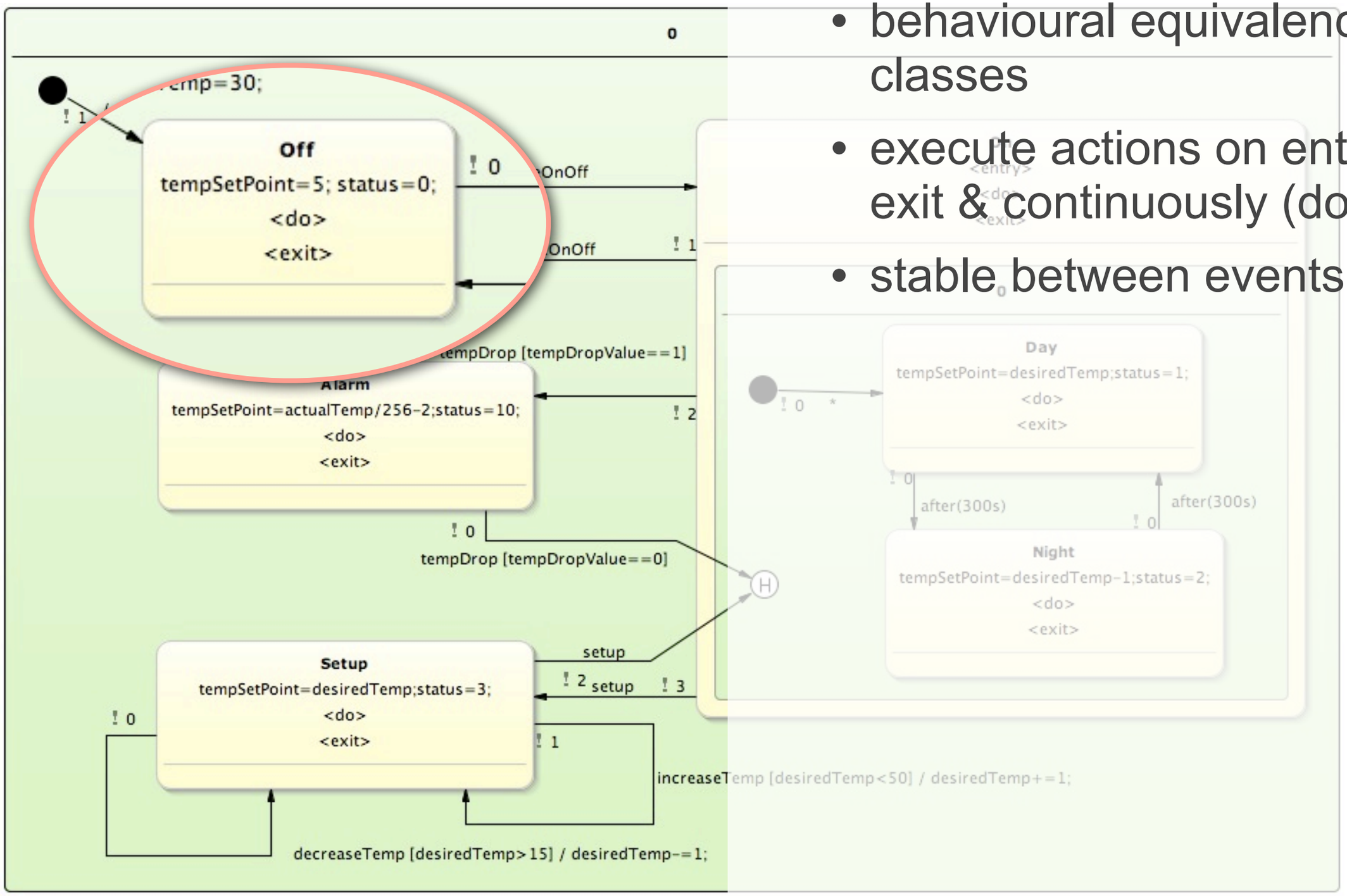


„A state models a situation during wich some invariant condition holds“

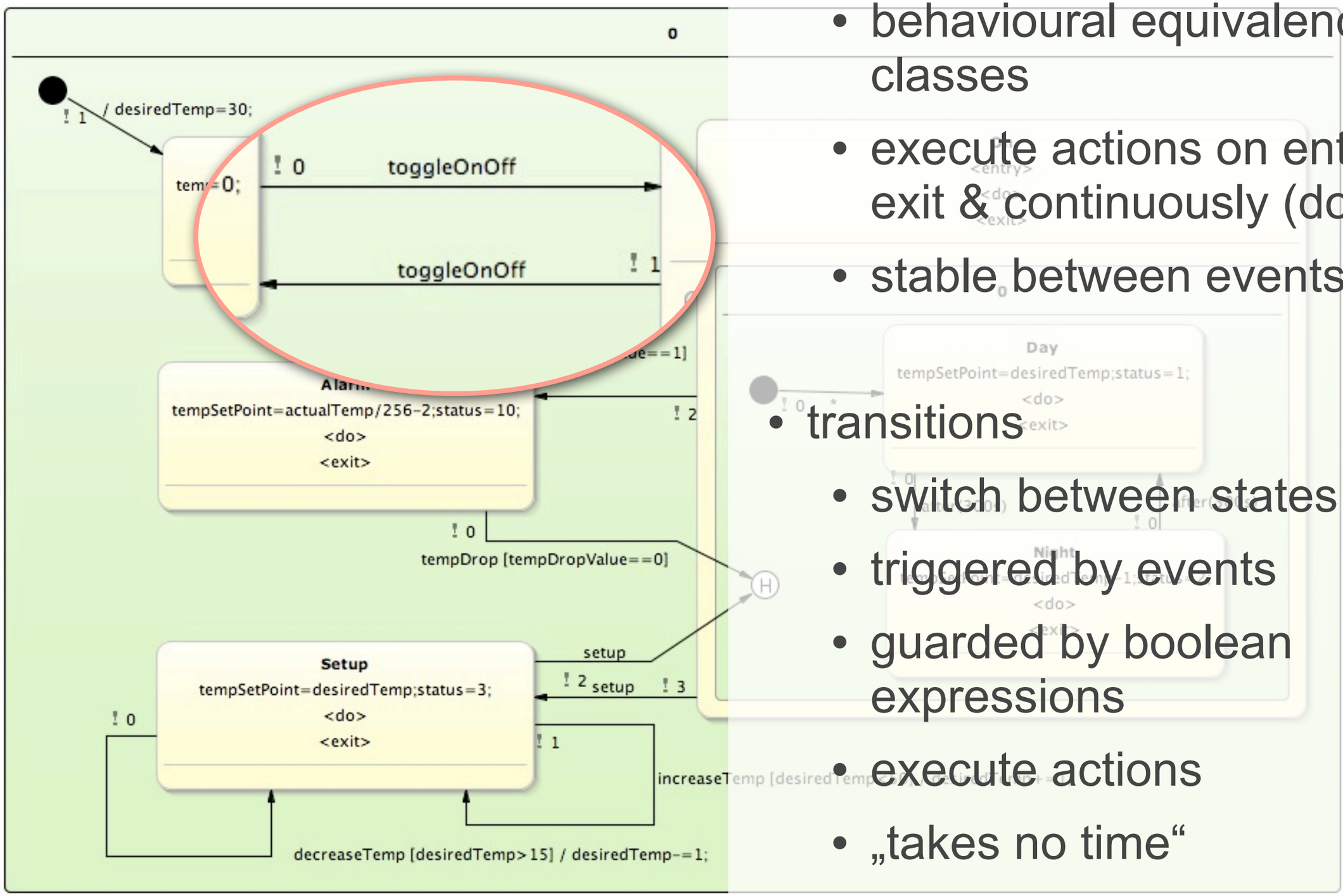
(UML 2 Specification)

states and transitions

- states
  - behavioural equivalence classes
  - execute actions on entry, exit & continuously (do)
  - stable between events

