TTK4130 Modeling and Simulation

Lecture 1

TTK4130 Modeling and Simulation

To learn:

- Formulate mathematical models from first principles
- Simulate models using computer

Main purpose: Control system design/testing/validation, hence *dynamic* modeling

Instructors:

- Lecturer: Leif Erik Andersson
- Teaching assistant: Mikkel Eske Nørgaard Sørensen (D351 A)
  - 4 student assistants

Course Information

- All course information is provided on Itslearning. There will be no handout of material
- We will not cover the complete curriculum in our lectures; rather focus on the most important and/or difficult parts

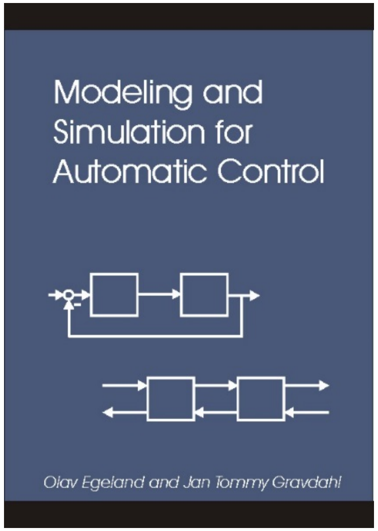
- Lectures:
  - Tuesdays 10:15-12:00 (KJL5)
  - Thursdays 8:15-10:00 (EL3)

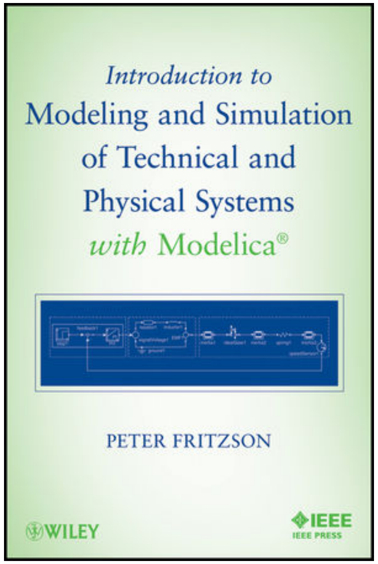
- Exam:
  - May 15th, 2017
  - Examination support code: A (Open book)

- Grading: Exam counts 100%

Syllabus

- Books:
  - "Modeling and Simulation for Automatic Control", 2002, by O.Egeland and J.T. Gravdahl (ISBN 82-92356-00-2) **(E)**
    - Errata in itslearning
  - "Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica, 2011, by Peter Fritzson **(F)**





Lecture schedule

Week	Date	Theme	Literature
2	10.01	Introduction to Modelica	F: 1, 2
	12.01	More introduction. State-space models, transfer functions. Modeling software, network models.	E: 1.1-1.3, 2.1-2.2 (E:1.4-1.5)
3	17.01	Energy functions, passivity	E: 2.3-2.4
	19.01	More passivity	E: 2.4
4	24.01	Modeling of complex systems. Simulation: Order, test system	F: 3, 4, E: 14.1-14.2
	26.01	Explicit Runge-Kutta methods	E: 14.3-14.4
5	31.01	Electrical motors	E: 3.1-3.4
	02.02	Implicit Runge-Kutta methods	E: 14.5
6	07.02	Stability, Padé approximations	E: 14.6
	09.02	Stability, frequency properties, automatic step size adjustment	E: 14.6-14.7
		Implementation, BDF and differential-algebraic systems	E: 14.8, 14.11, 14.12
7	14.02	Hydraulic motors, transmission lines	E: 4.1-4.6
	16.02	Friction	E: 5
8	21.02	Vectors, dyadics, rotation matrices	E: 6.1-6.4
	23.02	Euler angles, angle axis	E: 6.5-6.6
9	28.02	Euler parameters, angular velocities	E: 6.7-6.8
	02.03	Kinematic differential equations	E: 6.9
10	07.03	Kinematics of a rigid body , Newton-Euler equations of motion	E: 6.12-6.13, 7.3
	09.03	Newton-Euler equations of motion, Modelica.Multibody	E: 7.3
11	14.03	Lagrange equations of motion	E: 7.7, 8.1-8.2
	16.03	Lagrange equations of motion, recap, examples	
12	21.03	Process modelling and balance laws, I	E: 10.4, 11.1-4 (+ slides)
	23.03	Process modelling and balance laws, II	E: 10.4, 11.1-4 (+ slides)
13	28.03	Guest lecture: Erlend Kristiansen, Comsol Multiphysics	
	30.03	No lecture (excursion)	
14	04.04	No lecture (excursion)	
	06.04	No lecture (excursion)	
15	11.04	No lecture (Easter)	
	13.04	No lecture (Easter)	
16	18.04	No lecture (Easter)	
	20.04	Topic: Not decided yet	
17	25.04	Topic: Not decided yet	
	27.04	No lecture (TTK4135 lab.rapport)	
18	02.05	Topic: Not decided yet	
	04.05	Topic: Not decided yet	

E: “Modeling and Simulation for Automatic Control” by O. Egeland and J.T. Gravdahl
F: “Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica” by P. Fritzon

Course information, assignments

- A minimum number of exercises (8 of 11) must be approved to enter the final examination

- The deadlines for all assignments are absolute
  - Schedule:
    - Out: Latest Wednesdays (week before delivery)
    - In: Tuesdays 7 days later – 12:00

Exercise Schedule

Assignments	Upload	Deadline
1	11/1-17 at 12.00 (noon)	17/1-17 at 12.00 (noon)
2	18/1-17 at 12.00 (noon)	24/1-17 at 12.00 (noon)
3	25/1-17 at 12.00 (noon)	31/1-17 at 12.00 (noon)
4	1/2-17 at 12.00 (noon)	7/2-17 at 12.00 (noon)
5	8/2-17 at 12.00 (noon)	14/2-17 at 12.00 (noon)
6	15/2-17 at 12.00 (noon)	21/2-17 at 12.00 (noon)
7	22/2-17 at 12.00 (noon)	7/3-17 at 12.00 (noon)
8	1/3-17 at 12.00 (noon)	8/3-17 at 12.00 (noon)
9	8/3-17 at 12.00 (noon)	14/3-17 at 12.00 (noon)
10	15/3-17 at 12.00 (noon)	28/3-17 at 12.00 (noon)
11	29/3-17 at 12.00 (noon)	25/4-17 at 12.00 (noon)

Exercise class:

Exercise	Exercise hours	Room
1	16/1-17 8.00-12.00	G124-G128 and G134
2	23/1-17 8.00-12.00	G124-G128 and G134
3	30/1-17 8.00-12.00	G124-G128 and G134
4	6/2-17 8.00-12.00	G124-G128 and G134
5	13/2-17 8.00-12.00	G124-G128 and G134
6	20/2-17 8.00-12.00	G124-G128 and G134
7	27/2-17 8.00-12.00	G124-G128 and G134
8 (7)	6/3-17 8.00-12.00	G124-G128 and G134
9	13/3-17 8.00-12.00	G124-G128 and G134
10	20/3-17 8.00-10.00	KJL2
10	27/3-17 8.00-10.00	KJL2
11	24/4-17 8.00-10.00	KJL2

Exercise Schedule

Assignments	Upload	Deadline
1	11/1-17 at 12.00 (noon)	17/1-17 at 12.00 (noon)
2	18/1-17 at 12.00 (noon)	24/1-17 at 12.00 (noon)
3	25/1-17 at 12.00 (noon)	31/1-17 at 12.00 (noon)
4	1/2-17 at 12.00 (noon)	7/2-17 at 12.00 (noon)
5	8/2-17 at 12.00 (noon)	14/2-17 at 12.00 (noon)
6	15/2-17 at 12.00 (noon)	21/2-17 at 12.00 (noon)
7	22/2-17 at 12.00 (noon)	7/3-17 at 12.00 (noon)
8	1/3-17 at 12.00 (noon)	8/3-17 at 12.00 (noon)
9	8/3-17 at 12.00 (noon)	14/3-17 at 12.00 (noon)
10	15/3-17 at 12.00 (noon)	28/3-17 at 12.00 (noon)
11	29/3-17 at 12.00 (noon)	25/4-17 at 12.00 (noon)

Exercise class:

Exercise	Exercise hours	Room
1	16/1-17 8.00-12.00	G124-G128 and G134
2	23/1-17 8.00-12.00	G124-G128 and G134
3	30/1-17 8.00-12.00	G124-G128 and G134
4	6/2-17 8.00-12.00	G124-G128 and G134
5	13/2-17 8.00-12.00	G124-G128 and G134
6	20/2-17 8.00-12.00	G124-G128 and G134
7	27/2-17 8.00-12.00	G124-G128 and G134
8 (7)	6/3-17 8.00-12.00	G124-G128 and G134
9	13/3-17 8.00-12.00	G124-G128 and G134
10	20/3-17 8.00-10.00	KJL2
10	27/3-17 8.00-10.00	KJL2
11	24/4-17 8.00-10.00	KJL2

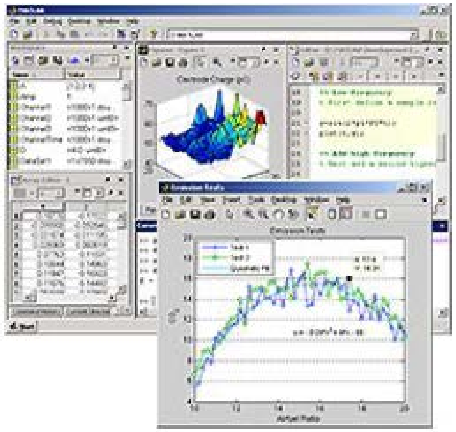
Course information, assignments

- A minimum number of exercises (8 of 11) must be approved to enter the final examination

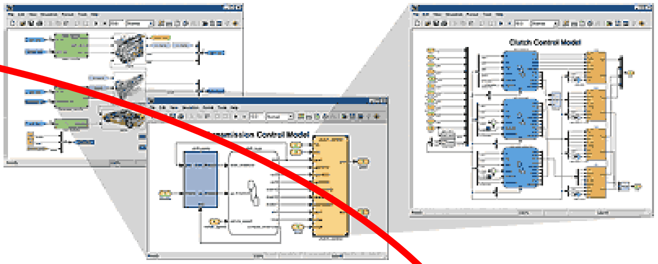
- The deadlines for all assignments are absolute
  - Schedule:
    - Out: Latest Wednesdays (week before delivery)
    - In: Tuesdays 7 days later – 12:00
  - One exercise class with assistants present ahead of the deadline for every assignment (Mondays 8:00-12:00, G124-G128 & G134)
  - Several exercises require computers
    - Matlab/Simuling & Dymola
    - Computer lab G124-128 & G134 reserved Mondays 8-12

