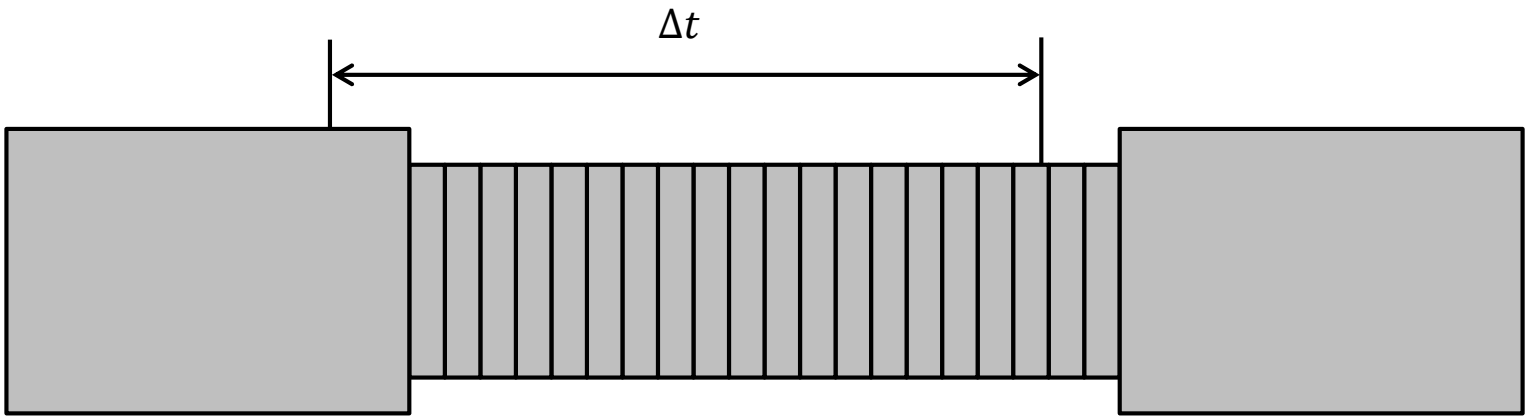
TLM-based Co-simulation using FMI and Direct Tool Connections

Robert Braun

LINKÖPING UNIVERSITY

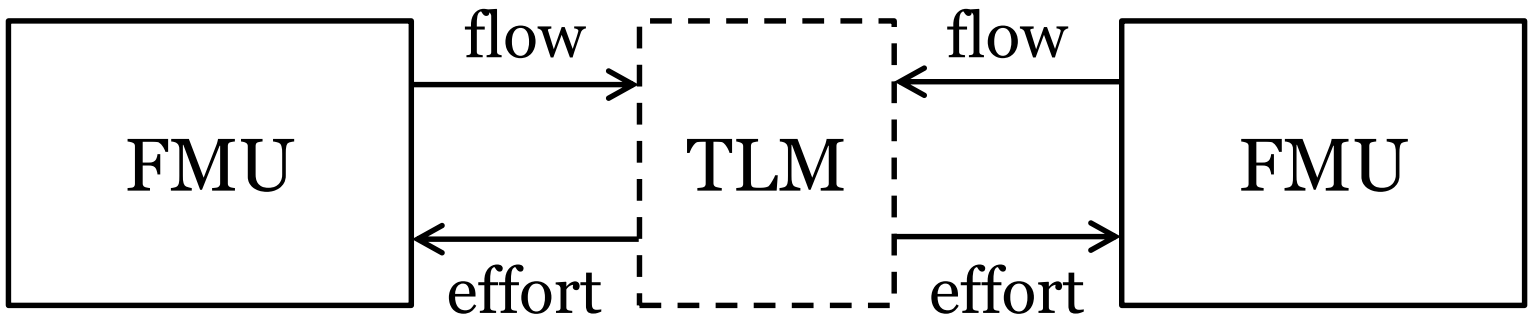
Transmission Line Modeling

Every physical element has a natural time delay:

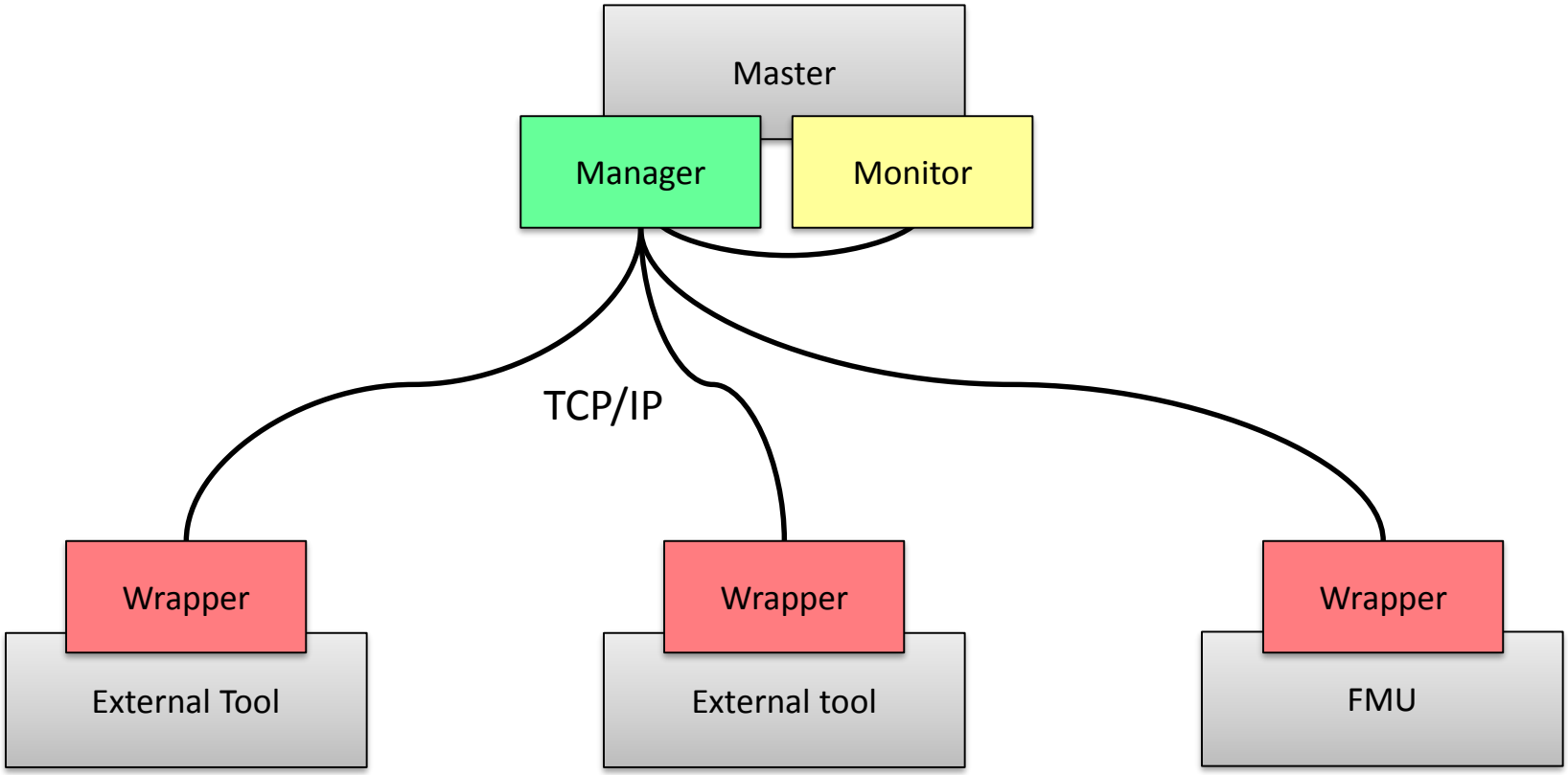


E.g. deformable bodies, compressible fluids…

Transmission Line Modeling



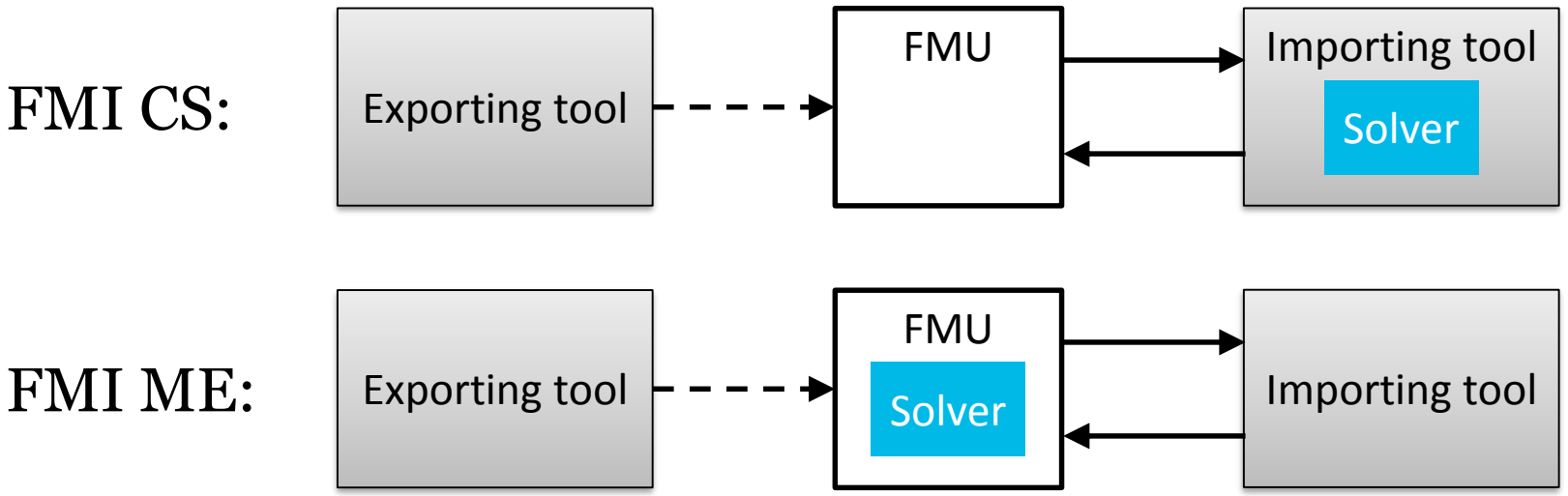
OMTLMSimulator Overview



Functional Mockup Interface (FMI)

Open standard for simulation tool coupling

- FMI for co-simulation (FMI CS)
- FMI for model exchange (FMI ME)