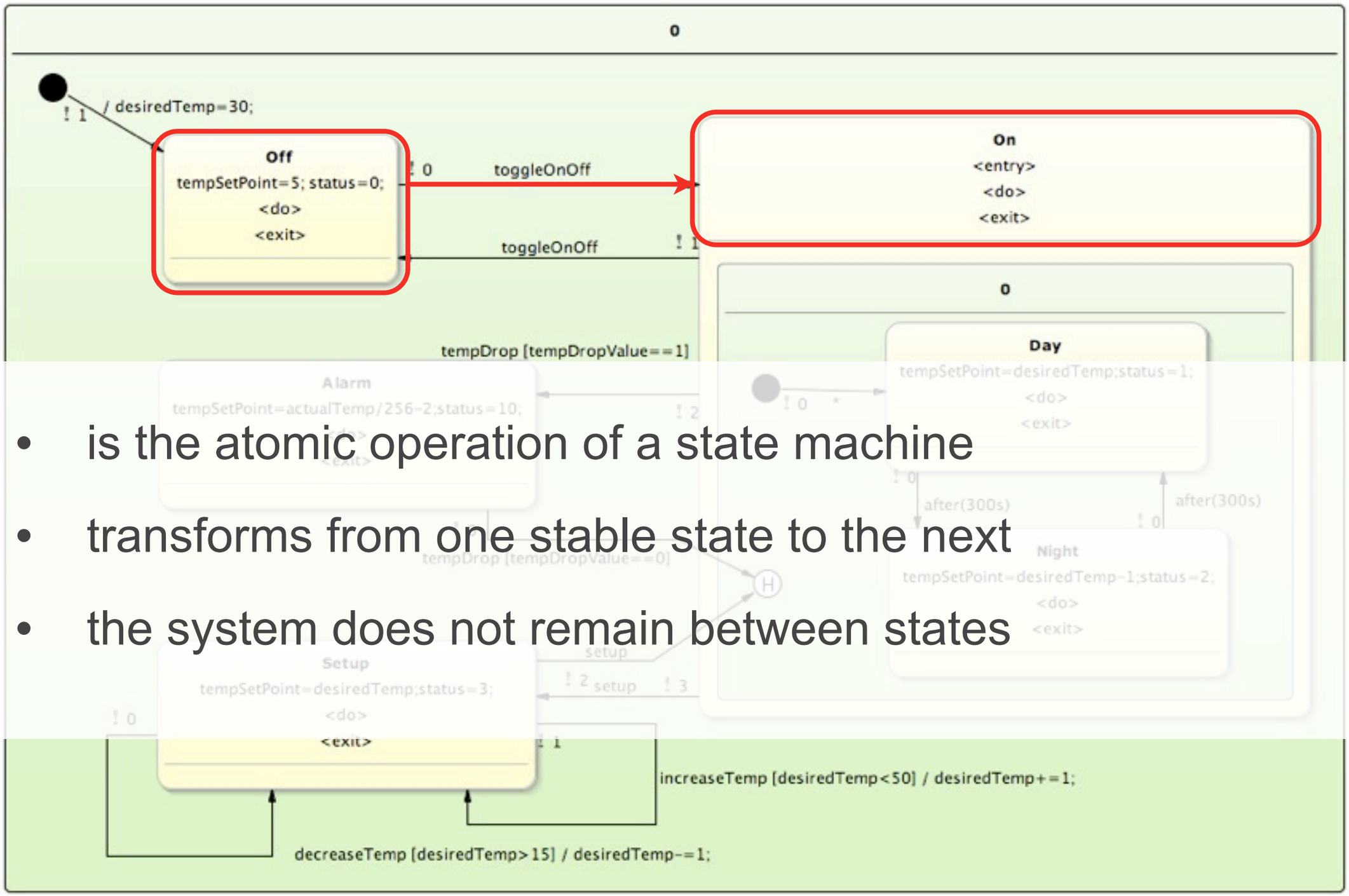


states and transitions



- states
  - behavioural equivalence classes
  - execute actions on entry, exit & continuously (do)
  - stable between events
- transitions
  - switch between states
  - triggered by events
  - guarded by boolean expressions
  - execute actions
  - „takes no time“

run to completion step



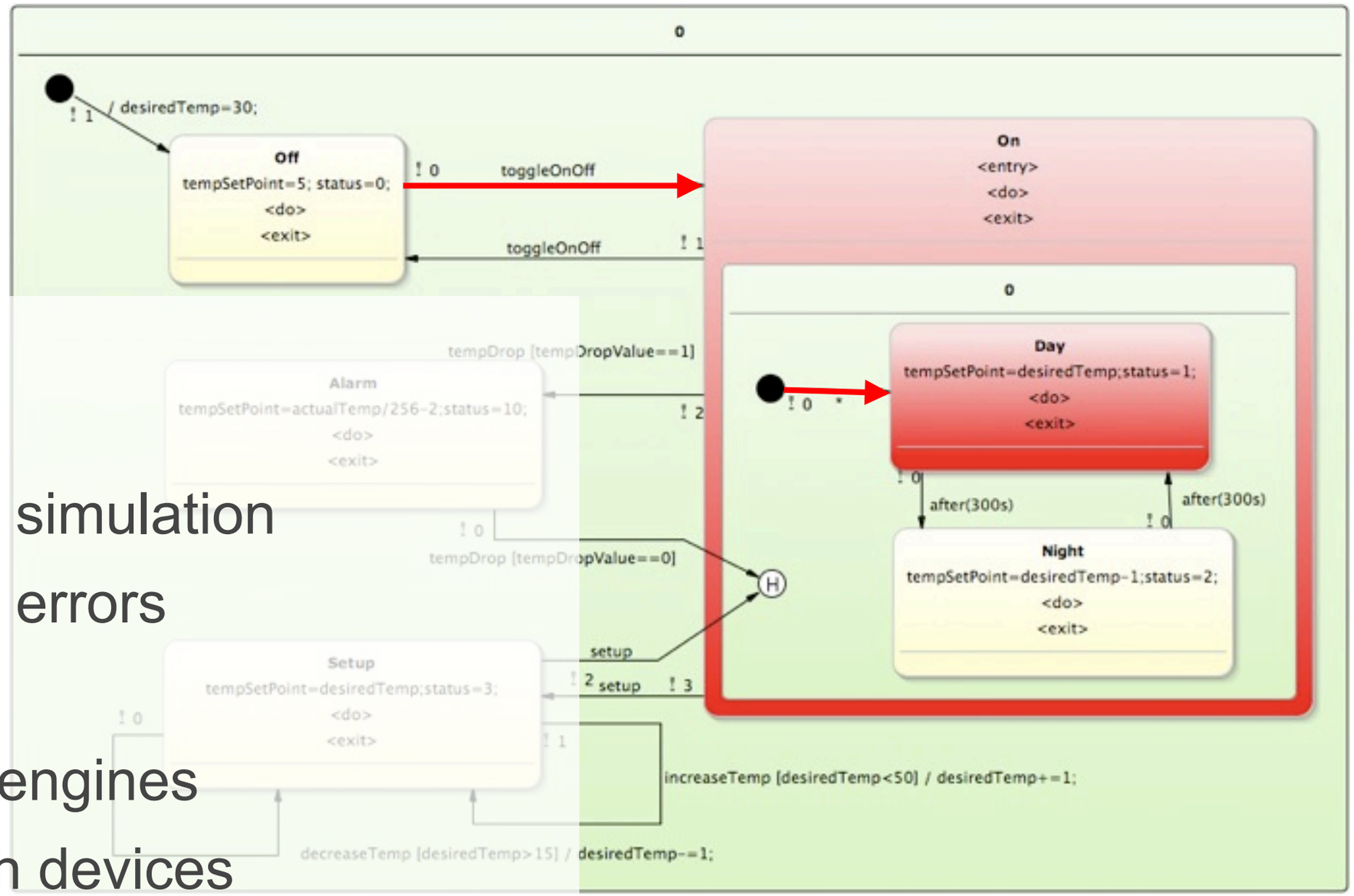
- is the atomic operation of a state machine
- transforms from one stable state to the next
- the system does not remain between states

statechart semantics

- using an own, simple meta model
- close to UML state machines
- but:
  - YSCs are self contained with an interface well defined by events and variables
  - core execution semantics are cycle-driven and not event-driven
    - allows processing concurrent events
    - event driven behaviour can be defined on top
  - time is an abstract concept for statecharts
  - time control is delegated to the environment
- model interpreter and different flavours of generated code follow the same core semantics