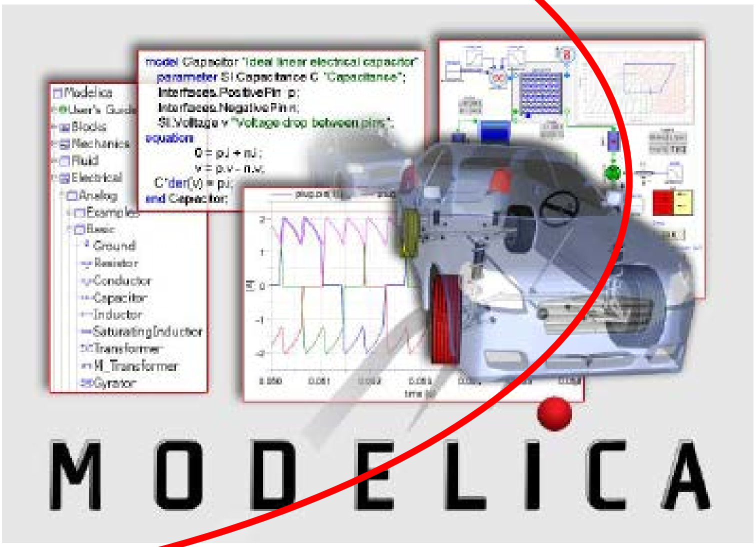
Software

- Matlab/Simulink
  - Computer labs
  - Your own computer
  - Some familiarity with Matlab/Simulink is assumed



- Dymola
  - Based on the Modelica modelling language
  - Installed on computer labs, but you can also use your PCs (Windows and Linux)
  - License:
    - License sever (via VPN), but limited number of licenses (50)
    - Much can be done with demo license





Introduction to Modelica

Slides taken/adapted from Peter Fritzson

History

- Development from autumn 1996 through person from industry and academics

- Modelica 1.0 in September 1997

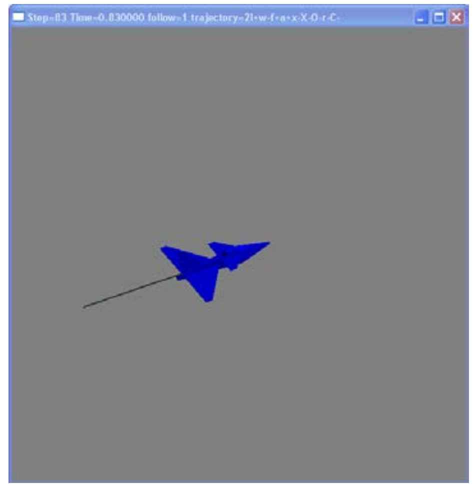
- At the moment: Modelica 3.3 (2012)

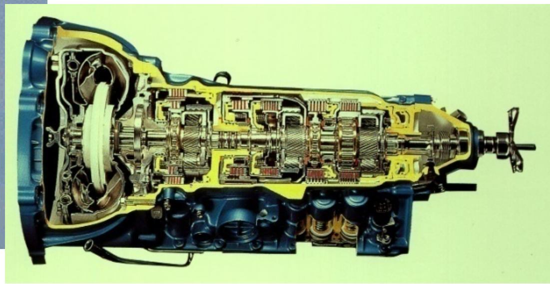
What is Modelica?

A language for modeling of complex physical systems

- Robotics
- Automotive
- Aircrafts
- Satellites
- Process systems
- Power plants
- Systems biology

MODELICA

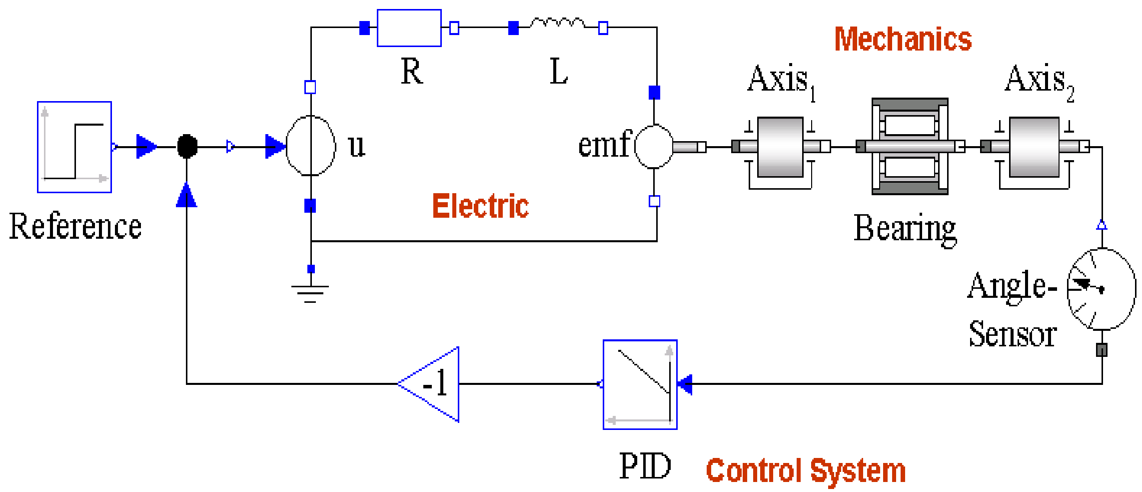




What is Modelica?

What is Modelica?

A language for modeling of complex physical systems



Primary designed for simulation, but there are also other usages of models, e.g. optimization.

What is Modelica?

A language for modeling of complex physical systems

i.e., Modelica is <u>not</u> a tool

Free, open language specification:


MODELICA

Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling

Language Specification

Version 2.2
February 2, 2005

by the

Modelica Association

Available at: http://www.modelica.org/

There exist several free and commercial tools, for example:

- OpenModelica from OSMC
- Dymola by Dassault systems / Dynasim
- SimulationX by ITI
- MapleSim by MapleSoft
- SystemModeler by Wolfram

Modelica technology

- Modelica is primarily an *equation-based* language

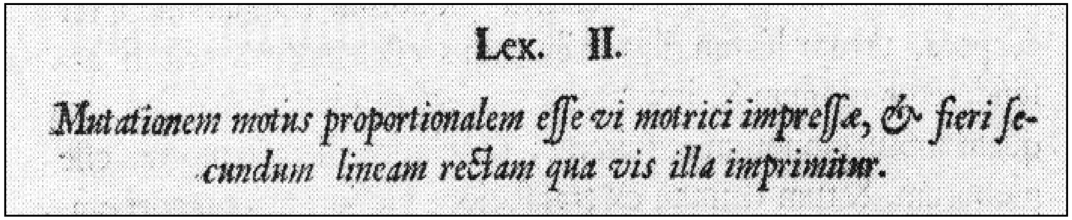
The Form – Equations

- Equations were used in the third millennium B.C.
- Equality sign was introduced by Robert Recorde in 1557

The Form – Equations

- Equations were used in the third millennium B.C.
- Equality sign was introduced by Robert Recorde in 1557

Newton still wrote text (Principia, vol. 1, 1686)



Lex. II.

Mutationem motus proportionalem esse vi motrici impressæ, & fieri secundum lineam rectam qua vis illa imprimitur.

“The change of motion is proportional to the motive force impressed”

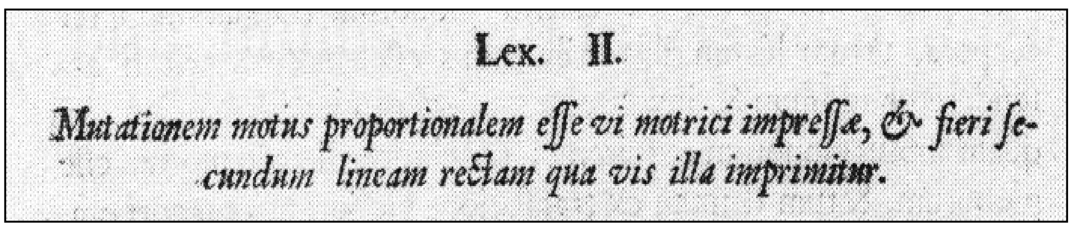
or

$$m \cdot a = \sum F$$

The Form – Equations

- Equations were used in the third millennium B.C.
- Equality sign was introduced by Robert Recorde in 1557

Newton still wrote text (Principia, vol. 1, 1686)



Lex. II.

Mutationem motus proportionalem eſſe vi motrici impreſſæ, & fieri ſecundum lineam rectam qua vis illa imprimitur.

Programming languages usually do not allow equations!

Do other programming languages not also use equations?

- Difference between *equation* and *assignment*

Do other programming languages not also use equations?

- Difference between *equation* and *assignment*

- Equations do not prescribe a certain data flow direction a execution order

Equation	Assignment
$R * i = v;$	$i := v/R;$ $v := R * i;$ $R := v/i;$

Do other programming languages not also use equations?

- Difference between *equation* and *assignment*

- Equations do not prescribe a certain data flow direction a execution order

Equation	Assignment
$R * i = v;$	$i := v/R;$ $v := R * i;$ $R := v/i;$
Causality unspecified	Output ⟵ Input

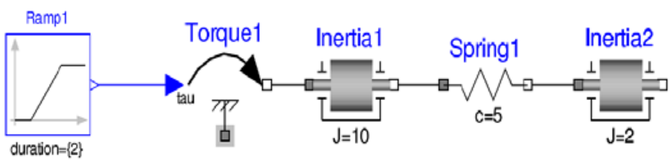
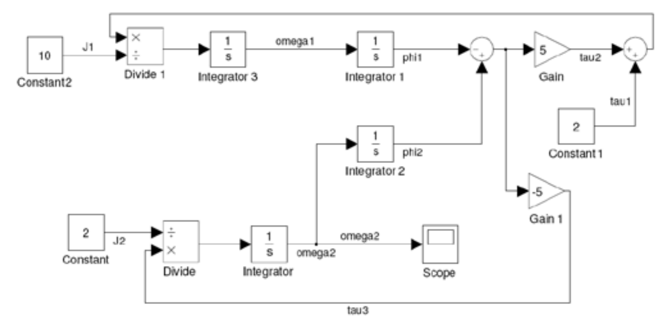
- → Acausal modeling

Advantages of Equations

- More flexible than assignments
- Key to physical modeling capabilities
- Increases reuse potential of Modelica classes/models

Acausal modeling

The order of computations is not decided at modeling time

<table>
<tr><th></th><th>Acausal</th><th>Causal</th></tr>
<tr><td>Visual Component Level</td><td>
</td><td>
</td></tr>
<tr><td>Equation Level</td><td>A resistor equation:
<code>R*i = v;</code></td><td>Causal possibilities:
<code>i := v/R;</code>
<code>v := R*i;</code>
<code>R := v/i;</code></td></tr>
</table>

