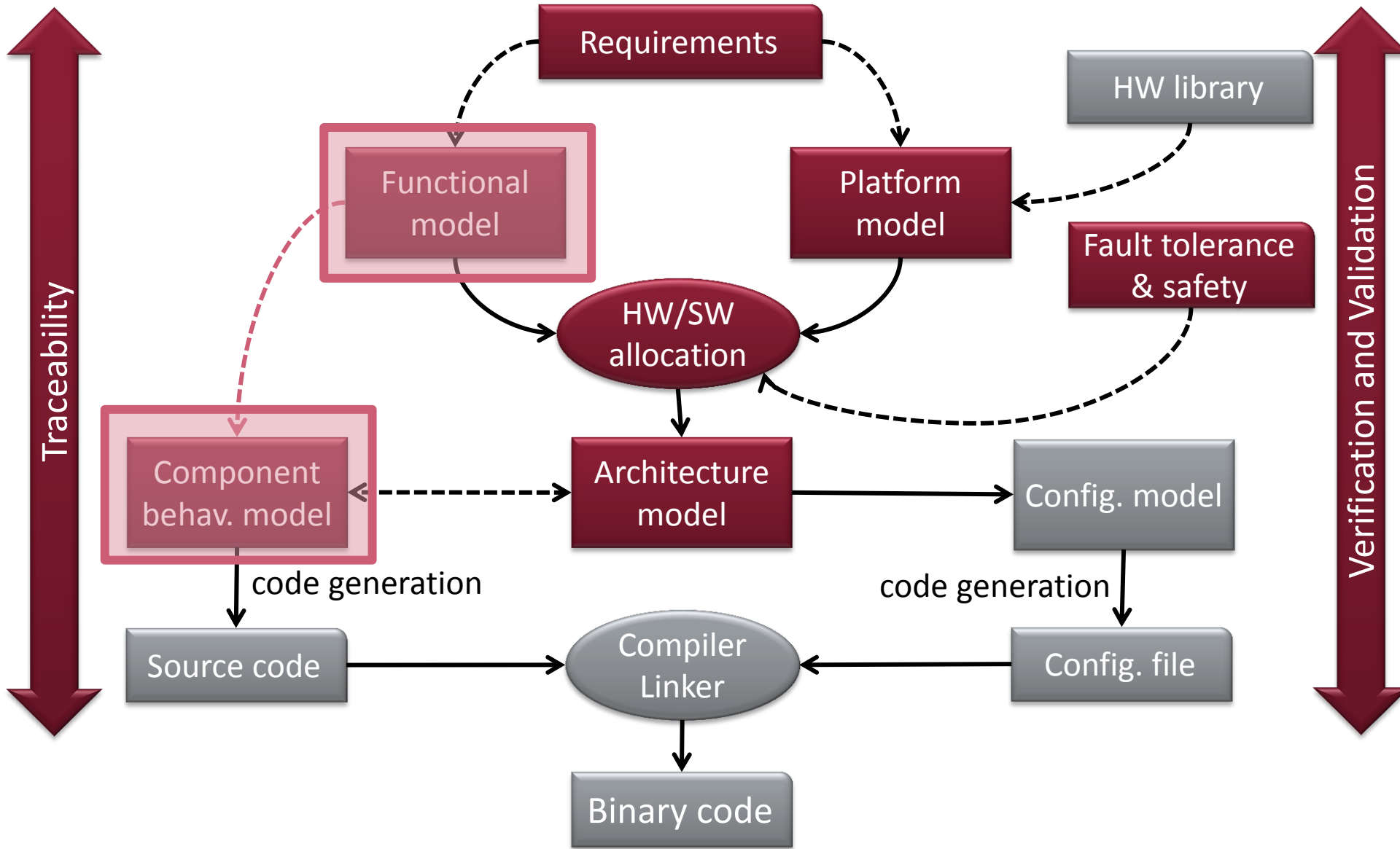


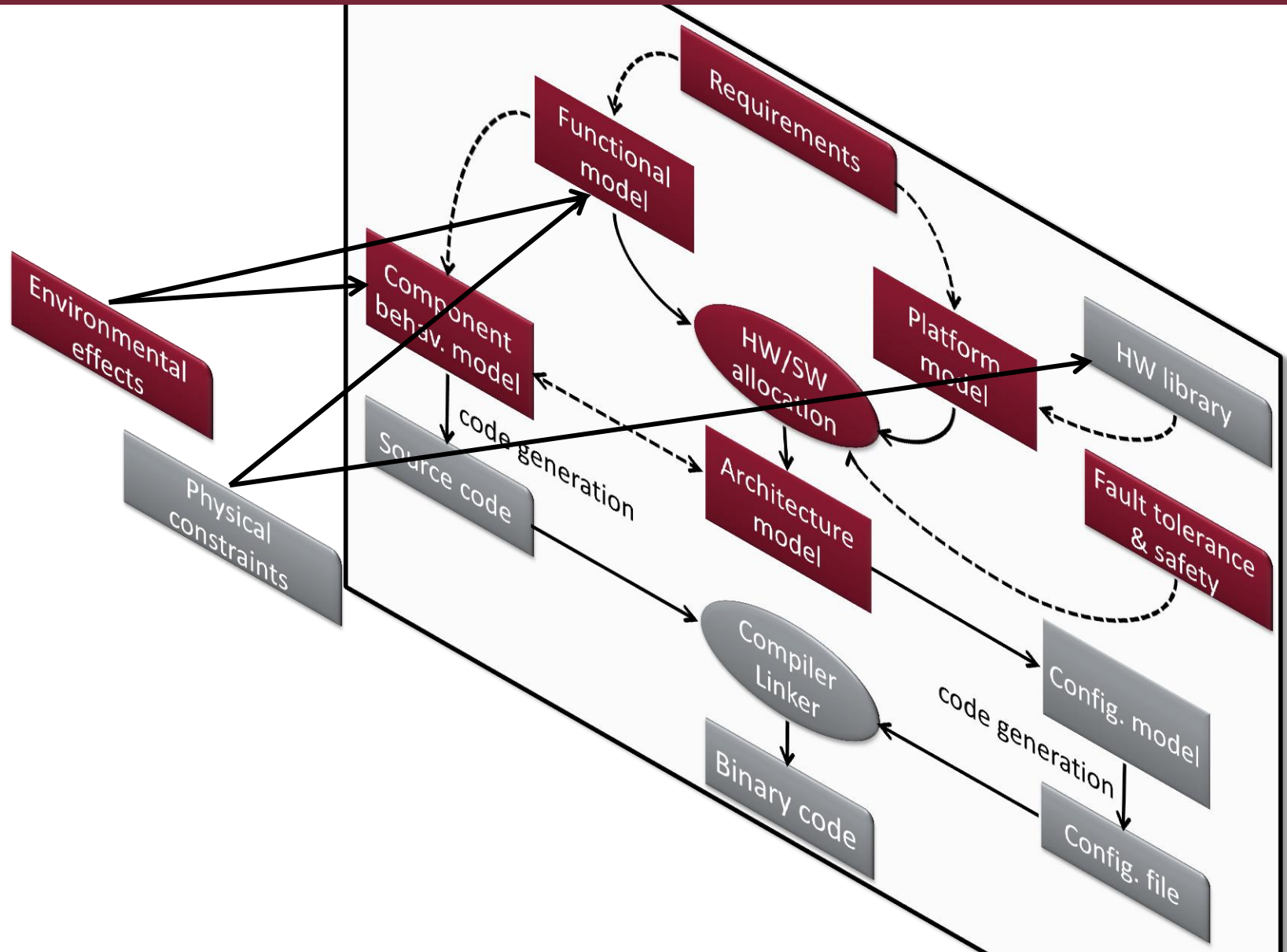
# Modeling physical properties

Controller, plant and environment model

# Platform-based systems design



# Platform-based systems design



# Learning Objectives

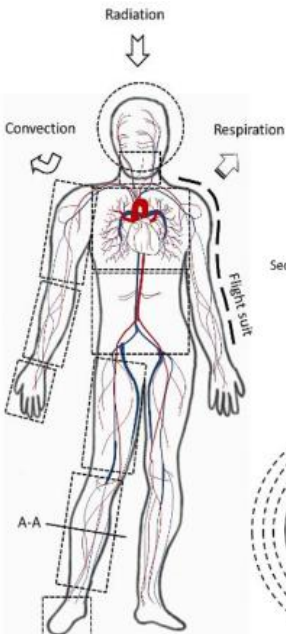
## Modeling physical parameters and constraints

- Include physical properties in a model
- Include rules constraining physical properties
- Capture properties and constraints using the SysML language

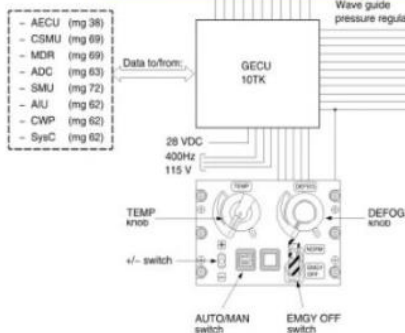
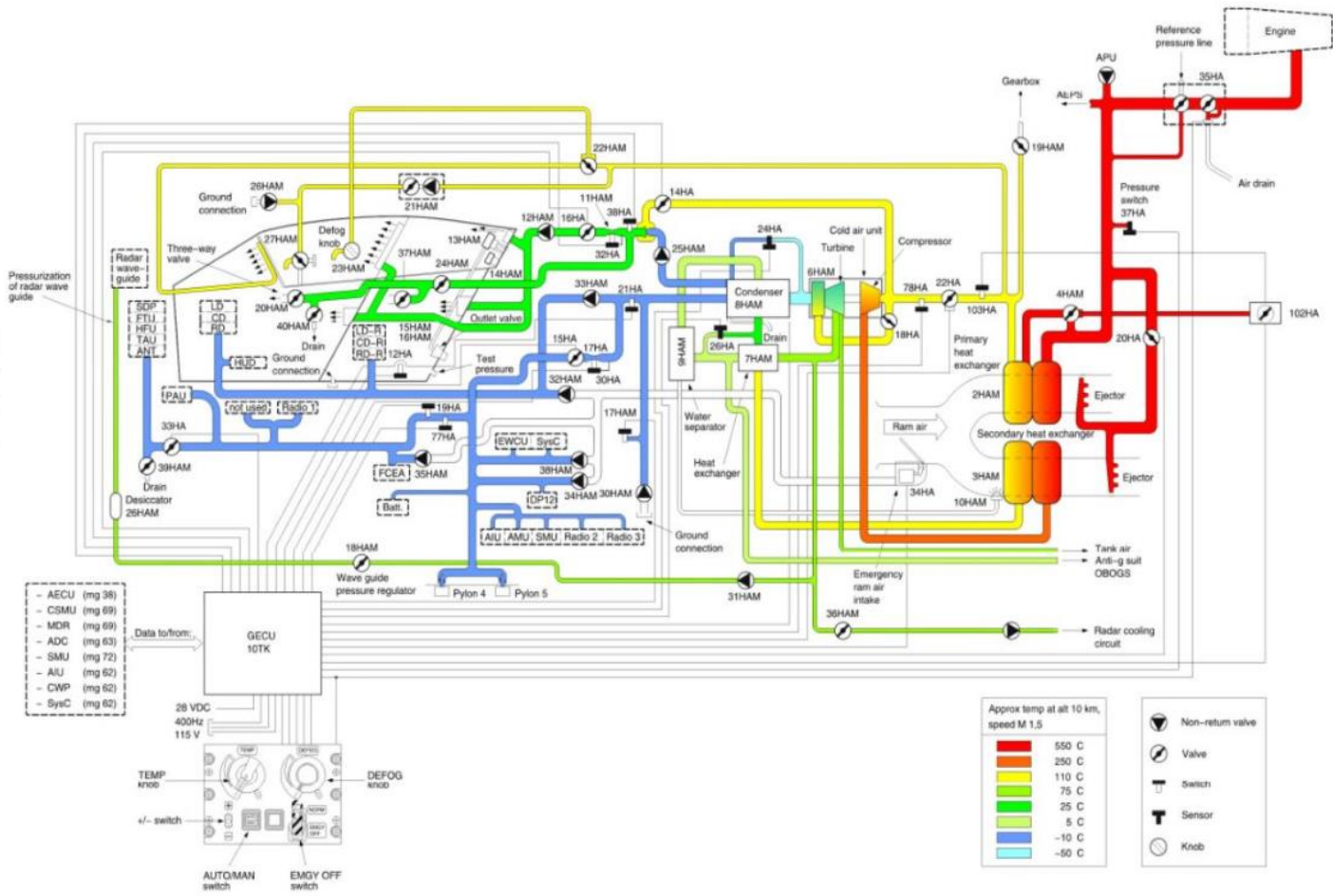
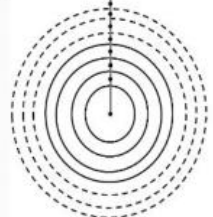
## Joint analysis of the system and the environment

- Modeling the controller, the plant, and the environment
- Capture both continuous-time and discrete time properties
- Identify the connection between the system, the plant, and the controller
- Analyze system properties and execute simulations using models
- Learn the basic modeling concepts of the Modelica language

# Thermal model of an aircraft



- Section A-A:
- Jacket
  - Air / Clothing Layer
  - Under Clothing
  - Skin
  - Fat
  - Muscle
  - Bone



Copyright:

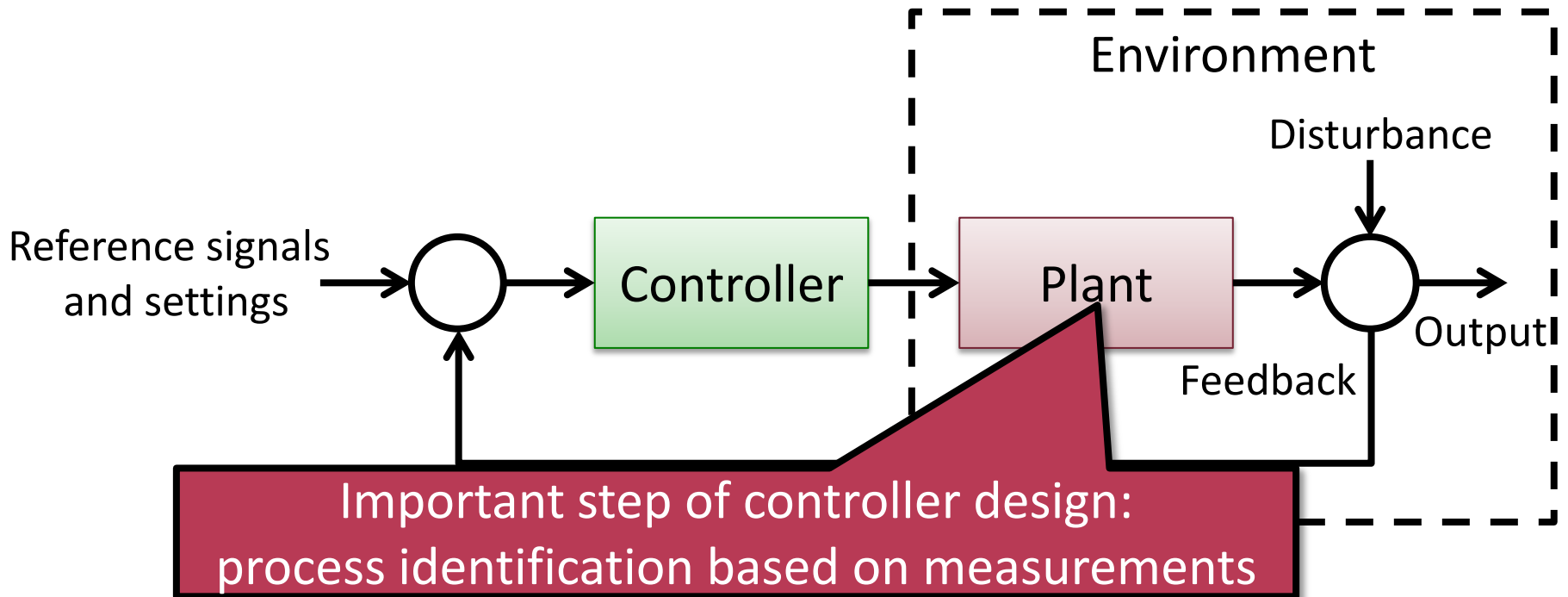


**SAAB**



# Controller, Plant, and Environment

- Typical system control loop



- Co-designing controller and the plant would be the ideal setting

# Controller design

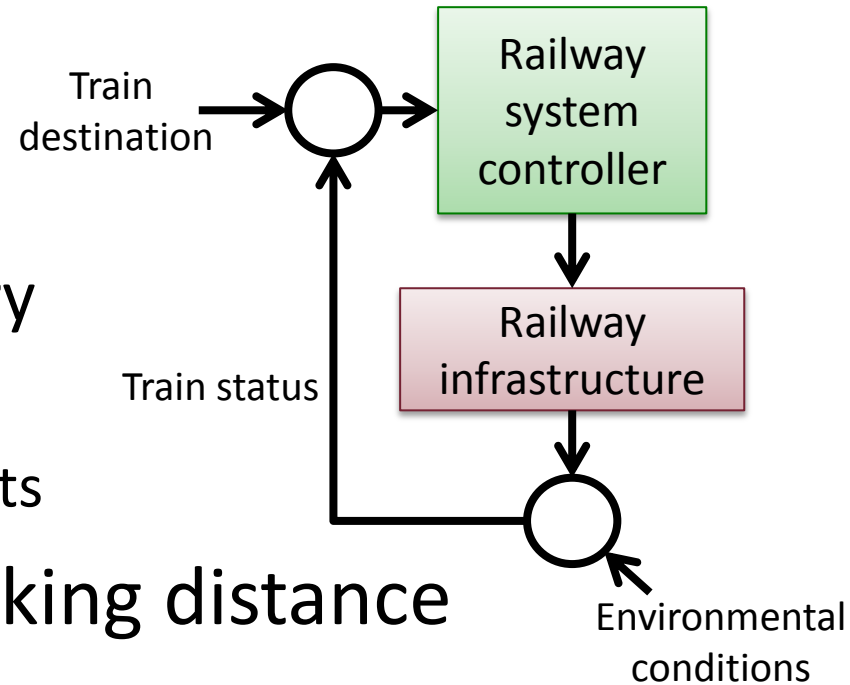
- Controller functional design using blocks
  - BDD: defines element hierarchy and containment
  - IBD: template for component internal structure
- Challenge: validate the design of the controller
  - On-site testing and calibration can be
    - Expensive (time + cost)
    - Dangerous
  - Instead:
    - create plant model and environment model with physical properties and
    - run simulations

# Example railway system controller

- Controller aims to
  - monitor the trains
  - apply brakes when necessary
    - too close to each other
    - prevent derailment at turnouts

- Parameters influencing braking distance

- Weather conditions
- Speed
- Landscape
- ... (anything else?)



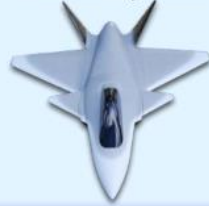


# Thermal model of an aircraft

## Boundary Conditions

- Flight mission (Mach, altitude, ...)
- Pressure, Temp., Humidity with altitude
- Sun radiation, Sun position,
- Pressure, Temperature, Humidity change over horizontal distance
- Non standard atmospheres model?
- Time varying heat loads from e.g. sensors

## Geometry Data



Model description  
[language/tool origin]

Functional Mock-up  
Unit (FMU)