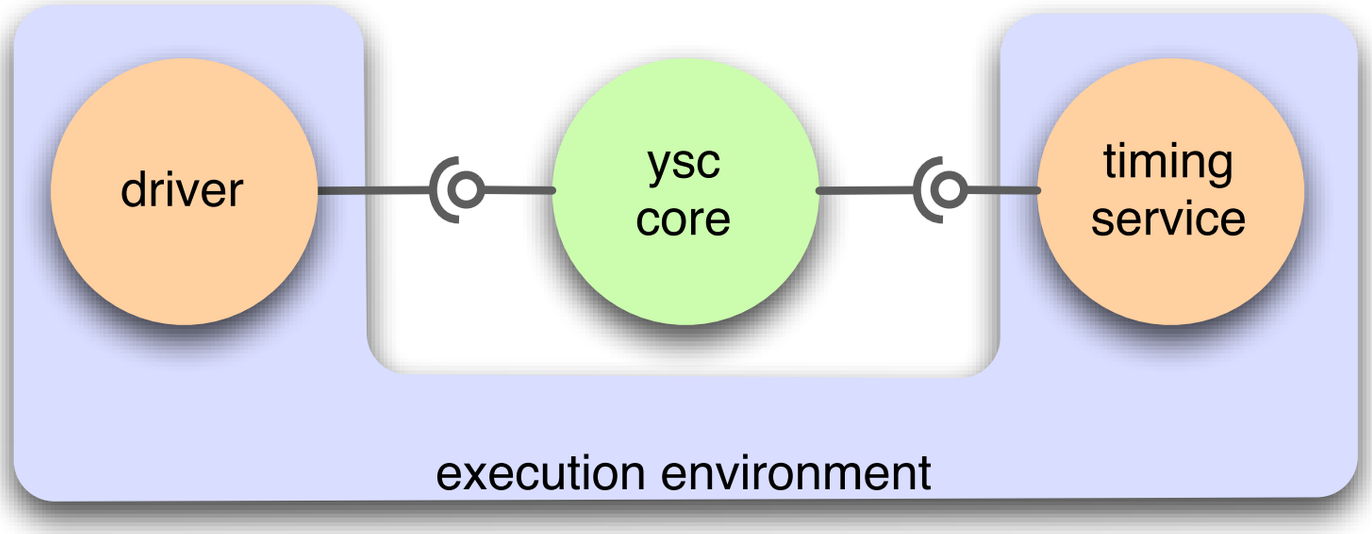
simulation

- model interpreter
- allows interactive simulation
- helpful for finding errors
- API
  - for simulation engines
  - interacting with devices
  - visualising device state
- currently no real debug capability



generating code

- target languages C, Java, PLC
- generated core code is runtime platform agnostic
- defines interfaces towards execution environment



- no black box code generators - open & extendible
- parts of the execution environment can be generated by project specific generator extensions

Textual Models

The notion of models is typically associated with graphical representation

The main property of models:

abstraction

graphical representation is just the notation

text can capture the identical concepts

textual modeling

… is typically used for specialized modeling languages

… these are also known as **Domain Specific Languages**

domain specific languages(DSLs)

> *„A domain-specific programming language (domain-specific language, DSL) is a programming language designed to be useful for a specific set of tasks. This is in contrast to general-purpose programming language (general-purpose language, GPL), such as C or Java, or general-purpose modeling languages like UML."*
>
> *Quelle: Wikepedia*

- Why DSLs?
  - Focus on the relevant aspects of a language
  - Apply the notion of the domain ⇒ more comprehensive for domain experts
  - Optimization by specialization
  - Use of modeling concepts that GPLs (e.g. UML) do not support

⇒ Important for the success of model driven development

an simple example ...

```
monitor motorTemperatur fault {
      min 0
      max 150
      delay 1000
  }
```

Where Xtext comes into play

- Allows you to define textual DSLs easily
- Builds ready-to-use tool-chain
- Allows customizing and tweaking
- Durable foundation with Java Eclipse EMF
- Integrates well into the Eclipse modeling world
- Strong community

easy!

defining an own language

```
Model:
    monitors+=Monitor*;

Monitor:
    'monitor' name=ID severity=Severity '{'
        'min' minValue=INT
        'max' maxValue=INT
        'delay' delay=INT
    '}';

enum Severity: failure | warning;
```

Xtext Features

- Generates the following artefacts from the grammar definition:
  - Incremental parser and lexer (ANTLR-3 based) - reads textual models.
  - Serializer - saving textual models.
  - Linker - manages cross document/model references
  - EMF Ecore meta model(s)
  - EMF Resource implementation - integrates the parser into the resource framework.
  - Eclipse IDE integration:
    - Eclipse Workbench UI Editor:
      - Syntax Coloring
      - Model Navigation (F3)
      - Code Completion
      - Outline View Adaption
    - M2T-Xpand based code generator (stub)

textual vs. graphical

textual models

… are sequential

…good for

…structures

…decalarative models

…feels like code

…easy to edit

…integrates seamlessly with SCM

…low development costs

graphical models

…are 2 dimensional

…good for

…visualizing connected structures (graphs)