Example – Simple differential equation

- Equation (ODE):
  - $\frac{dx}{dt} = ax \; ; \; x(0) = 1;$

Example – Simple differential equation

Equation (ODE):

$$\frac{dx}{dt} = -x, \quad x(0) = 1$$

Name of model

Initial condition

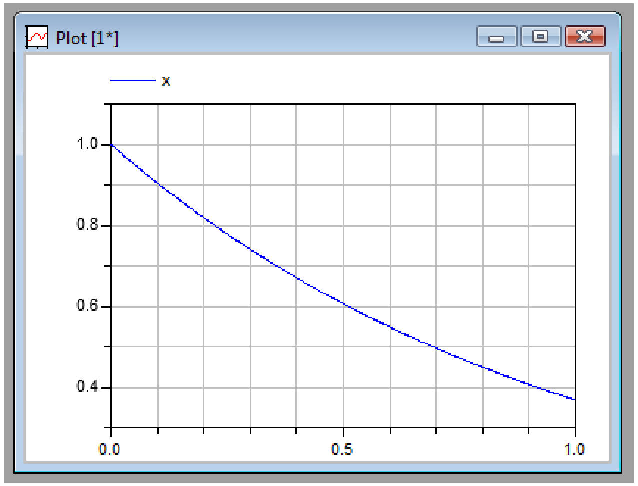
Continuous-time variable

Parameter, constant during simulation

```
model FirstOrder "A simple equation"
  Real x(start=1);
  parameter Real a = -1;
equation
  der(x)= a*x;
end FirstOrder;
```

Differential equation

Simulation in Dymola:



Modelica Variables and Constants

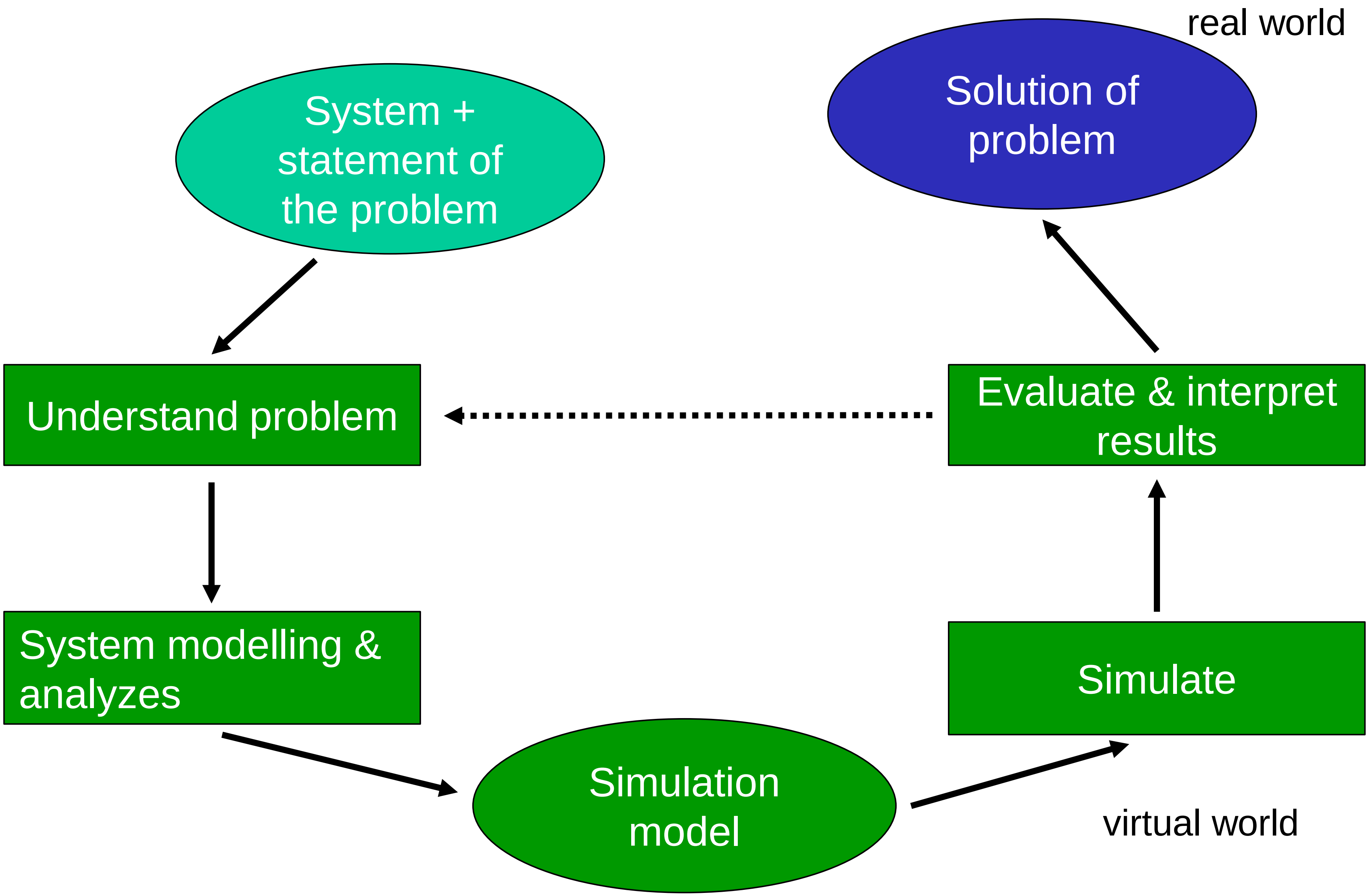
- Built-in primitive data types

Boolean	**true** or **false**
Integer	Integer value, e.g. **42** or **–3**
Real	Floating point value, e.g. **2.4e-6**
String	String, e.g. **“Hello world”**
Enumeration	Enumeration literal e.g. **ShirtSize.Medium**

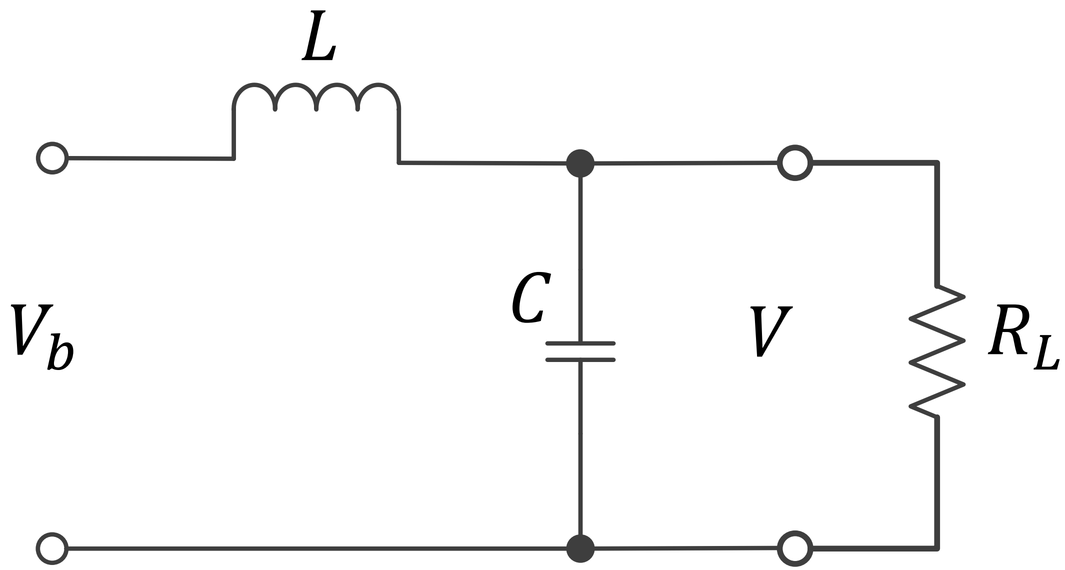
- Parameters are constant during simulation
- Two types of constants in Modelica
  - `constant`
  - `parameter`

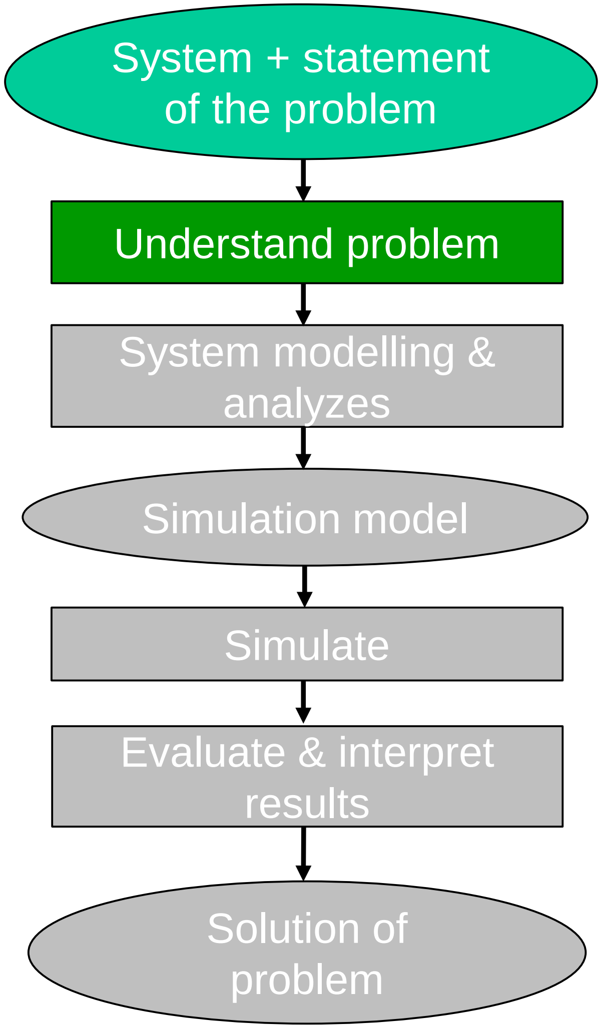
```
constant  Real    PI=3.141592653589793;
constant  String  redcolor = "red";
constant  Integer one = 1;
parameter Real    mass = 22.5;
```