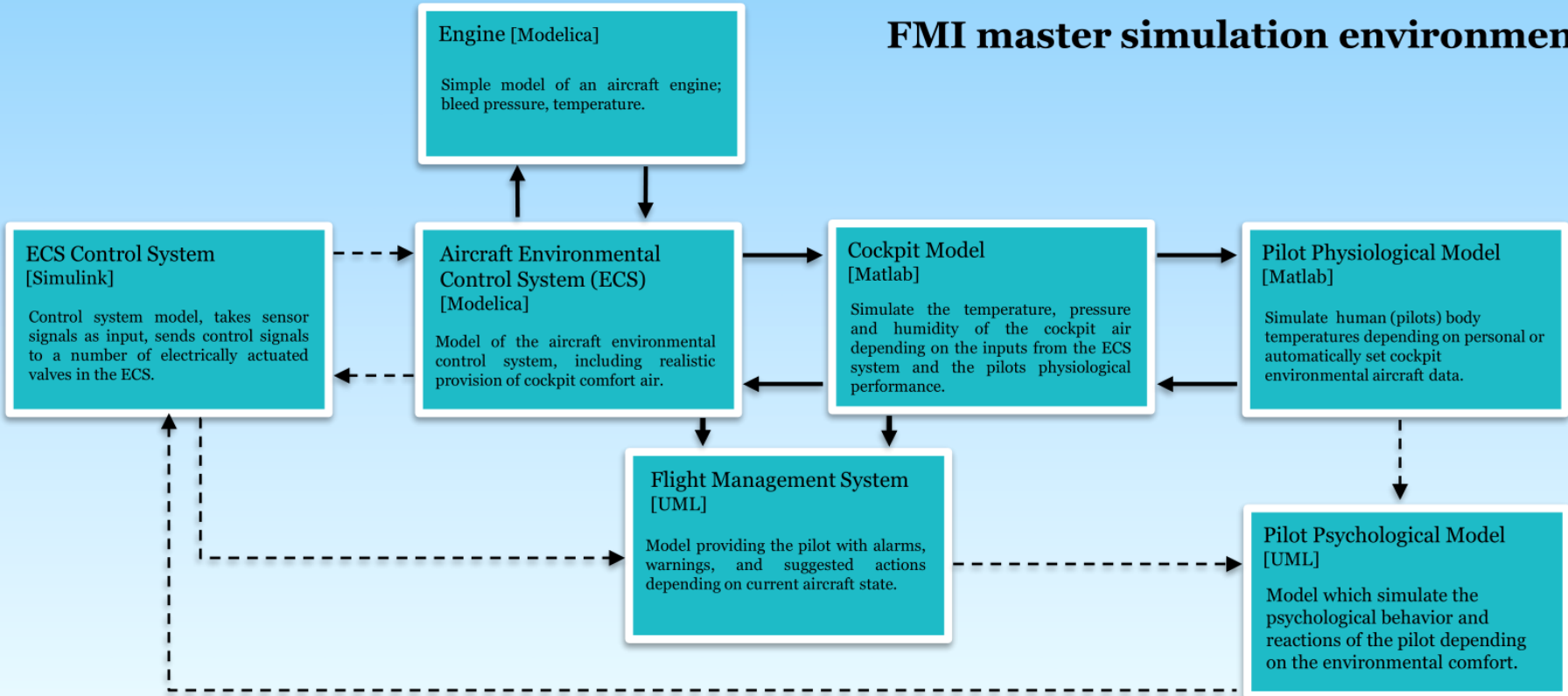


Physical connection



Information signal

## FMI master simulation environment



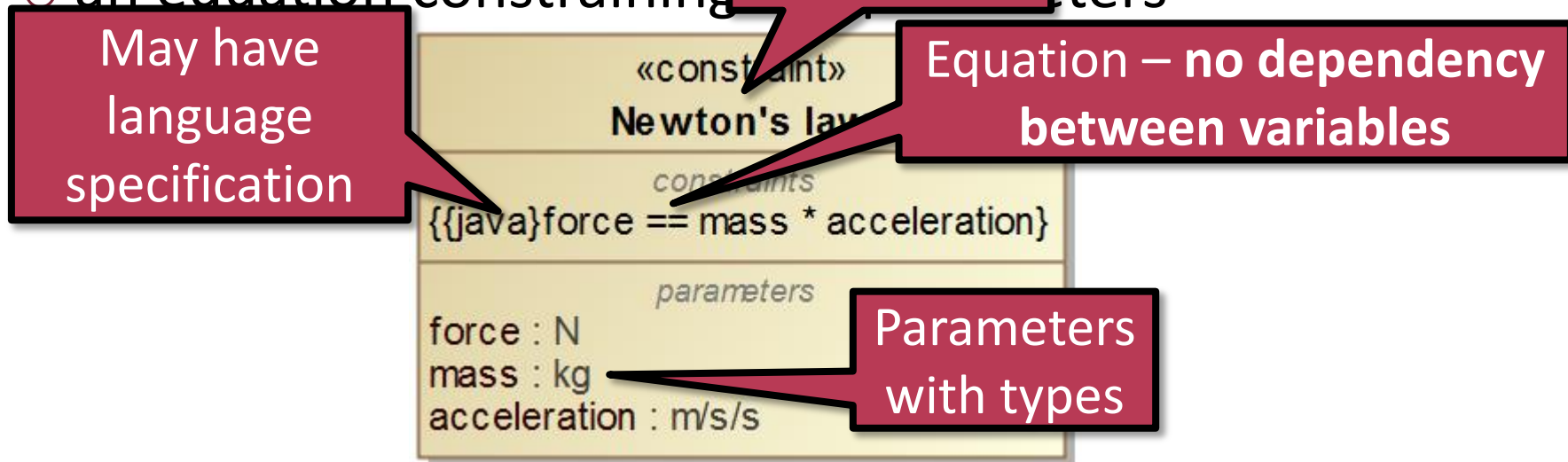
Copyright:  **SAAB**

# Constraints and physical parameters in SysML

Constraint blocks

# Constraint blocks

- **Constraint:** equations with parameters bound to the properties of the system
- **Constraint block:** supports the definition and the reuse of constraints. It holds
  - a set of parameters and
  - an equation constraining parameters



# Assignments and equations

- An assignment in a typical programming language is a **causal** connection, where the left hand side is the dependent variable:

$$y := x + 3$$

- An **acausal** connection is like a mathematical equation; there is no notion of inputs/outputs. So

$$y = x + 3$$

and

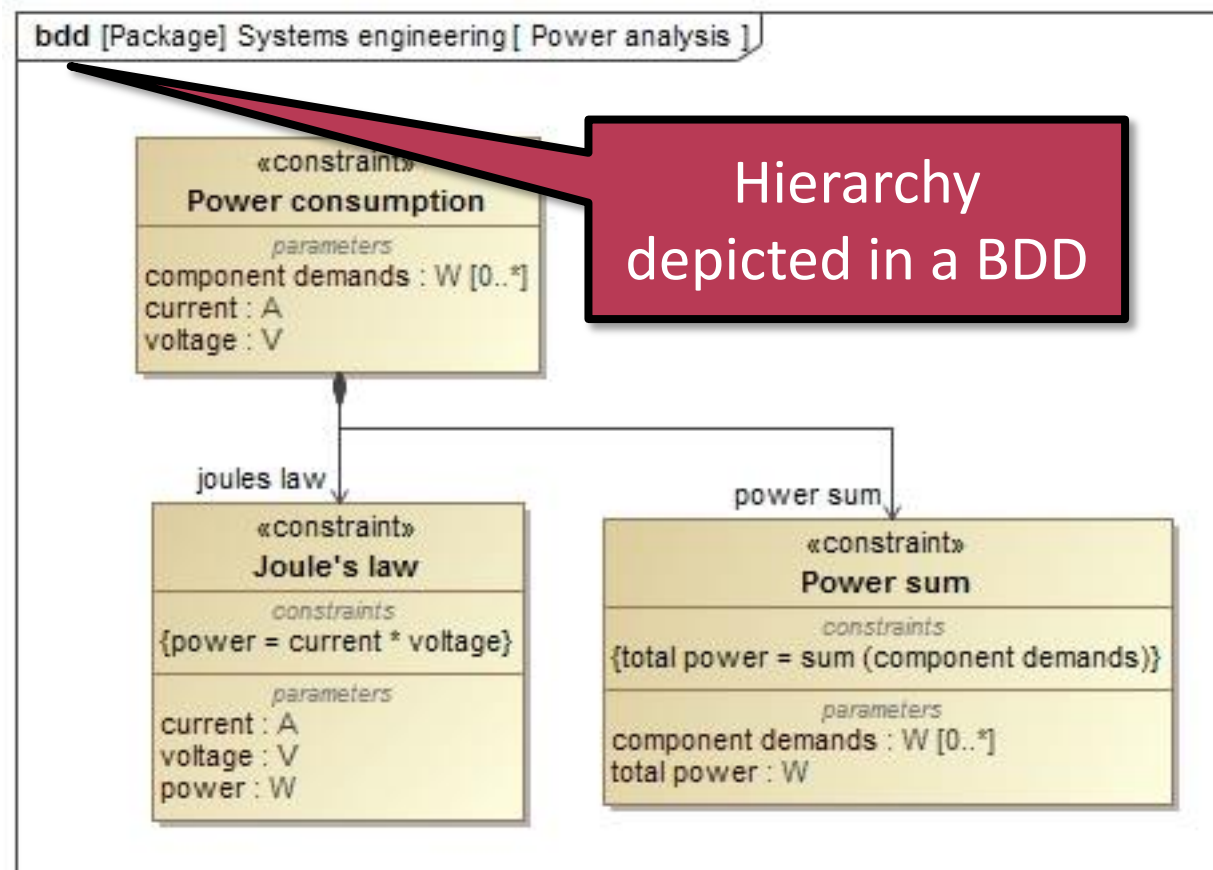
$$y - 3 - x = 0$$

have the same meaning.