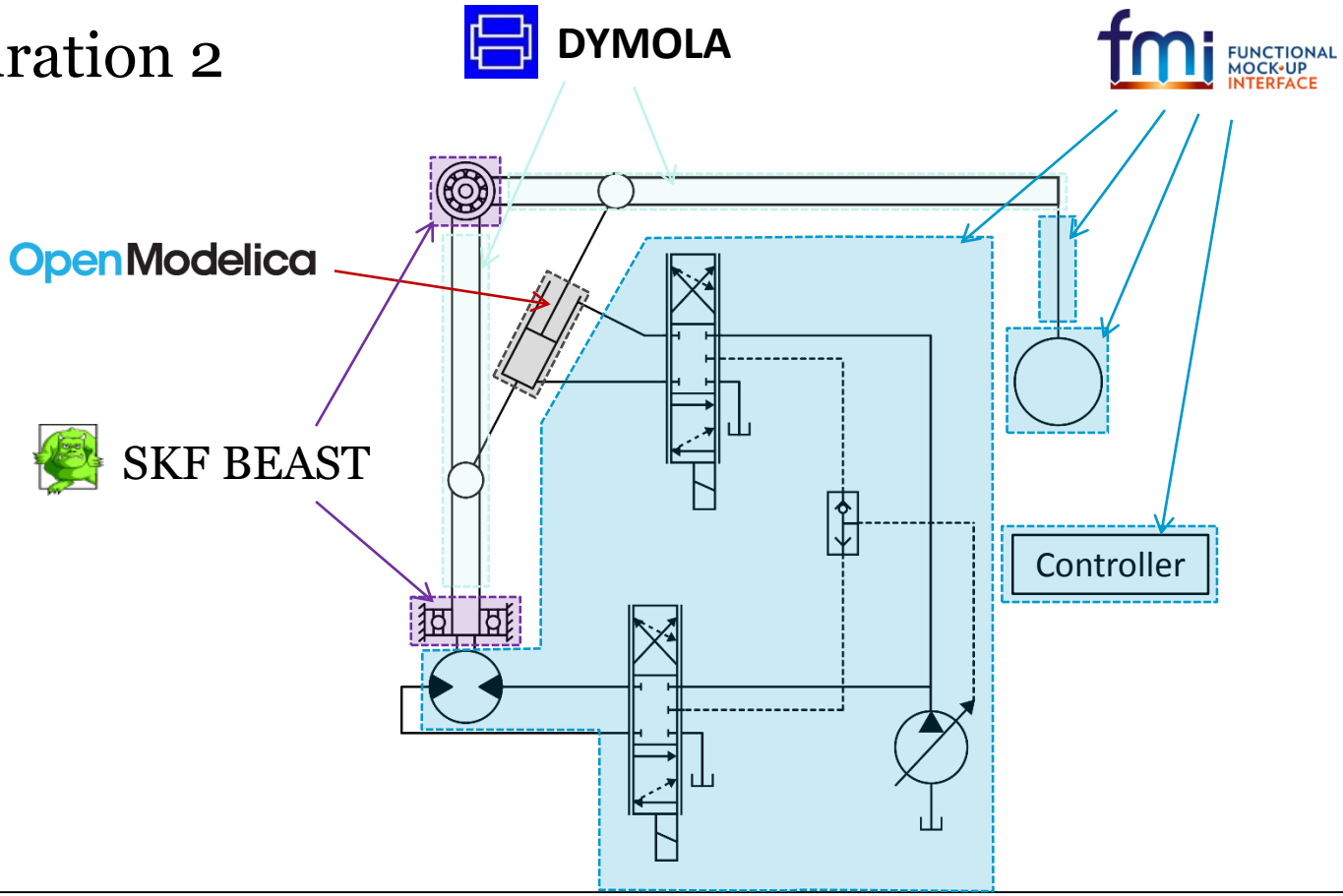


Tool support

- FMI
- Direct tool connections
  - OpenModelica
  - Dymola
  - Wolfram SystemModeler
  - Hopsan
  - Simulink
  - (Adams)

Demonstrator Model

Configuration 2



Demonstrator Model

Configurations

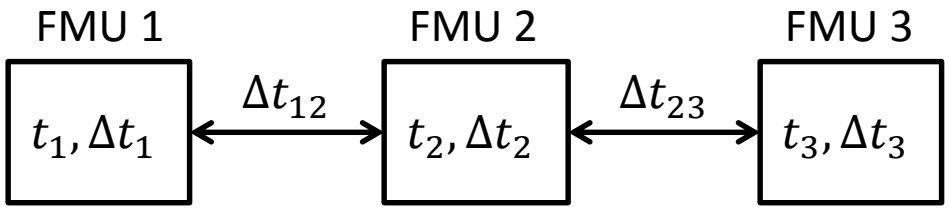
Part	Configuration 1	Configuration 2
Boom	OpenModelica (FMI ME)	Dymola
Jib	OpenModelica (FMI ME)	Dymola
Sling	OpenModelica	Dymola (FMI CS)