Modelling and Simulation Process



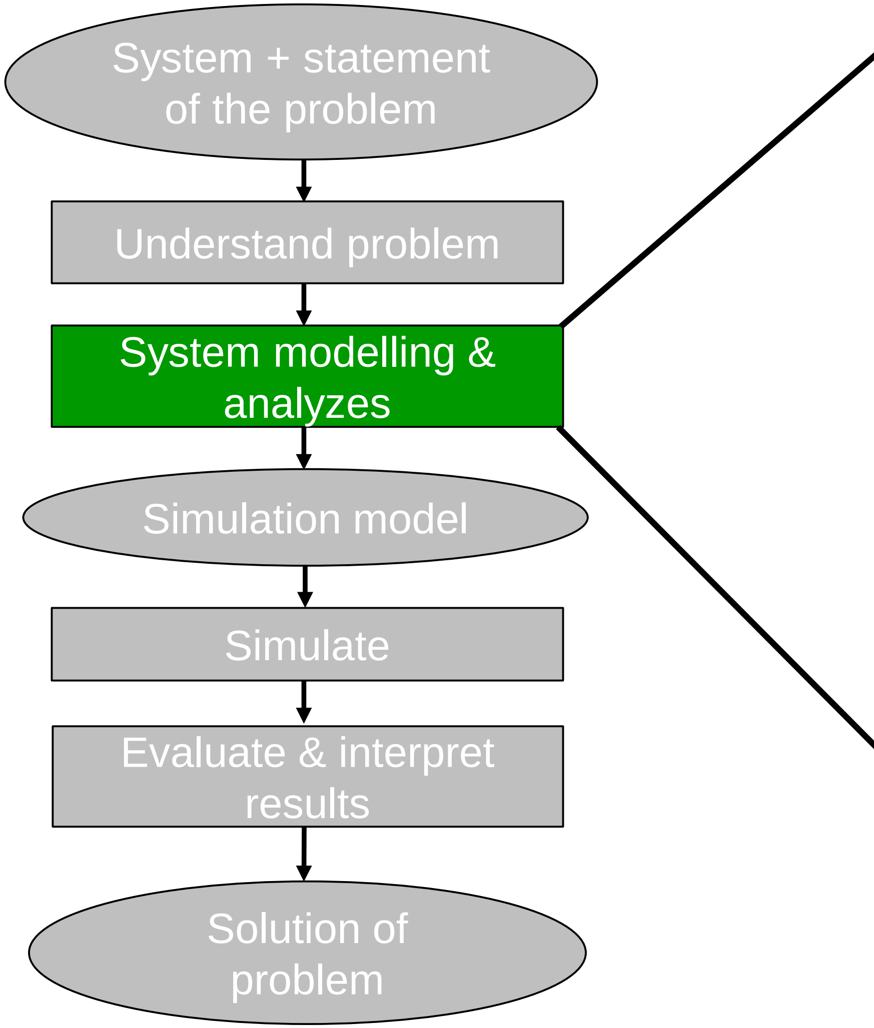
2^{nd} Example – Low-Pass RLC Filter

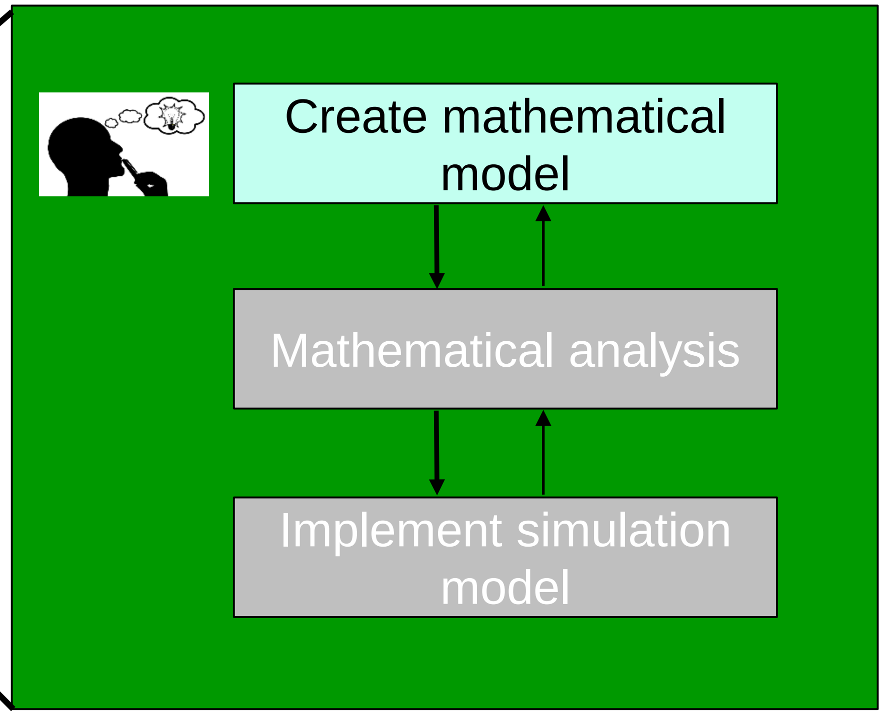


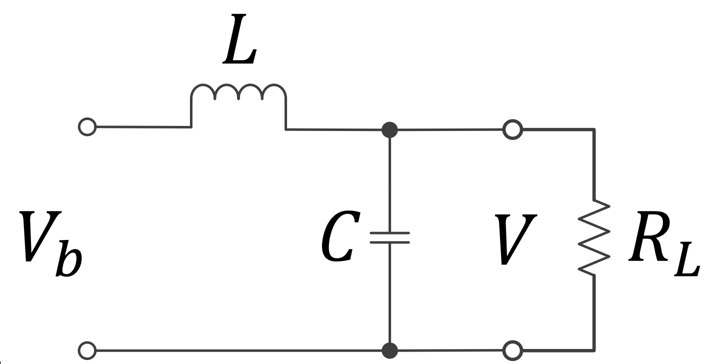


Solve for: V, i_L, i_R and i_C

Create mathematical model





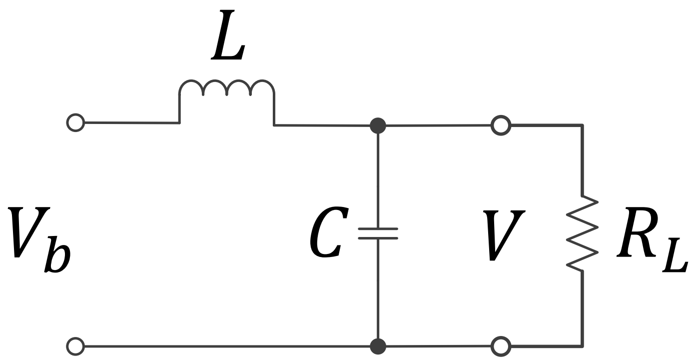


Solve for: V, i_L, i_R and i_C

Equations:

- Type equation here.

2nd Example – Low-Pass RLC Filter

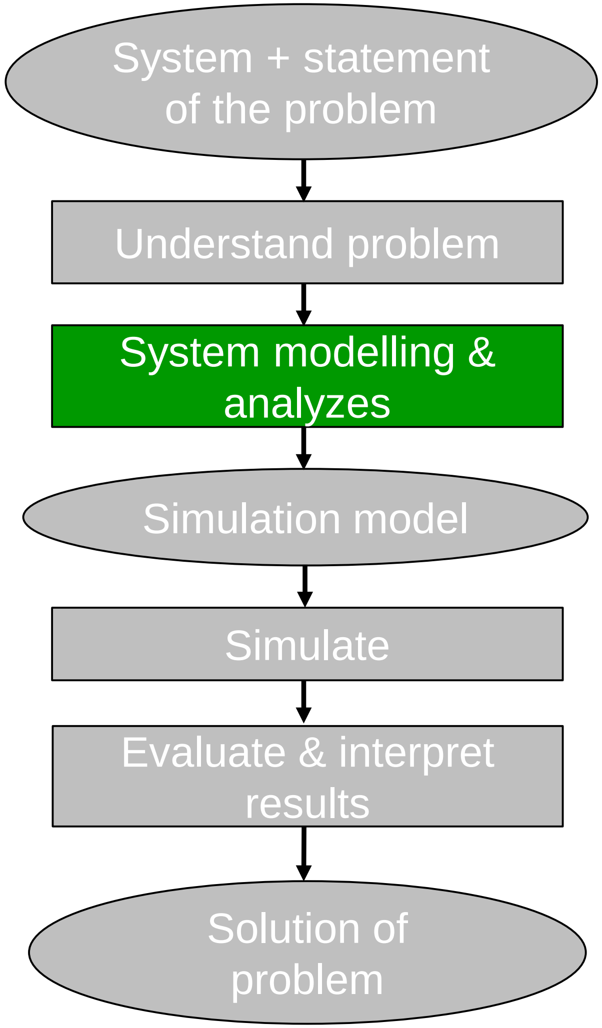


$$V = i_R R$$

$$C\frac{dV}{dt} = i_C$$

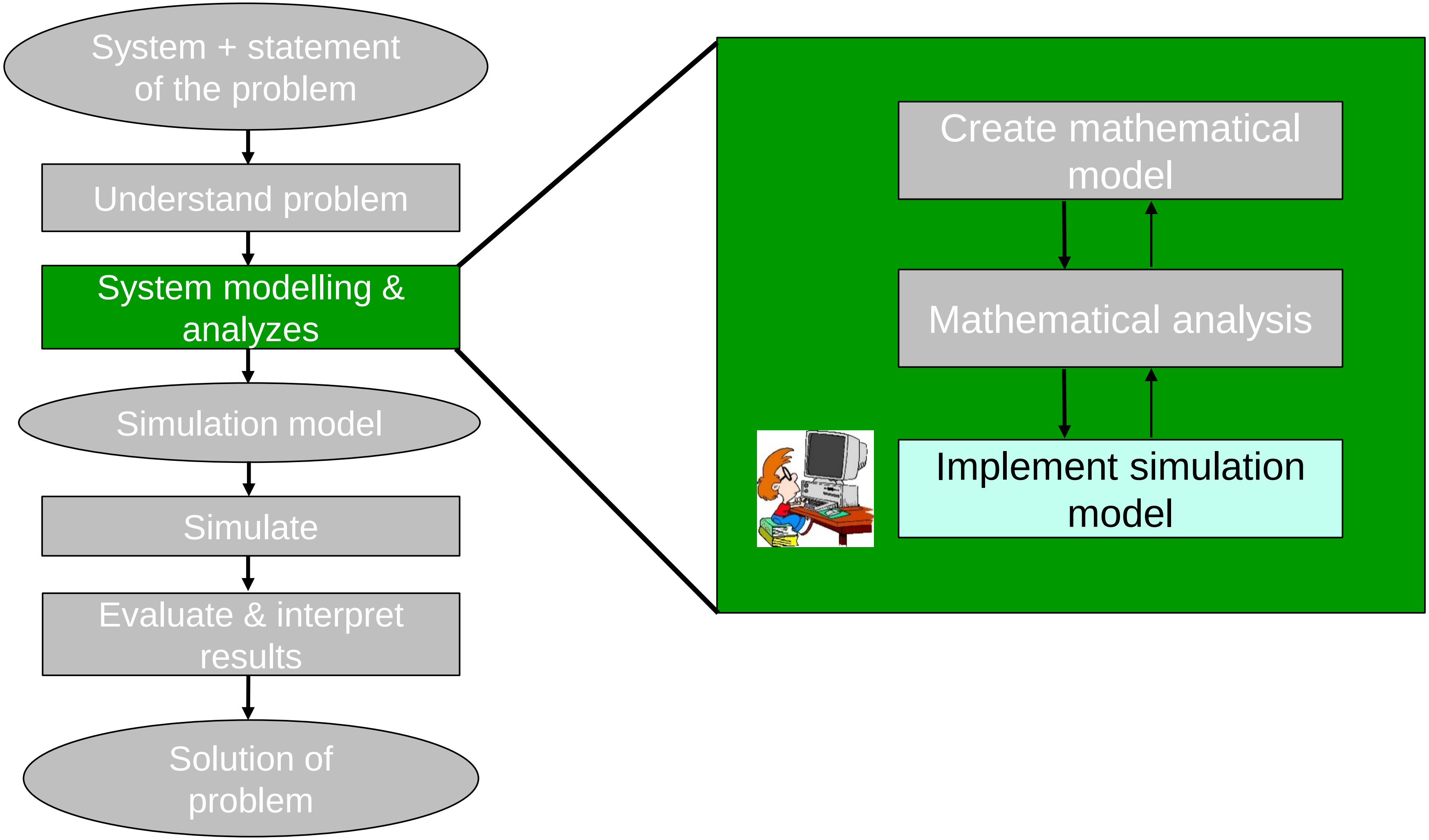
$$L\frac{di_L}{dt} = (\mathrm{V_b} - \mathrm{V})$$

$$i_L = i_R + i_C$$



Example taken from: http://book.xogeny.com/

2nd Example – Low-Pass RLC Filter



2nd Example – Code

Name of model

Import of SIunits package

```
model RLC5 "A resistor-inductor-capacitor circuit model"

  import SI = Modelica.SIunits "imports SIunits from Modelica package";
  // import Modelica.SIunits.* "other way to import package. Here top-level";

  a
  type Voltage = Real(unit="V");
  type Current = Real(unit="A");

  parameter Voltage Vb=24 "Battery voltage";
  parameter SI.Inductance L = 1;
  parameter SI.Resistance R = 100;
  parameter SI.Capacitance C = 1e-03;

  Voltage V;
  Current i_L;
  Current i_R;
  Current i_C;

equation
  V = i_R*R;
  C*der(V) = i_C;
  L*der(i_L) = (Vb-V);
  i_L = i_R+i_C;

end RLC5;
```