- If any of the variables has a new value, it enforces that the other variables change accordingly.

# Constraint definition

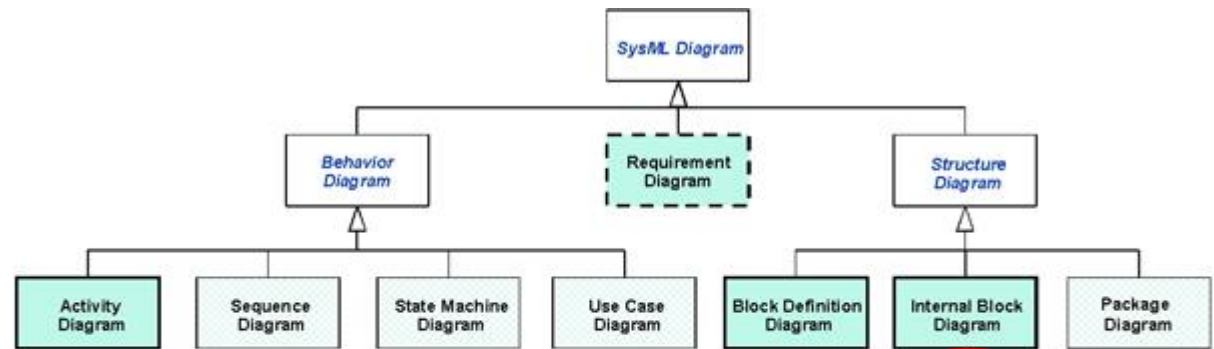
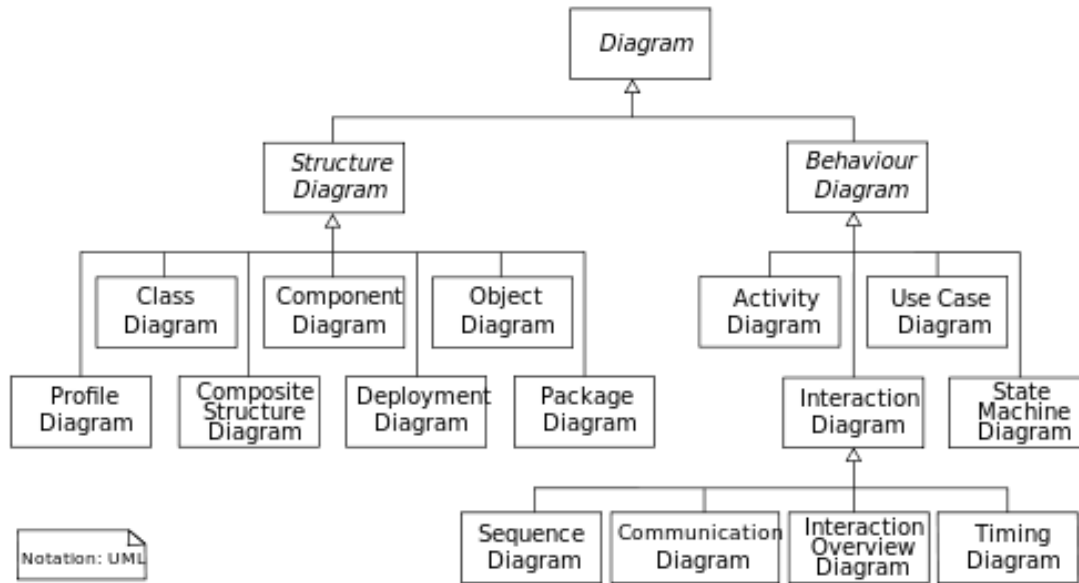
- **Composition** is used to define complex constraints from simple equations