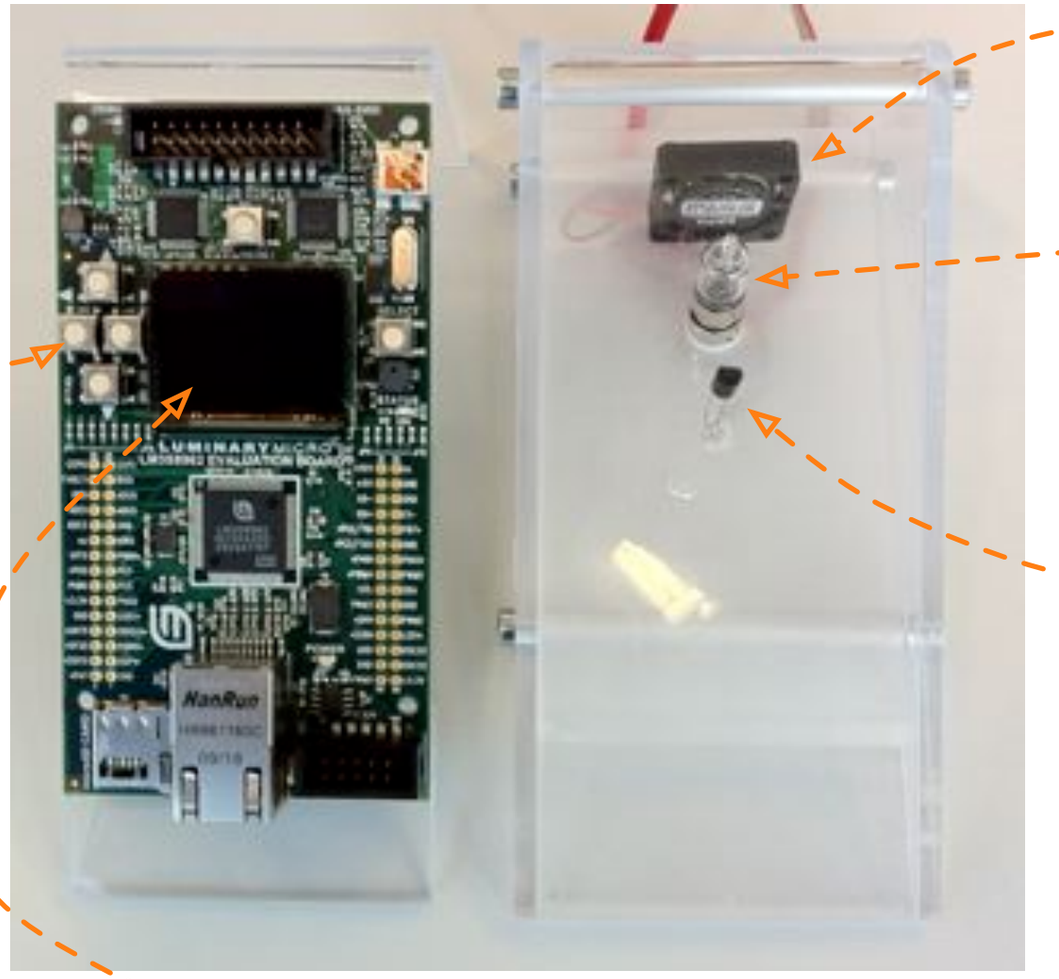
…usability heavily depends on tool

…diff & merge must be supported by tool

…high development costs

Integrating
all into an
Example Application

… a Heating Control



Fan

Heater

Temperature Sensor

Display

Button Panel

µC: ARM Cortex M3
(Luminary Micro LM3S8962)