New type definition

Parameter, constant during simulation

Continuous-time variable

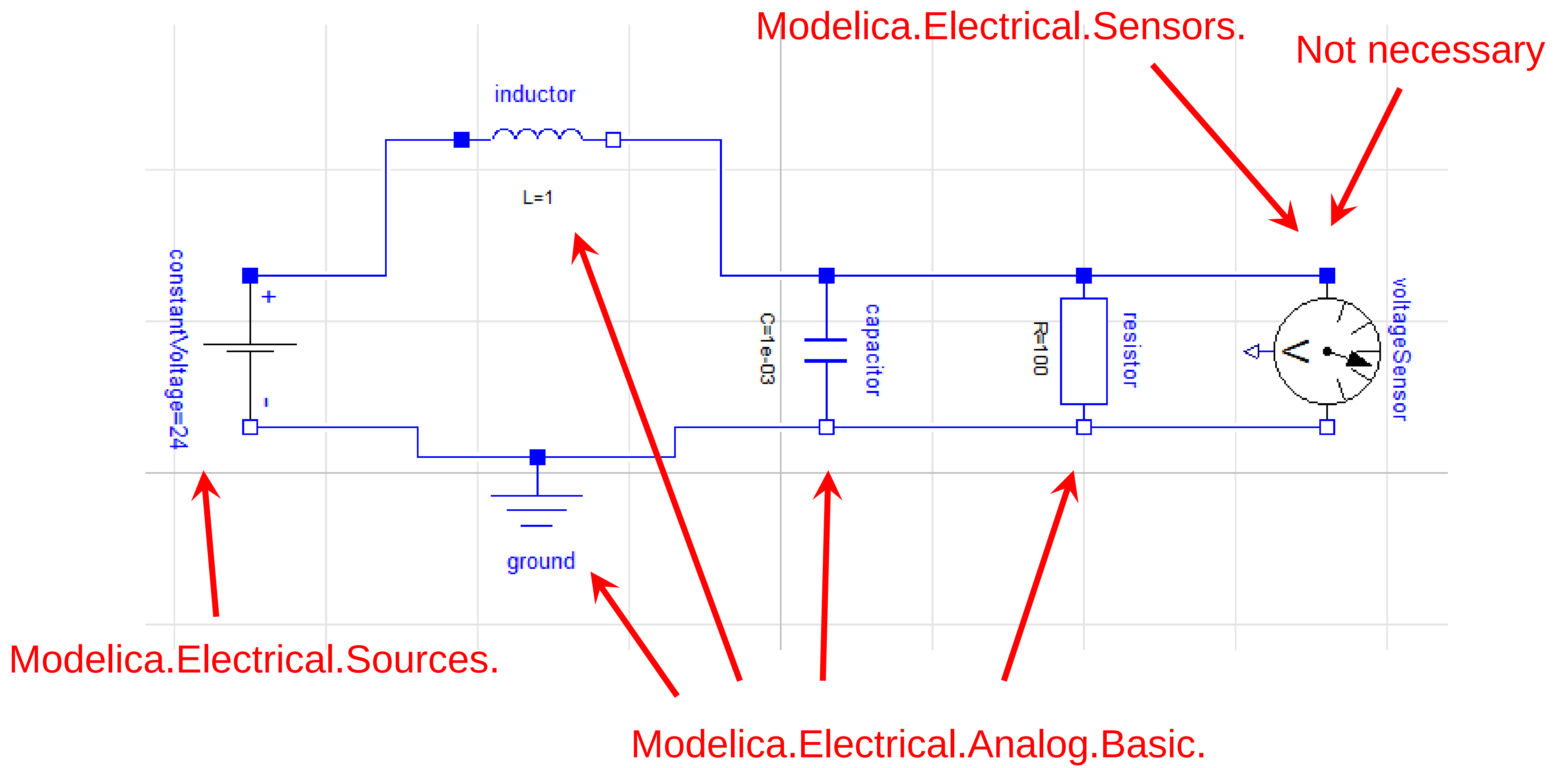
Differential equation

Algebraic equation

Modelica technology

- Modelica includes *graphical* editing for application model design based on predefined components

2nd Example – Graphical solution with the help of Dymola standard packages



Connectors and Connector Classes

Connectors are instances of *connector classes*

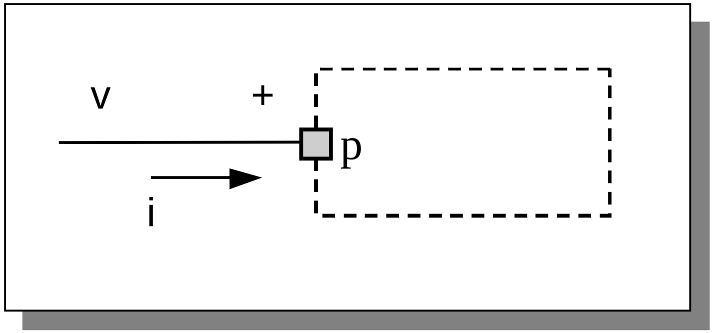
electrical connector

connector class

keyword `flow` indicates that currents of connected pins sum to zero.

an instance p of class `Pin`

```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;

Pin p;
```



The `flow` prefix

Two kinds of variables in connectors:

- *Non-flow variables* ***potential*** or energy level
- *Flow variables* represent some kind of **flow**

The `flow` prefix

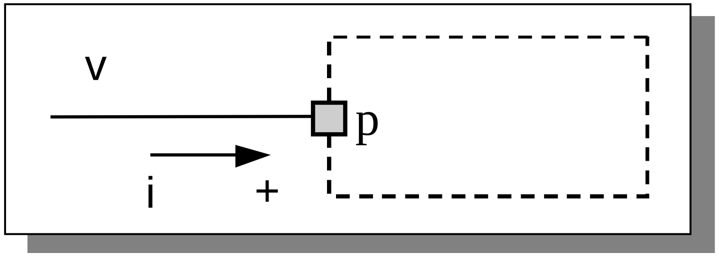
Coupling

- *Equality coupling*, for non-`flow` variables
  - In electrics: $v_1 = v_2 = \ ... = v_n$ (Kirchhoff's 2nd law)

- *Sum-to-zero coupling*, for `flow` variables
  - In electrics: $i_1 + i_2 + \ ... + i_n = 0$ (Kirchhoff's 1st law)

The value of a `flow` variable is *positive* when the current or the flow is *into* the component

positive flow direction:



Connectors and Connector Classes

Connectors are instances of *connector classes*

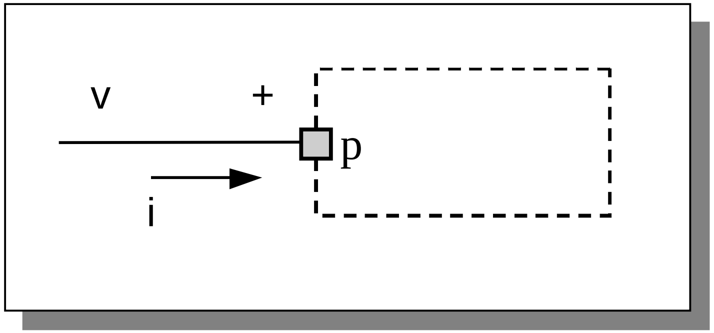
electrical connector

connector class

keyword `flow` indicates that currents of connected pins sum to zero.

an instance p of class `Pin`

```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;

Pin p;
```



What does the product of the electric pair of connector variables represent?

Connectors and Connector Classes

Connectors are instances of *connector classes*

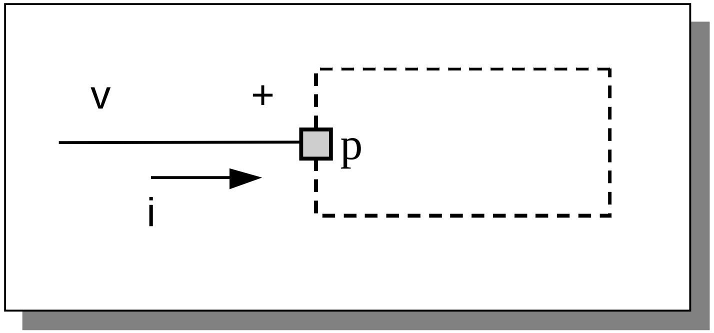
electrical connector

connector class

keyword `flow` indicates that currents of connected pins sum to zero.

```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;

Pin p;
```

an instance p of class `Pin`

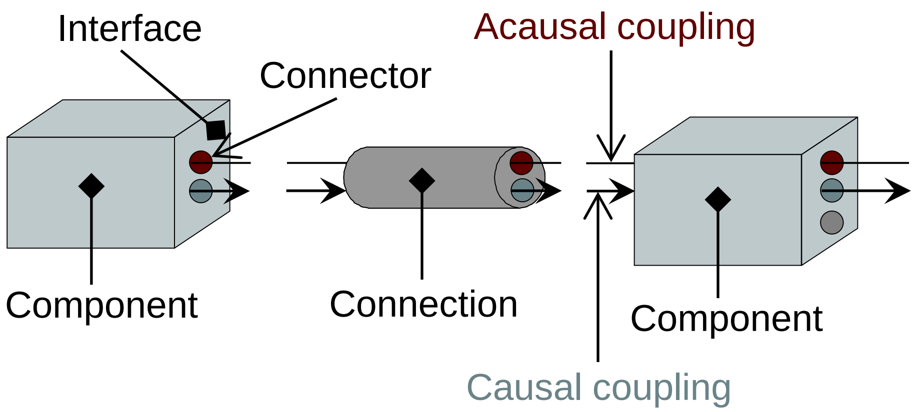


What does the product of the electric pair of connector variables represent?