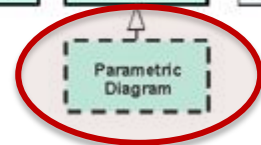
# Parametric diagram

Specification of bindings between system parameters

# Parametric Diagram (PAR)

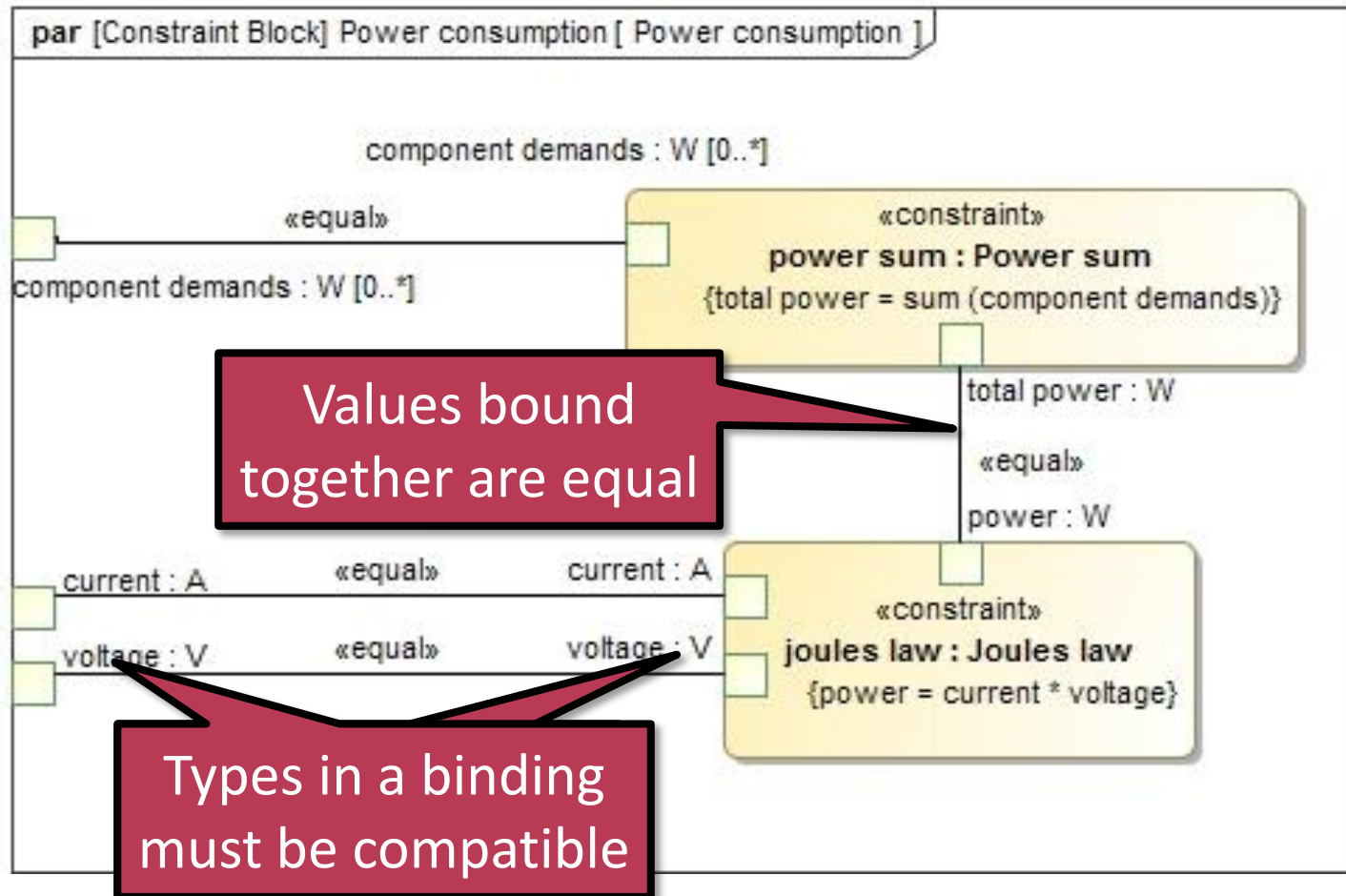


- Same as UML 2
- Modified from UML 2
- New diagram type



# Parameter bindings

- Goal: describe the application of constraints in a particular context





# Applications of parametrics

- Parametric specification
  - Define parametric relationships in the system structure
- Parametric analysis
  - Evaluating constraints on the system parameters to calculate values and margins for a given context
  - Checking design alternatives
  - Tool support: ParaMagic plug-in for MagicDraw
- There are modeling standards with better support for this modeling aspect...
  - ...such as Modelica

# Modelica

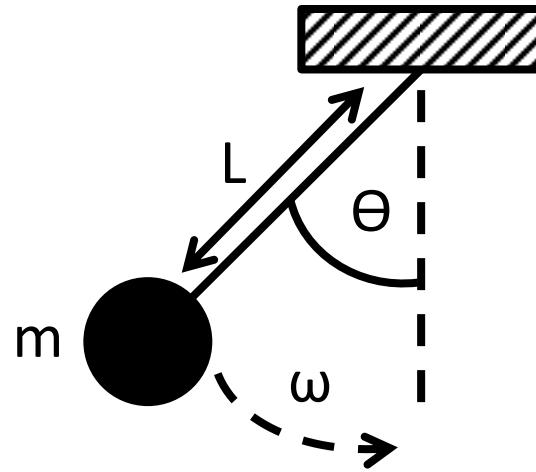
A language for modeling and simulating  
complex physical systems

# Overview of Modelica

- **Modelica** is an object-oriented, equation based language designed to model complex physical systems containing process-oriented subcomponents of different nature
  - Describing both continuous-time and discrete-time behaviour
- The **Modelica Standard Library** provides more than 1000 ready-to-use components from several domains
  - Full high-school + 1st year university physics (and much more)
- Implementations
  - Commercial e.g. by Dymola, Maplesoft, Wolfram MathCore
  - Open-source: JModelica
- Modeling and simulation IDE: OpenModelica

# Example: modeling a simple pendulum

- Simple pendulum



- Behavior of the pendulum as a function of time:

$$\begin{pmatrix} \dot{\theta}(t) \\ \dot{\omega}(t) \end{pmatrix} = \begin{pmatrix} \omega(t) \\ -\frac{g}{L}\theta(t) \end{pmatrix}$$

# Modelica code for simple pendulum

Model name

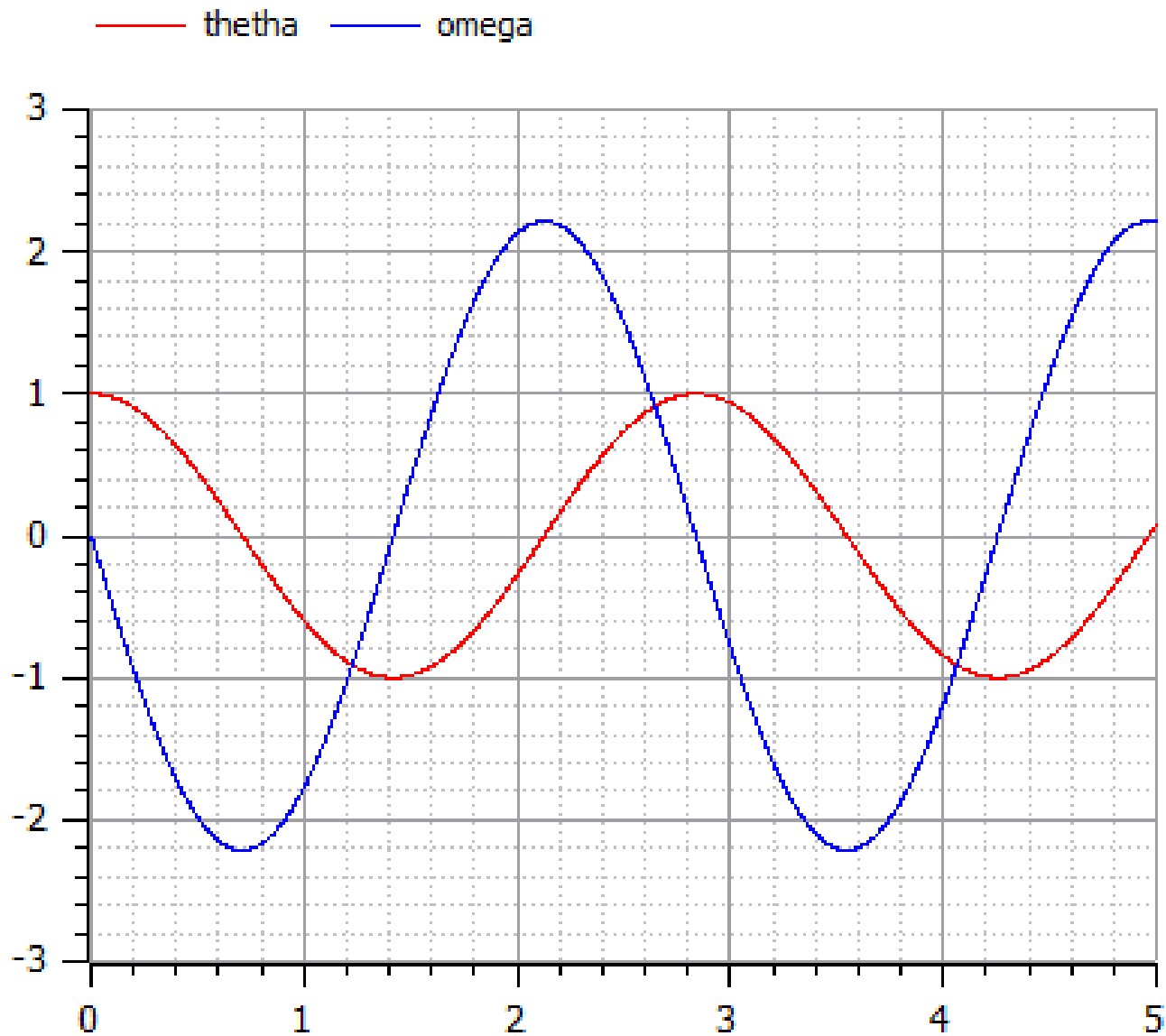
```
model SimplePendulum
  parameter Real L=2.0;
  constant Real g=9.81;
  Real theta (each start = 1.0);
  Real omega;
equation
  der(theta) = omega;
  der(omega) = -(g/L)*theta;
end SimplePendulum;
```