RTOS: FreeRTOS

Target Language: C

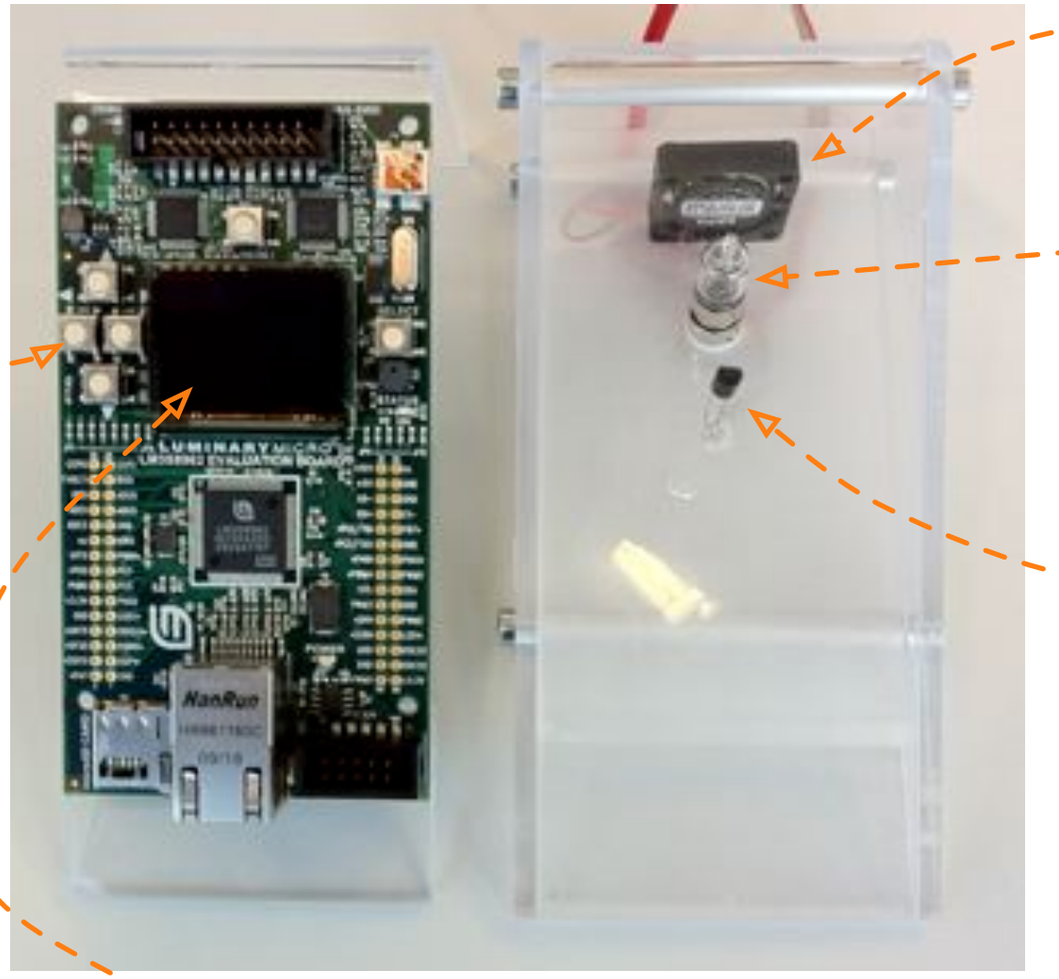
Heating Control - Requirements

- The user can switch on and off the heating control using a button
- The user can define the temperature setpoint using the buttons.
- The heating control controls the temperature of the environment continuously
- In the case of a temperature drop the control must switch off the heater
- The system displays the current temperature, the temperature setpoint, the current heating power, and the current status of the system

Heating Control - System Properties

- Reactive: system continuously interacts with the environment
- System is event-driven (asynchronous inputs)
- System continuously processes data from the environment (isochronous inputs)
- Non deterministic environment
- State dependent: the state of the system evolves over time depending on previous inputs and time
- Real Time