- $v \left[\frac{Nm}{C}\right] \cdot i \left[\frac{C}{s}\right] = p \left[\frac{Nm}{s} = W\right]$

→ Flow of energy

Software Component Model

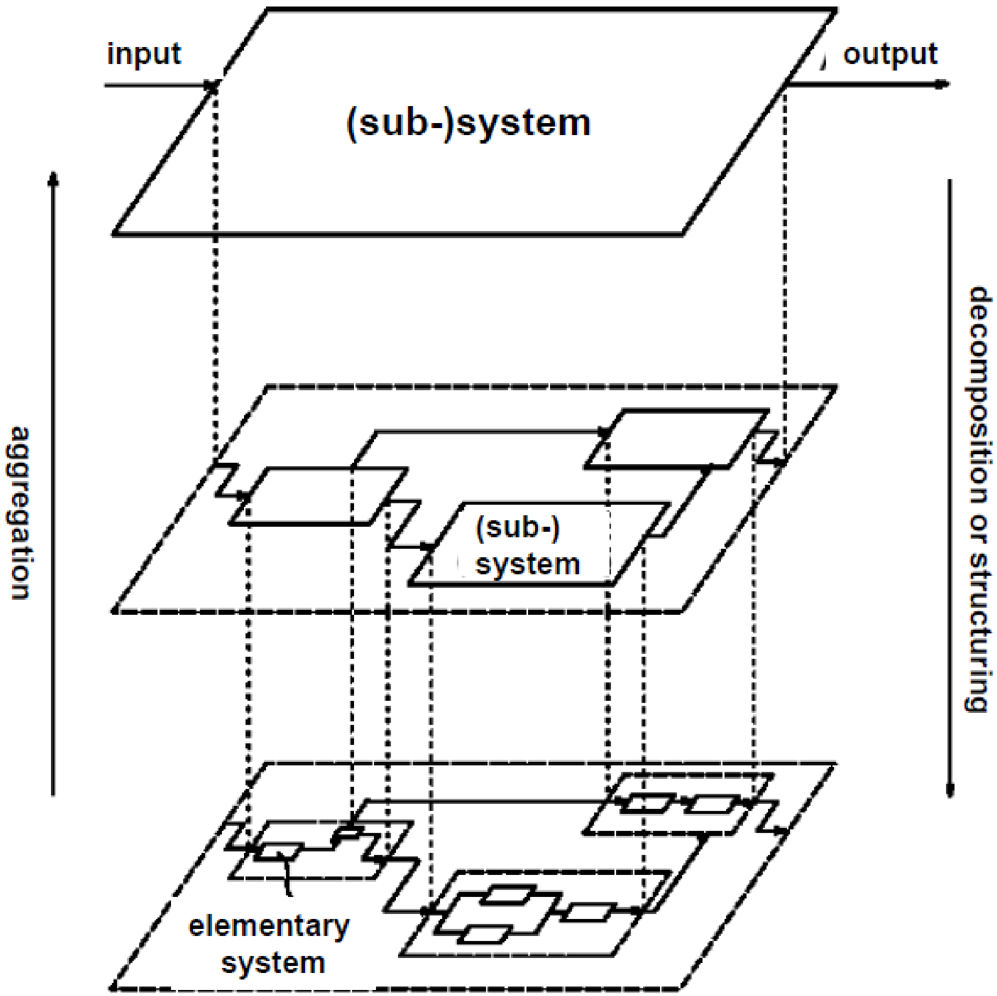


A component class should be defined *independently of the environment,* very essential for *reusability*

A component may internally consist of other components, i.e. *hierarchical* modeling

Complex systems usually consist of large numbers of *connected* components

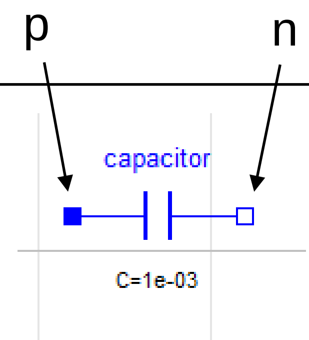
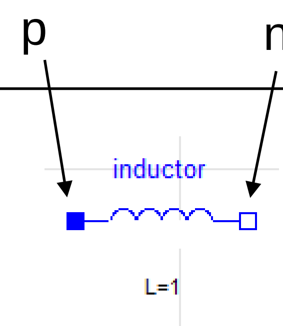
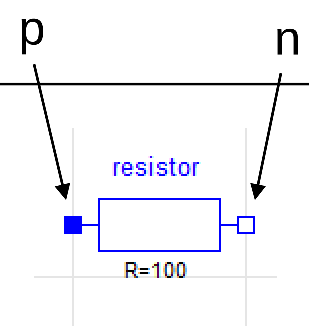
Decomposition and Aggregation of Systems



Picture taken from AVT-PT Aachen – Modeling Technical Systems

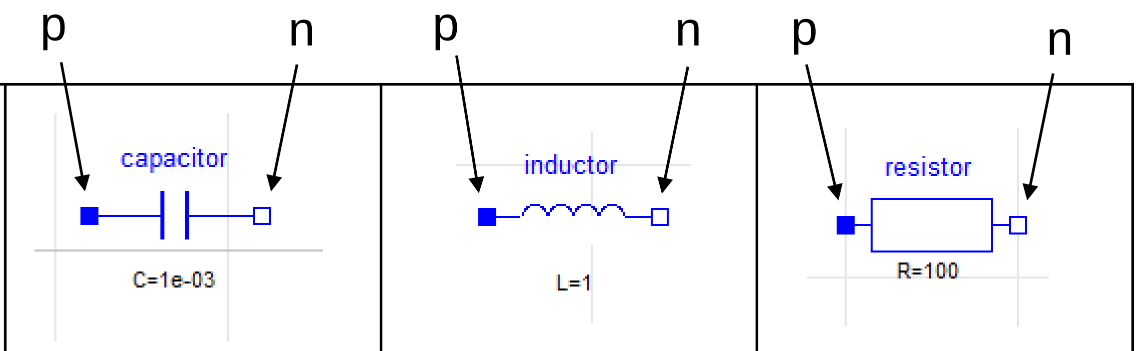
Use of Inheritance & Connectors

- Connector-Name (Pin): p, n

<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Equations:</td><td>$0 = p.i + n.i$
$v = p.v - n.v$
$i = p.i$
$i = C * der(v)$</td><td>$0 = p.i + n.i$
$v = p.v - n.v$
$i = p.i$
$v = L * der(i)$</td><td>$0 = p.i + n.i$
$v = p.v - n.v$
$i = p.i$
$v = R * i$</td></tr>
</table>

Use of Inheritance & Connectors

- Connector-Name (Pin): p,n



Equations:	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $i = C * der(v)$	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $v = L * der(i)$	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $v = R * i$

→ Only <u>one</u> equation different

Reuse same components

1) Create connector:

```
connector Pin
  Modelica.SIunits.Voltage v;//identical at connection
  flow Modelica.SIunits.Current i;//sum-to-0 at connection
end Pin;
```

Reuse same components

1) Create connector:

```
connector Pin
  Modelica.SIunits.Voltage v;//identical at connection
  flow Modelica.SIunits.Current i;//sum-to-0 at connection
end Pin;
```

2) Create “blueprint” model class TwoPin:

```
partial model TwoPin “Superclass of elements
with two electrical pins”
  Pin p, n;
  Modelica.SIunits.Voltage v;
  Modelica.SIunits.Current i;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
```

Reuse same components

1) Create connector:

```
connector Pin
  Modelica.SIunits.Voltage v;//identical at connection
  flow Modelica.SIunits.Current i;//sum-to-0 at connection
end Pin;
```

2) Create “blueprint” model class TwoPin:

```
partial model TwoPin “Superclass of elements
with two electrical pins”
  Pin p, n;
  Modelica.SIunits.Voltage v;
  Modelica.SIunits.Current i;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
```

partial because:

- Problem is structurally singular
- 6 variables & only 5 equations

Reuse same components with extends

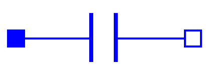
3) Create model with previous components

```
model Resistor “Ideal electrical resistor”
  extends TwoPin
  parameter Modelica.SIunits.Resistance R;

equation
  R*i = v;
end Resistor;
```

```
model Capacitor “Ideal electrical capacitor”
  extends TwoPin
  parameter Modelica.SIunits.Capacitance C;

equation
  C*der(v) = i;
end Capacitor;
```

Use of Modelica connectors

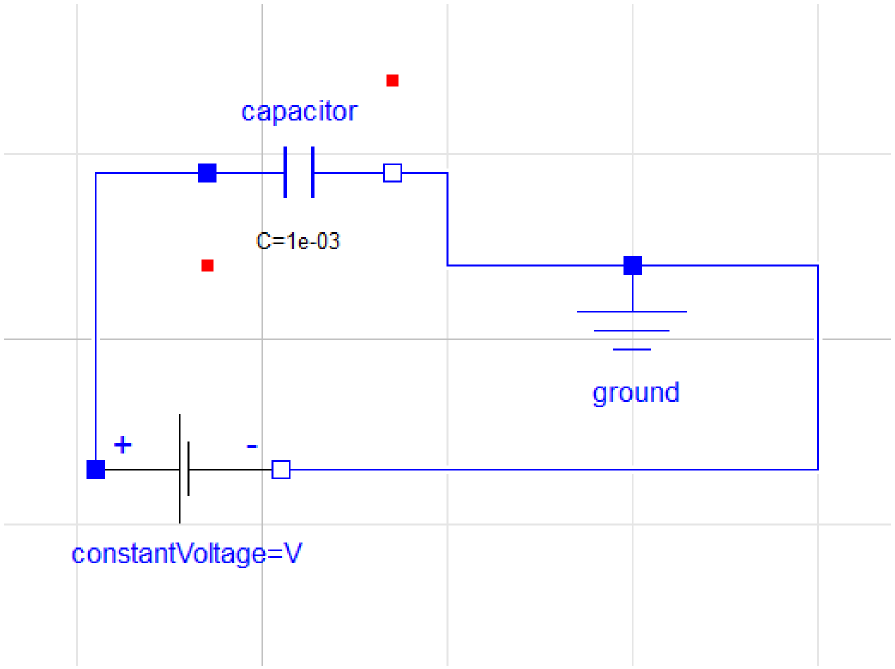
```
connector Pin
  Voltage      v;   // identical at connection
  flow Current i;   // sums to zero at connection
end Pin;
           partial model TwoPin
             Pin p, n;   Voltage v; Current i;
           equation
             v = p.v - n.v;
             0 = p.i + n.i;
             i = p.i;
           end TwoPin;
                     model Capacitor
                       extends TwoPin;
                       parameter Capacitance C;
                     equation
                       C*der(v) = i;
                     end Capacitor;
```