Continuous time variables, constants

Initial value

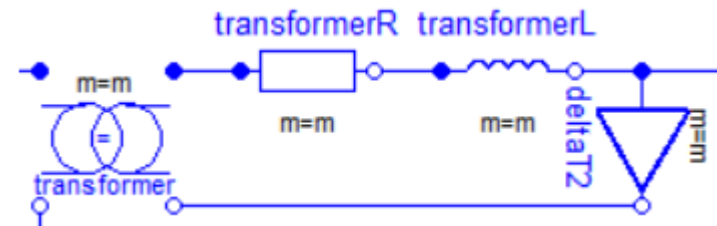
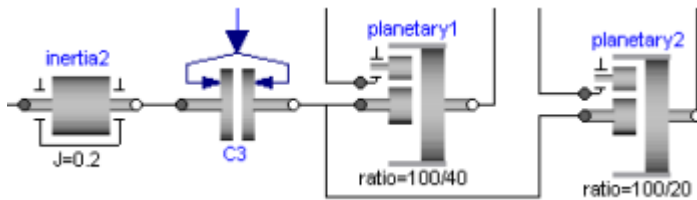
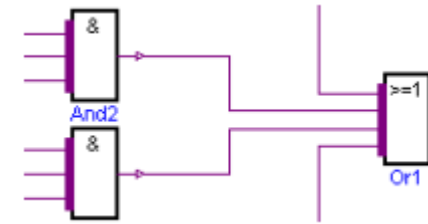
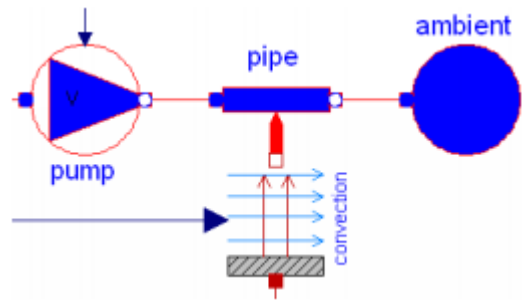
(Differential) equations

# Pendulum simulation results



# Modelica Standard Library

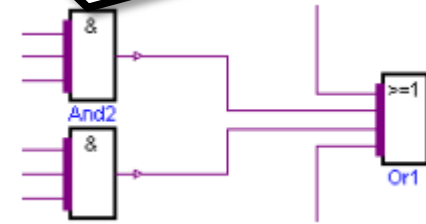
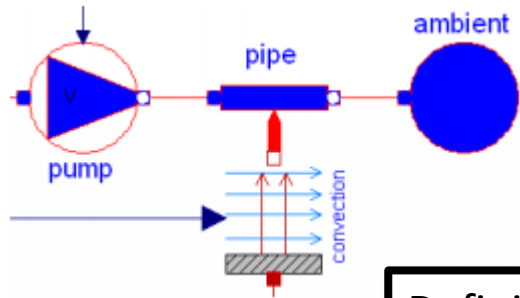
- Provides reusable building blocks (called classes) for Modelica models
- Version 3.2.1. has more than 1340 classes and models
- Various domains



# Modelica Standard Library

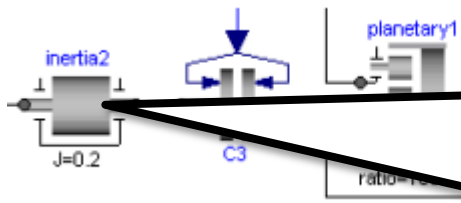
```

P Definition in Modelica:
M equation
V   auxiliary[1] = x[1];
V   for i in 1:n - 1 loop
      auxiliary[i + 1] = D.Tables.AndTable[auxiliary[i], x[i + 1]];
    end for;
V   y = pre(auxiliary[n]);
  
```



```

Definition in Modelica:
equation
  phi = flange_a.phi;
  phi = flange_b.phi;
  w = der(phi);
  a = der(w);
  J*a = flange_a.tau + flange_b.tau;
  
```



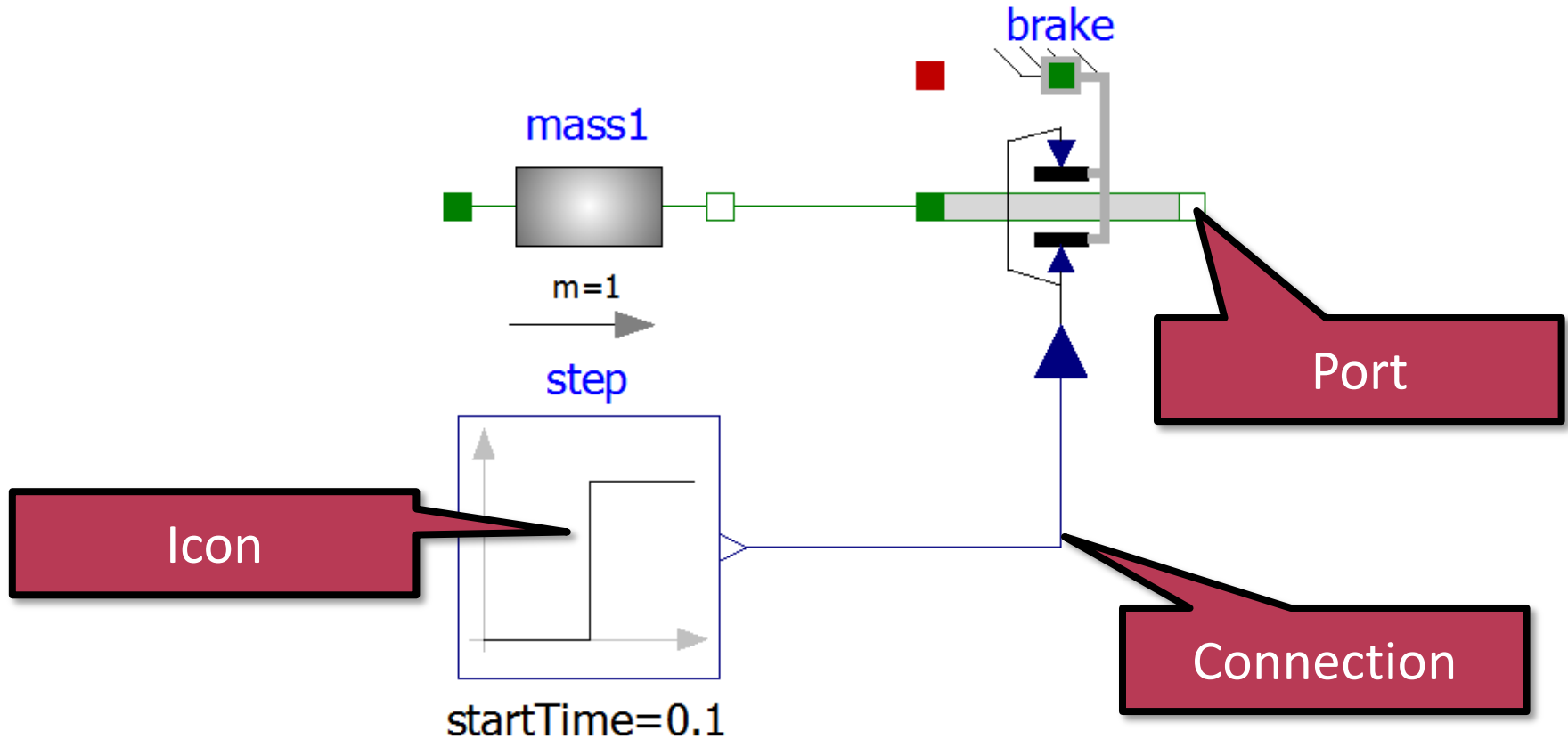


# Modelica and Simulation

- Simulating a model means to calculate the values of its variables at certain time instants
- Advantages
  - Observing dangerous/expensive behaviour at low cost with no risks
  - Resolves scaling issues (size, duration)
- Different algorithms and strategies for simulation
  - The task is to solve Ordinary Differential Equations (ODEs) generated from the model
  - Numerical techniques