Load	OpenModelica	Dymola (FMI CS)
Slewing Bearing	BEAST	BEAST
Jib Bearing	BEAST	BEAST
Piston	OpenModelica	OpenModelica
Hydraulic Circuit	Hopsan	Hopsan (FMI CS)
Controller	Simulink	Hopsan (FMI CS)

Demonstrator Model

-

Numerical Robustness

Distributed asynchronous co-simulation



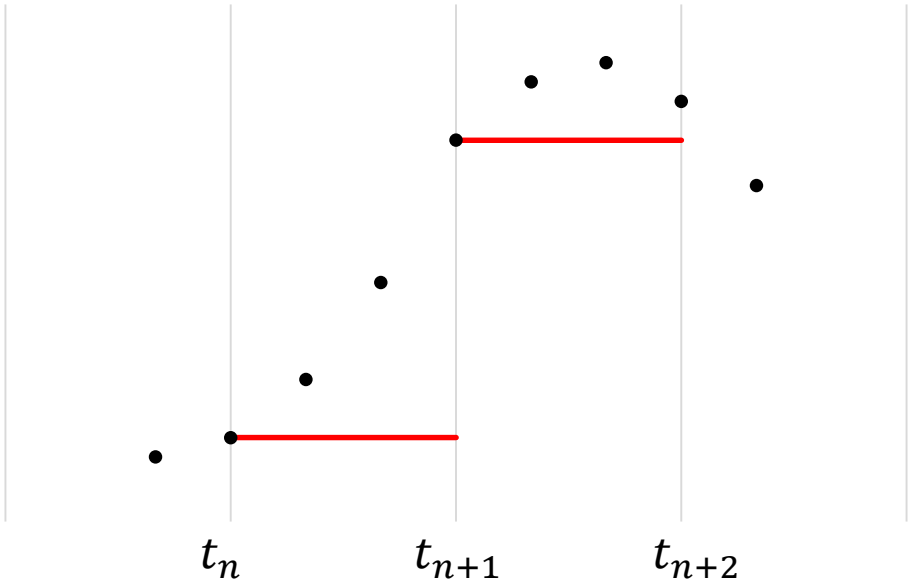
$$\begin{cases} t_1 \neq t_2 \neq t_3 \\ \Delta t_1 \neq \Delta t_2 \neq t_3 \\ \Delta t_{12} \neq \Delta t_{23} \end{cases}$$