Implementation and Execution of Modelica

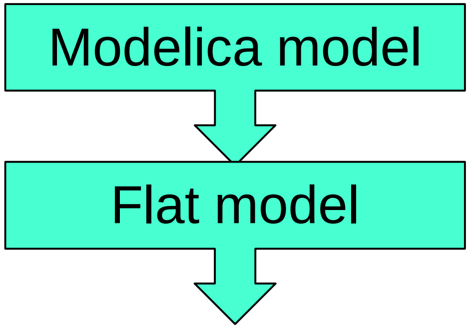
Modelica model

Modelica model




```
model Test_Capacitor

//   parameter Real i=1;
//   parameter Real v=20;
//
//   Pin p;

  Modelica.Electrical.Analog.Basic.Capacitor capacitor(C=1e-03);
  Modelica.Electrical.Analog.Sources.ConstantVoltage constantVoltage;
  Modelica.Electrical.Analog.Basic.Ground ground;
equation
  connect(constantVoltage.p, capacitor.p);
  connect(capacitor.n, ground.p);
  connect(ground.p, constantVoltage.n);

end Test_Capacitor;

partial package Modelica.Icons.Package "Icon for standard packages"

end Package;

model Modelica.Electrical.Analog.Basic.Capacitor
  "Ideal linear electrical capacitor"
  extends Interfaces.OnePort;
  parameter SI.Capacitance C(start=1) "Capacitance";

equation
  i = C*der(v);
end Capacitor;

partial package Modelica.Icons.InterfacesPackage
  "Icon for packages containing interfaces"
//extends Modelica.Icons.Package;
end InterfacesPackage;

partial model Modelica.Electrical.Analog.Interfaces.OnePort
  "Component with two electrical pins p and n and current i from p to n"

  SI.Voltage v "Voltage drop between the two pins (= p.v - n.v)";
  SI.Current i "Current flowing from pin p to pin n";
  PositivePin p
    "Positive pin (potential p.v > n.v for positive voltage drop v)";
  NegativePin n "Negative pin";
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;

connector Modelica.Electrical.Analog.Interfaces.PositivePin
  "Positive pin of an electric component"
  Modelica.SIunits.Voltage v "Potential at the pin";
  flow Modelica.SIunits.Current i "Current flowing into the pin";
```

Implementation and Execution of Modelica

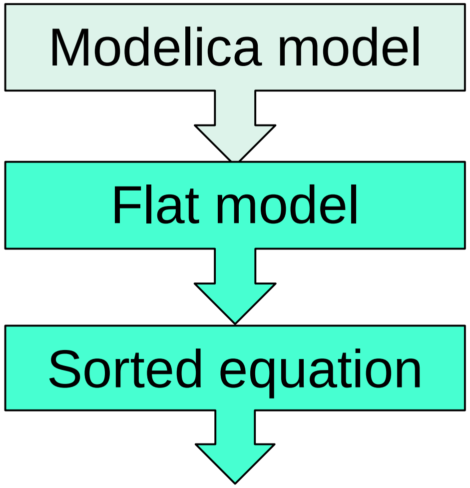


Parsed, preprocessing, flattening

Flat model

```
model Test_Capacitor
parameter Modelica.SIunits.Capacitance capacitor.C(start = 1) = 0.001
  "Capacitance";
parameter Modelica.SIunits.Voltage constantVoltage.V = 1 "Value of constant voltage";

Modelica.SIunits.Voltage capacitor.v "Voltage drop between the two pins (= p.v - n.v)";
Modelica.SIunits.Current capacitor.i "Current flowing from pin p to pin n";
Modelica.SIunits.Voltage capacitor.p.v "Potential at the pin";
Modelica.SIunits.Current capacitor.p.i "Current flowing into the pin";
Modelica.SIunits.Voltage capacitor.n.v "Potential at the pin";
Modelica.SIunits.Current capacitor.n.i "Current flowing into the pin";
Modelica.SIunits.Voltage constantVoltage.v "Voltage drop between the two pins (= p.v - n.v)";
Modelica.SIunits.Current constantVoltage.i "Current flowing from pin p to pin n";
Modelica.SIunits.Voltage constantVoltage.p.v "Potential at the pin";
Modelica.SIunits.Current constantVoltage.p.i "Current flowing into the pin";
Modelica.SIunits.Voltage constantVoltage.n.v "Potential at the pin";
Modelica.SIunits.Current constantVoltage.n.i "Current flowing into the pin";
Modelica.SIunits.Voltage ground.p.v "Potential at the pin";
Modelica.SIunits.Current ground.p.i "Current flowing into the pin";

// Equations and algorithms

  // Component capacitor
  // class Modelica.Electrical.Analog.Basic.Capacitor
    // extends Modelica.Electrical.Analog.Interfaces.OnePort
    equation
      capacitor.v = capacitor.p.v-capacitor.n.v;
      0 = capacitor.p.i+capacitor.n.i;
      capacitor.i = capacitor.p.i;
    // end of extends
  equation
    capacitor.i = capacitor.C*der(capacitor.v);

  // Component constantVoltage
  // class Modelica.Electrical.Analog.Sources.ConstantVoltage
    // extends Modelica.Electrical.Analog.Interfaces.OnePort
    equation
      constantVoltage.v = constantVoltage.p.v-constantVoltage.n.v;
      0 = constantVoltage.p.i+constantVoltage.n.i;
      constantVoltage.i = constantVoltage.p.i;
    // end of extends
  equation
    constantVoltage.v = constantVoltage.V;

  // Component ground
  // class Modelica.Electrical.Analog.Basic.Ground
  equation
    ground.p.v = 0;

  // Component
  // class Test_Capacitor
  equation
    capacitor.n.i+constantVoltage.n.i+ground.p.i = 0.0;
    constantVoltage.n.v = capacitor.n.v;
    ground.p.v = capacitor.n.v;
    capacitor.p.i+constantVoltage.p.i = 0.0;
    constantVoltage.p.v = capacitor.p.v;

end Test_Capacitor;
```

Implementation and Execution of Modelica



Parsed, preprocessing, flattening

“make equations causal”, perform BLT transformation

Lower Triangular Matrix (Example)

	v_G	v_{C1}	v_{R1}	u_R	i_{R1}	i_{S1}	i_{C1}	du_c	i_G
1)	x								
6)	x	x							
2)	x		x						
4)		x	x	x					
3)				x	x				
7)					x	x			
8)					x		x		
5)							x	x	
9)						x	x		x

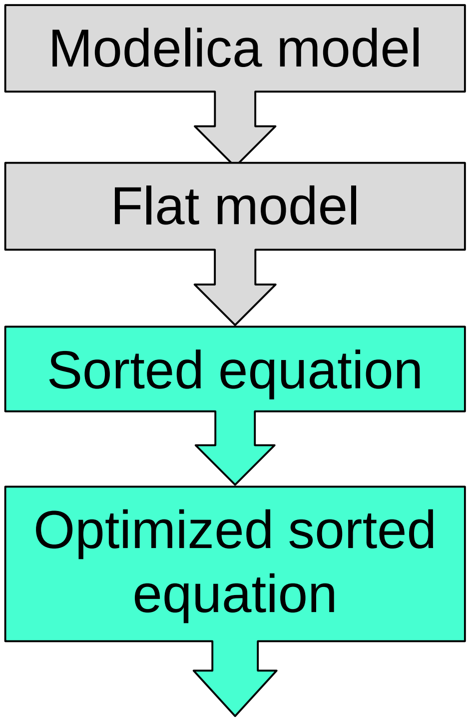
Causal List:

1) $\mathrm{v_G} \coloneqq 0$
6) $\mathrm{v_{C1}} \coloneqq -\mathrm{u_c} + \mathrm{v_G}$
2) $\mathrm{v_{R1}} \coloneqq \mathrm{v_G} + 10\mathrm{V}$
4) $\mathrm{u_R} \coloneqq \mathrm{v_{C1}} - \mathrm{v_{R1}}$
3) $\mathrm{i_{R1}} \coloneqq \mathrm{u_R}/\mathrm{R}$
7) $\mathrm{i_{S1}} \coloneqq \mathrm{i_{R1}}$
8) $\mathrm{i_{C1}} \coloneqq \mathrm{i_{R1}}$
5) $\frac{\mathrm{du_c}}{\mathrm{dt}} \coloneqq \mathrm{i_{C1}}/\mathrm{C}$
9) $\mathrm{i_G} \coloneqq \mathrm{i_{C1}} - \mathrm{i_{S1}}$

Block Lower Triangular Matrix (Example)

	x								
	x	x							
	x		x	x					
		x	x	x					
				x	x				
					x	x		x	x
					x		x		x
							x	x	
						x	x		x