Heating Control - Software Structure

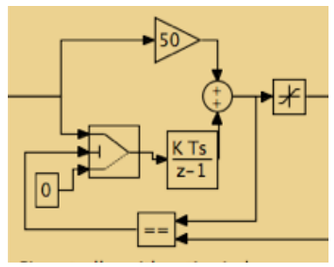


Fan

Heater

Temperature Sensor

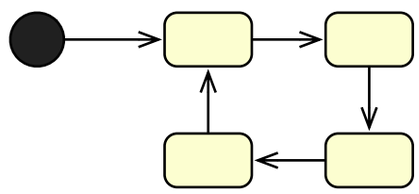
HeatingController

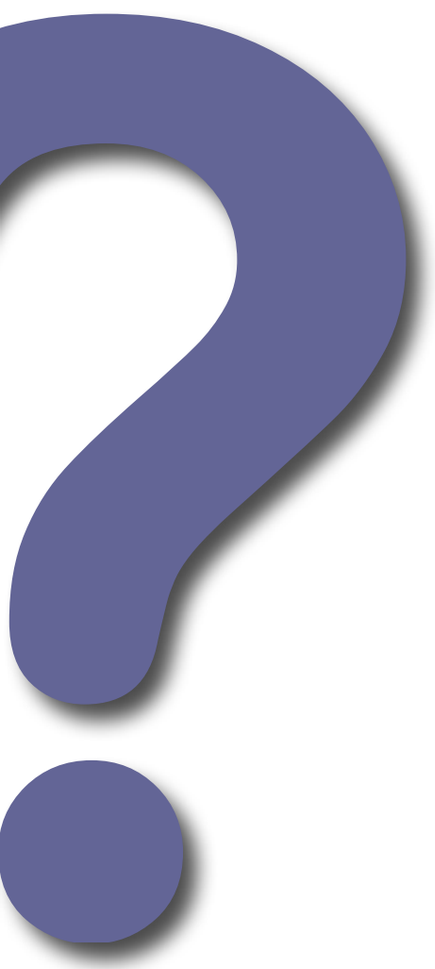


Display

Button Panel

State Control

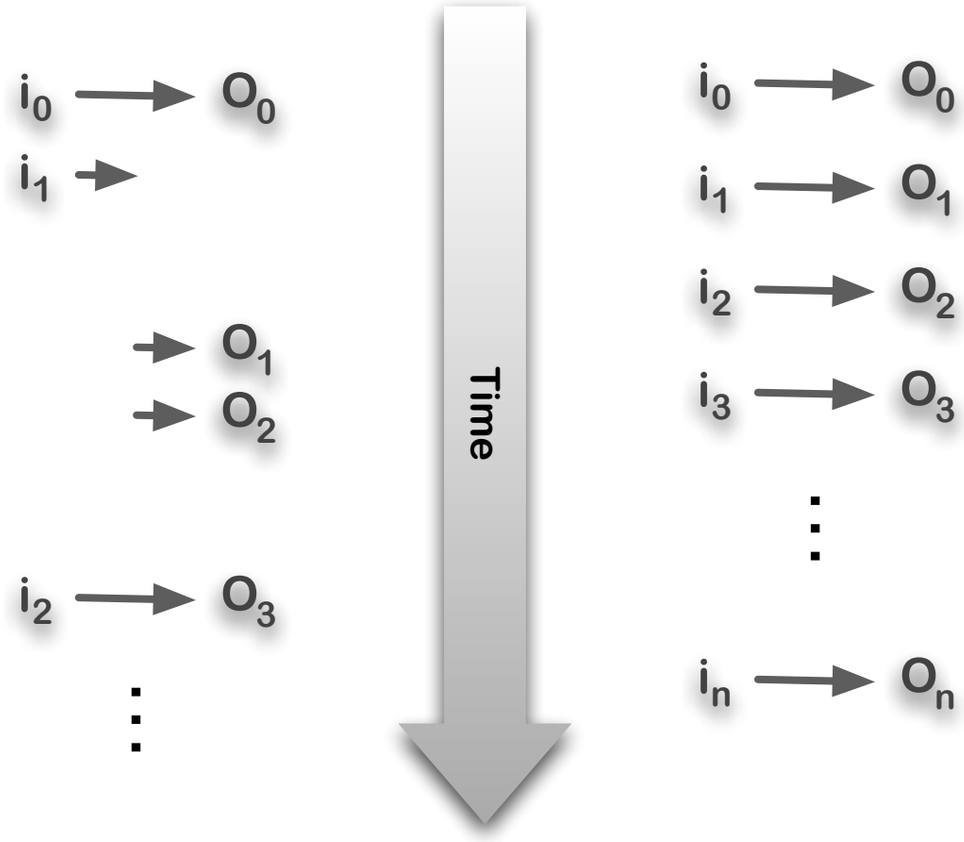




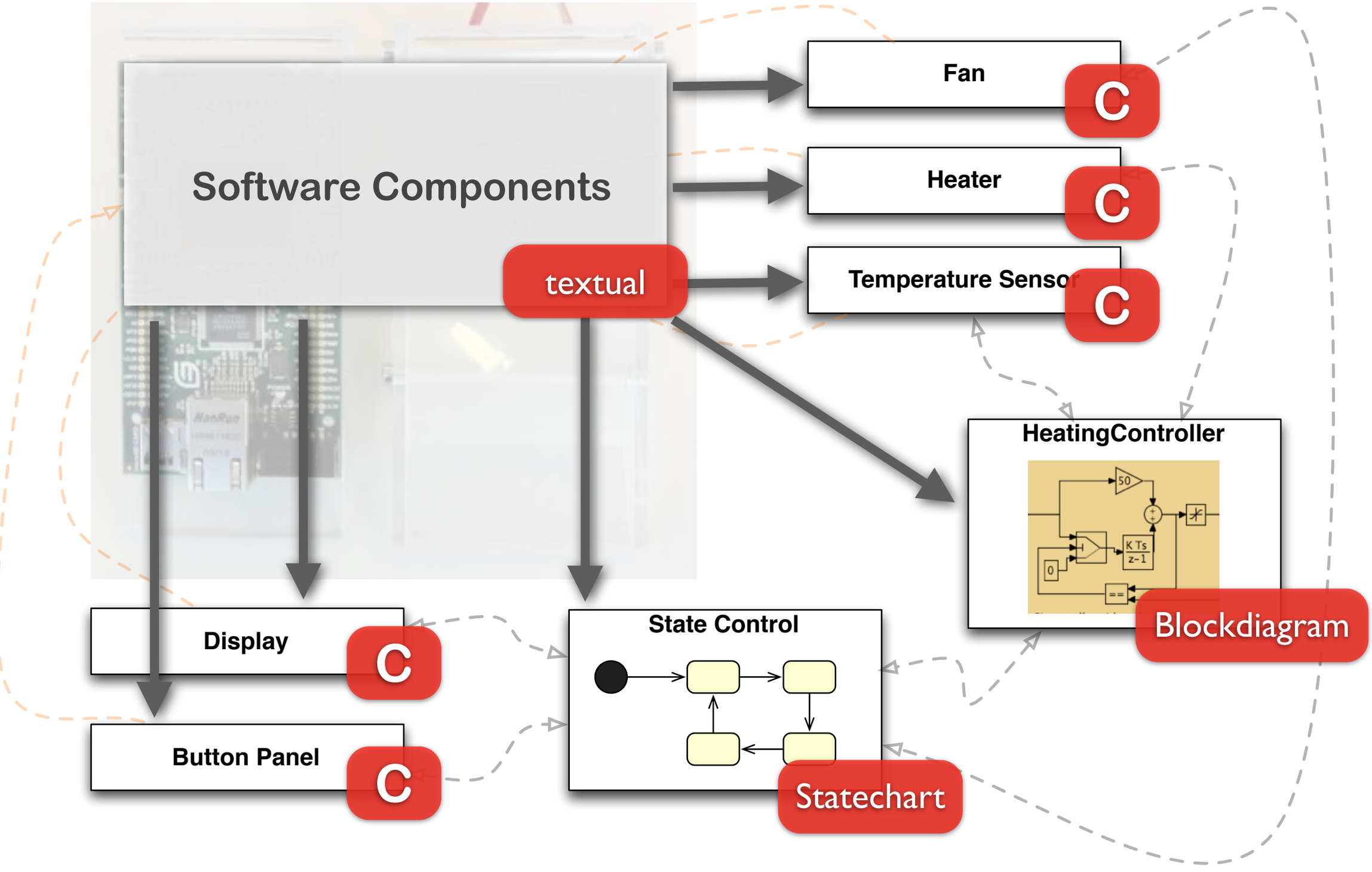
Statecharts vs. Blockdiagrams

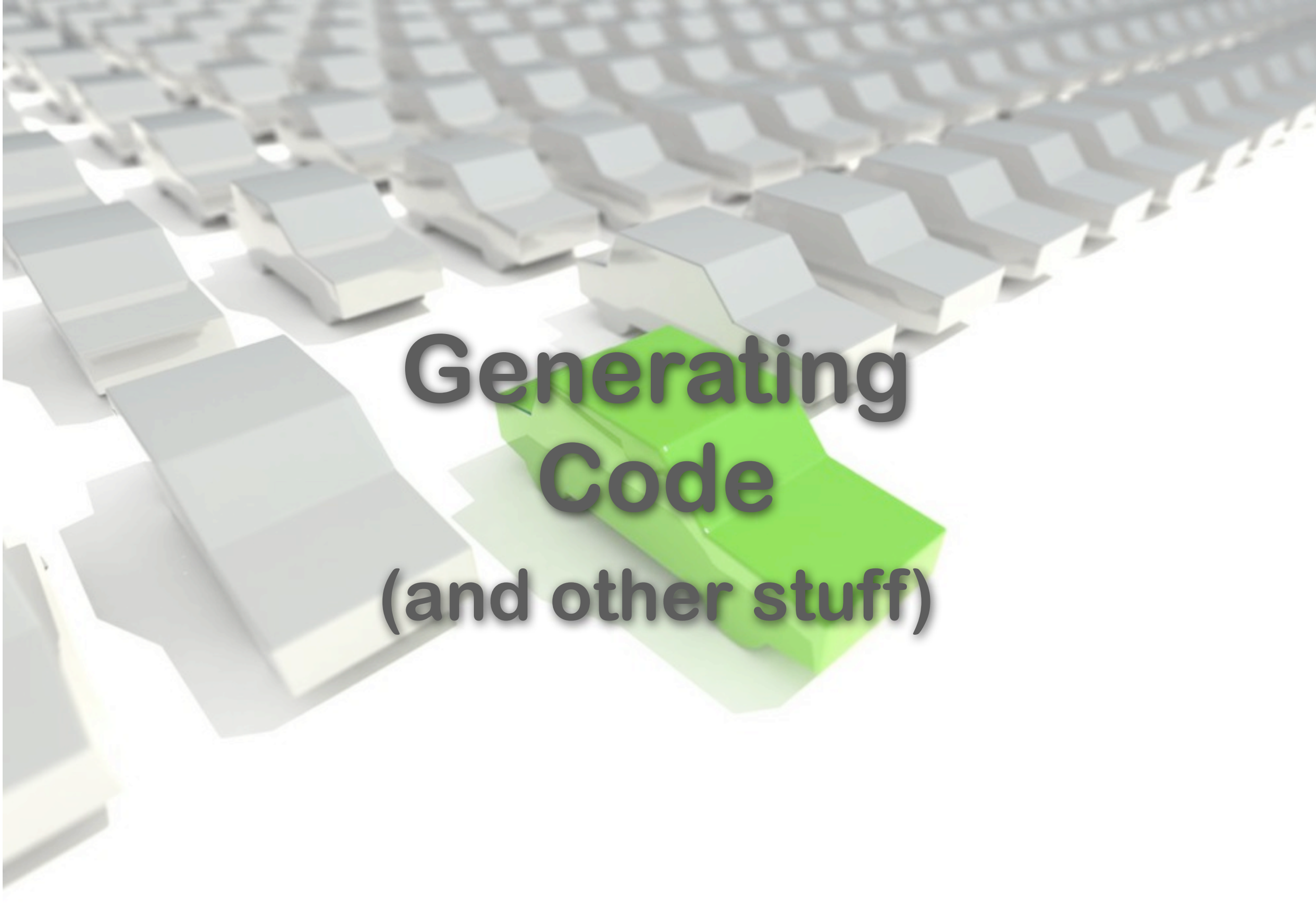
What is the difference?

Statecharts	Block Diagrams
model reactive systems	model dynamic systems
event driven	data flow centric
focuses on transition of the systems state and ist reactions	focuses on transformation of a flow of input values into a flow of output values
typically asynchronous	typically isochronous



Heating Control - Software




Generating
Code
(and other stuff)

Reusing Solutions

- code generators capture solution knowledge
- for each construct in a model the generator knows an implementation pattern
- variants of generators capture variants of solutions

Black Box Of The Shelf Code Generators