# Example plant model – train brakes

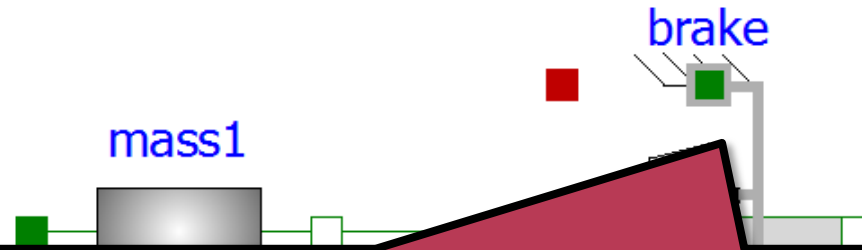
- Physical model for braking system carrying a mass



- Graphical notation in OpenModelicaEditor

# Example plant model – train brakes

- Physical model for braking system carrying a given mass



Class

Path: Modelica.Mechanics.Translational.Components.Brake

Comment: Brake based on Coulomb friction

Parameters

mue_pos	<input type="text" value="[0, 0.5]"/>	[v, f] Positive sliding friction characteristic ( $v \geq 0$ )
peak	<input type="text" value="1"/>	peak*mue_pos[1,2] = Maximum friction force for $v == 0$
cgeo	<input type="text" value="1"/>	Geometry constant containing friction distribution assumption
fn_max	<input type="text" value="1"/>	N Maximum normal force
useSupport	<input type="text" value="false"/> <input type="checkbox"/>	= true, if support flange enabled, otherwise implicitly grounded
useHeatPort	<input type="text" value="false"/> <input type="checkbox"/>	=true, if heatPort is enabled

# Example plant model – train brakes

```
model BrakeExample
```

```
  Brake brake (  
    fn_max=1  
    useSupport=false);
```

```
  Mass mass1 (  
    m=1,  
    s(fixed=true),
```

```
    v(start=  
  Step step (  
    startTime  
    height=2)
```

Brake, Mass, and Step are inbuilt classes to Modelica Library

Can describe both causal and acausal connections between ports

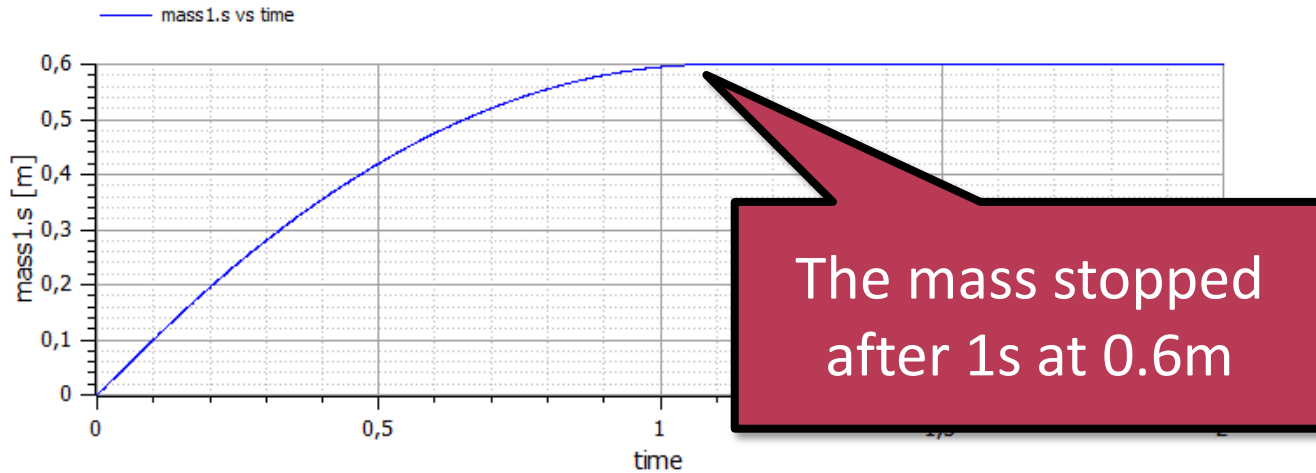
```
equation
```

```
  connect(mass1.flange_b, brake.flange_a);  
  connect(step.y, brake.f_normalized);
```

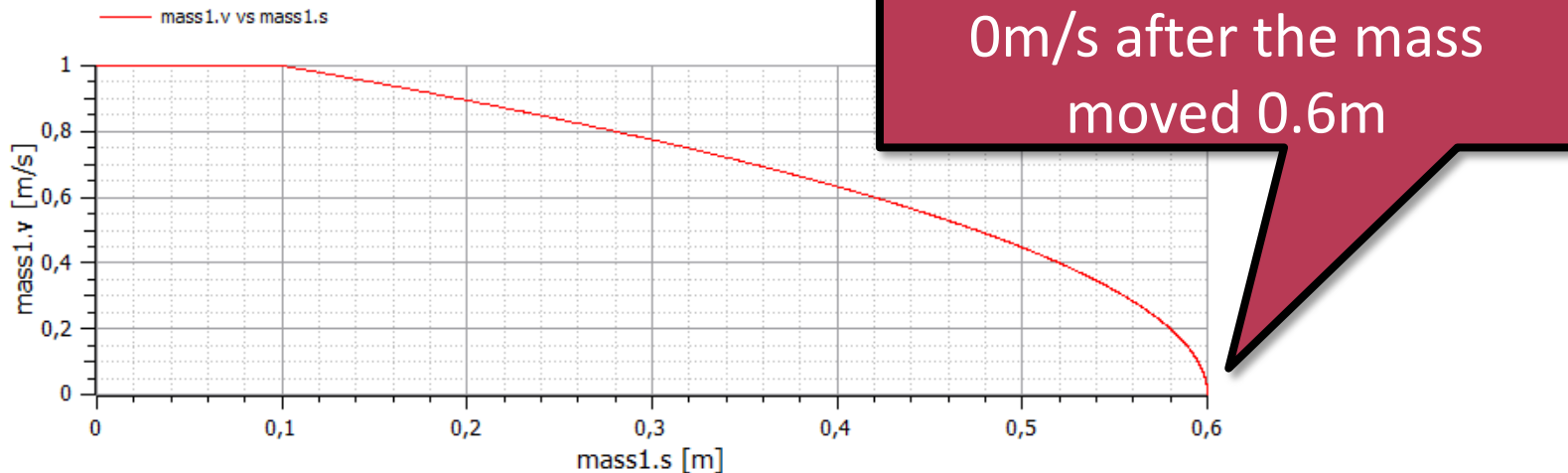
```
end BrakeExample;
```

# Brake times and distance

- Plot values w.r.t. time (displacement)



- X-Y plot (speed w.r.t. displacement)



# Summary

- Complex system design requires modeling of physical parameters
  - SysML constraint block, parametric diagram
- Modeling both discrete-time and continuous-time behaviour of cyber-physical systems
  - Modeling language for this purpose: Modelica
- Connecting models to study joint behavior
  - Simulation of models is especially useful when implementing and testing the system is expensive