Implementation and Execution of Modelica

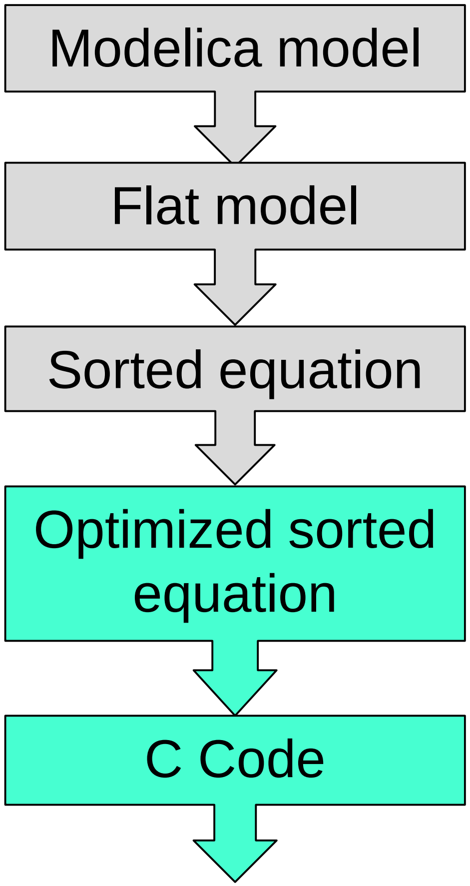


Parsed, preprocessing, flattening

“make equations causal”, perform BLT transformation

Optimize and eliminate equation

Implementation and Execution of Modelica



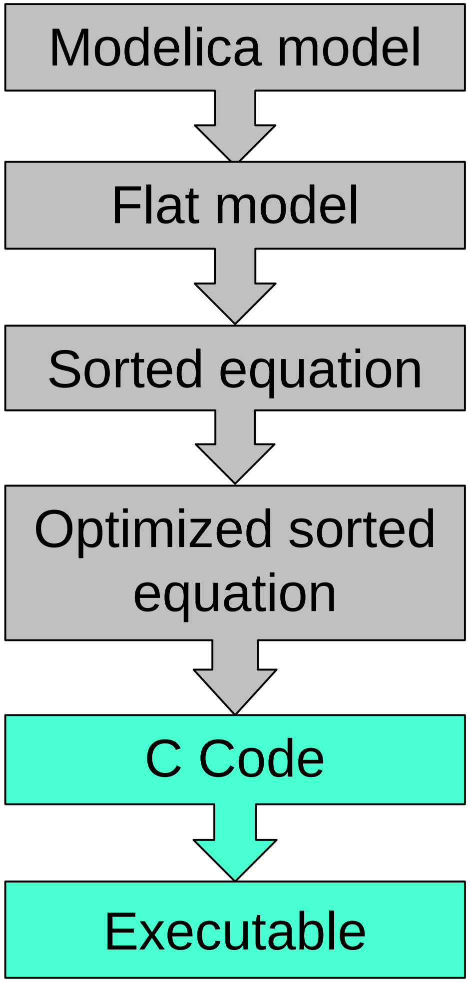
Parsed, preprocessing, flattening

“make equations causal”, perform BLT transformation

Optimize and eliminate equation

Generate C code

Implementation and Execution of Modelica



Parsed, preprocessing, flattening

“make equations causal”, perform BLT transformation

Optimize and eliminate equation

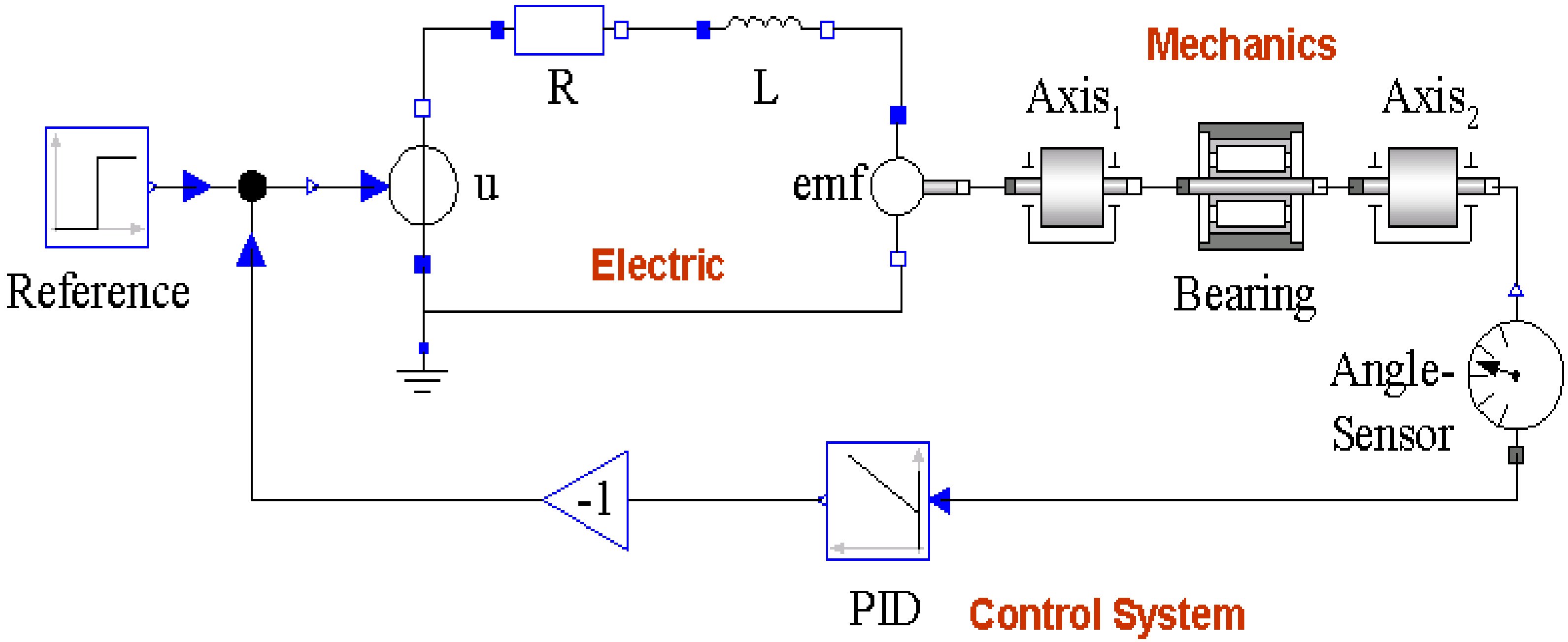
Generate C code

What is Special about Modelica?

- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation

What is Special about Modelica?

Multi-Domain Modeling



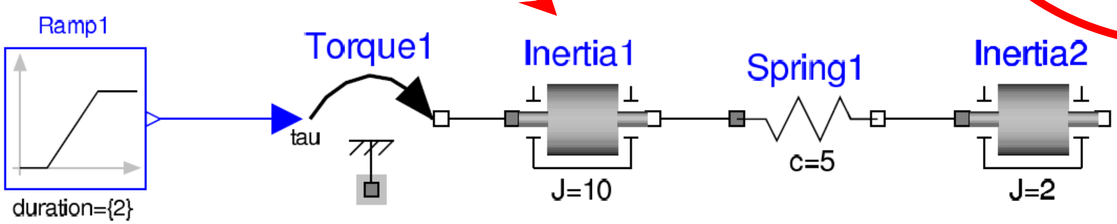
What is Special about Modelica?

Multi-Domain Modeling

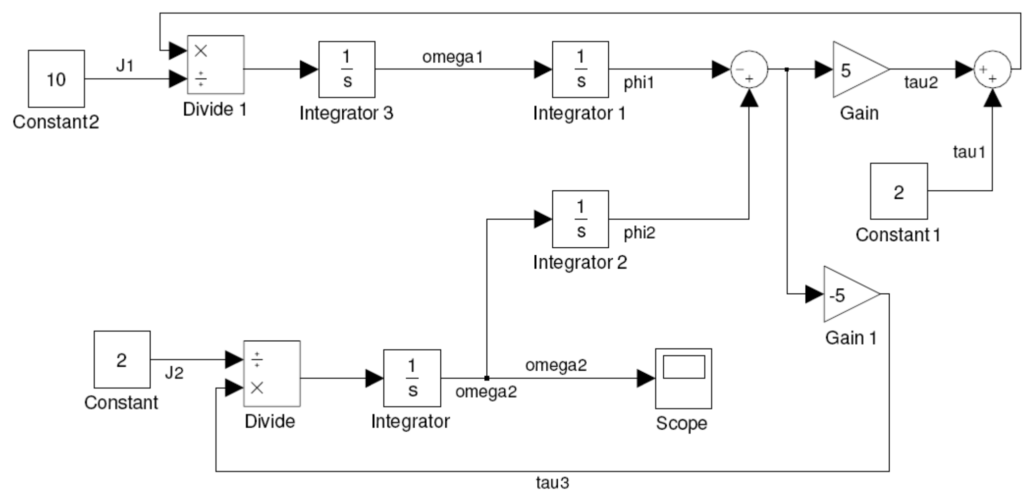
Visual Acausal Hierarchical Component Modeling

Keeps the physical structure

Acausal model (Modelica)



Causal block-based model (Simulink)



What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Hierarchical Component Modeling

A textual *class-based* language
OO primary used for as a structuring concept

Behaviour described declaratively using
- Differential algebraic equations (DAE) (continuous-time)
- Event triggers (discrete-time)

Variable declarations

```
model VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x”;
  Real y(start = 1)  "y coordinate”;
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = -x + lambda*(1 - x*x)*y;
end VanDerPol;
```

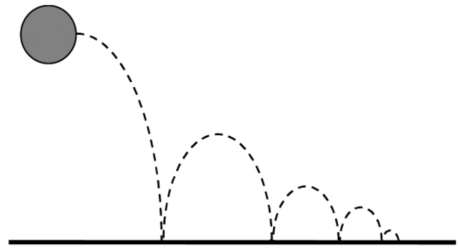
Differential equations

Typed Declarative Equation-based Textual Language

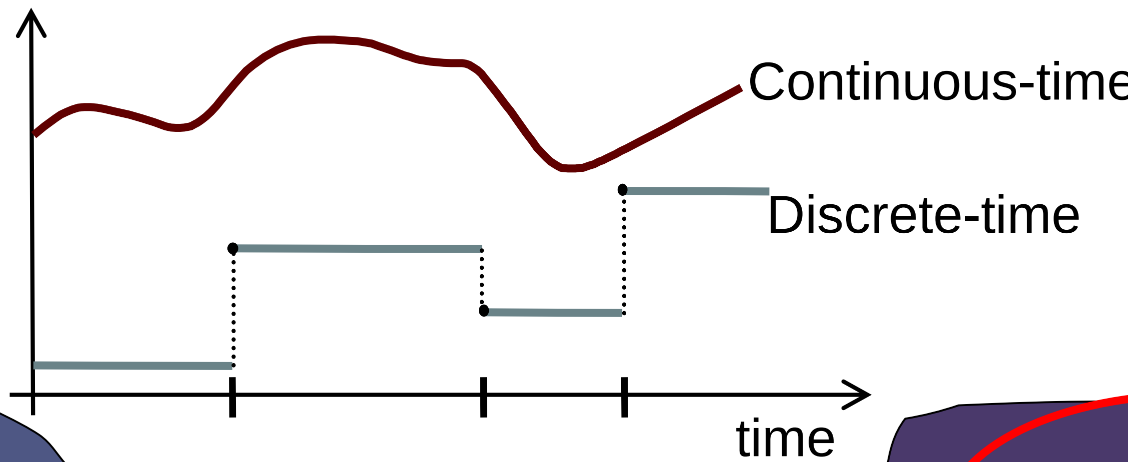
What is Special about Modelica?

Multi-Domain
Modeling

Visual Acausal
Component
Modeling



Hybrid modeling =
continuous-time + discrete-time modeling



Typed
Declarative
Equation-based
Textual Language

Hybrid
Modeling

Kvalitetssikring

- Fra http://snl.no/kvalitetssikring: Kvalitetssikring, planlagte og systematiske aktiviteter som gjøres for å oppnå at et produkt eller en tjeneste vil oppfylle kravene til kvalitet
- På samme måte som industrien tilbyr produkter til sine kunder, så tilbyr NTNU emner til sine studenter
- I så måte ønsker NTNU å tilby best mulig kvalitet på sine emner til studentene:
  - Sikre at emnets læringsmål er oppdaterte og relevante
  - Sikre at læringsaktivitetene i emnet bidrar til at studentene oppnår læringsutbyttet
  - Sikre at det er sammenheng mellom læringsmålene, læringsaktivitetene og vurderingsformene
- NTNU har egne wiki-sider med mye nyttig informasjon for både studenter og ansatte: https://innsida.ntnu.no/wiki (her kan man søke seg frem til det meste)
- Direkte lenke til NTNUs sider om kvalitetssikring av utdanning: https://innsida.ntnu.no/wiki/-/wiki/Norsk/Kvalitetssikring+av+utdanning
- Hvis ingenting blir gjort kan dere melde inn via NTNUs avvikssystem: https://innsida.ntnu.no/avvik

Referansegruppe

- Hver gang et emne har blitt undervist ved NTNU skal faglærer lage en emnerapport som beskriver tilstanden til emnet, inkludert tilbakemelding fra studenter og handlingsplan med tiltak for forbedringer til neste gang
- Tilbakemelding fra studenter innhentes vanligvis gjennom en referansegruppe, som skal gi faglærer en referanse på hva studentene mener om faget
- Referansegruppe:
  – Skal bestå av et representativt utvalg av emnets studenter (kjønn, studieprogram)
  – Minimum tre studenter
  – Skal løpende ta imot innspill fra alle studentene som følger emnet
  – Gjennomfører tre møter i løpet av semesteret: Oppstart, midtveis og ved avslutning av emnet
  – Skriver referansegrupperapport som oppsummerer studentenes synspunkter og forbedringsforslag
  – Alle medlemmene får tilbud om de ønsker bekreftelse på at de har deltatt i referansegruppen

Tilbakekobling for undervisning