… are bound to a specific modeling language - mostly a GPL

… are bound to a method

… are bound to a specific (set of) target platform

… have limited capability for customization

If there is a Black Box Generator that fits well, then buy !!

Individual Code Generators

black box code generators may not fit due to

… specialized methodology DSLs or UML based

… specific need regarding target platform

→ then individual generators must be built

Building Individual Code Generators

Approach

- … derive the generator from a reference implementation
- … choose a template based approach that makes building the generator itself easy

… defining the reference implementation is the engineering task

Creating Xpand Templates

derive template from reference implementation …

```
class System {
public:

    Scheduler* scheduler;

    InductionLoopComponent* byRoadInduction;
    TrafficLightComponent* byRoadLight;
    TrafficLightComponent* mainRoadLight;
    CrossingControllerComponent* controller;
    TrafficLightControllerComponent* byRoadLightController;
    TrafficLightControllerComponent* mainRoadLightController;

    System(Scheduler& scheduler);
    virtual ~System();

    void connect();
    void raiseTimerEvent(void* handle);
};
```

variable code

Creating Xpand Templates (2)

```
class System {
public:

    Scheduler* scheduler;

    InductionLoopComponent* byRoadInduction;
    TrafficLightComponent* byRoadLight;
    TrafficLightComponent* mainRoadLight;
    CrossingControllerComponent* controller;
    TrafficLightControllerComponent* byRoadLightController;
    TrafficLightControllerComponent* mainRoadLightController;

    System(Scheduler& scheduler);
    virtual ~System();

    void connect();
    void raiseTimerEvent(void* handle);
};
```

```
«DEFINE Class FOR System»
class System {
public:

    Scheduler* scheduler;

«FOREACH instances AS instance -»
    «instance.type.typeName()»* «instance.memberName()»;
«ENDFOREACH»

    System(Scheduler& scheduler);
    virtual ~System();

    void connect();
    void raiseTimerEvent(void* handle);
};
«ENDDEFINE»
```

variabler Code

Questions & Comments