Variable step-size, iterative solvers etc…

- Stability requires interpolation of inputs
- Can interpolate $x(t_n < t < t_{n+1})$ since $x(t > t_{n+1})$ is known

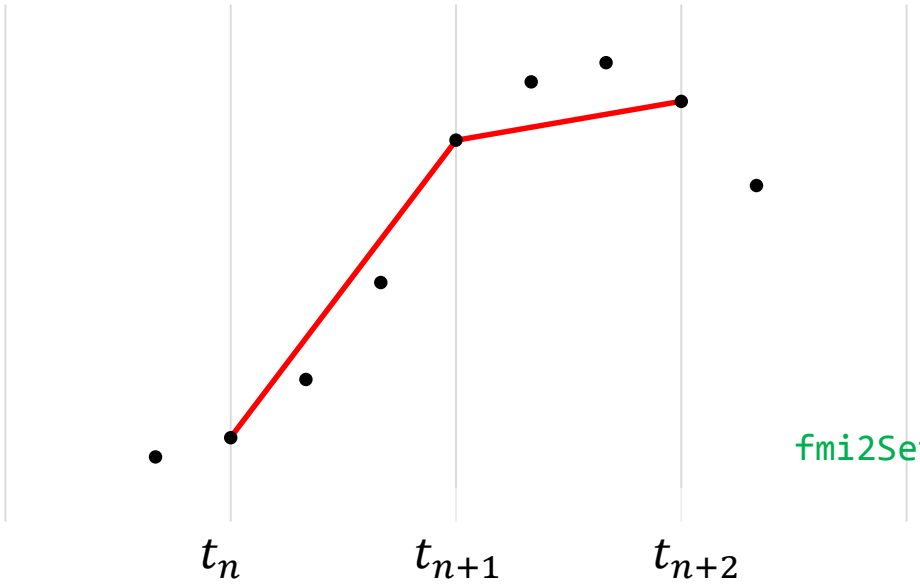
How handle interpolation with FMI for co-simulation?

Constant Extrapolation



$$x(t_n \leq t \leq t_{n+1}) = x(t_n)$$

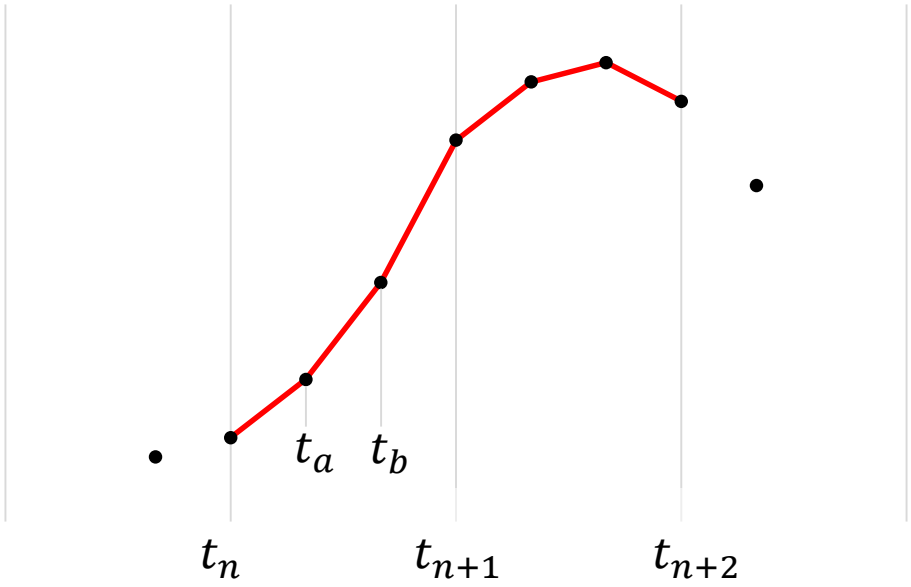
Coarse-grained Interpolation



`fmi2SetRealInputDerivatives()`

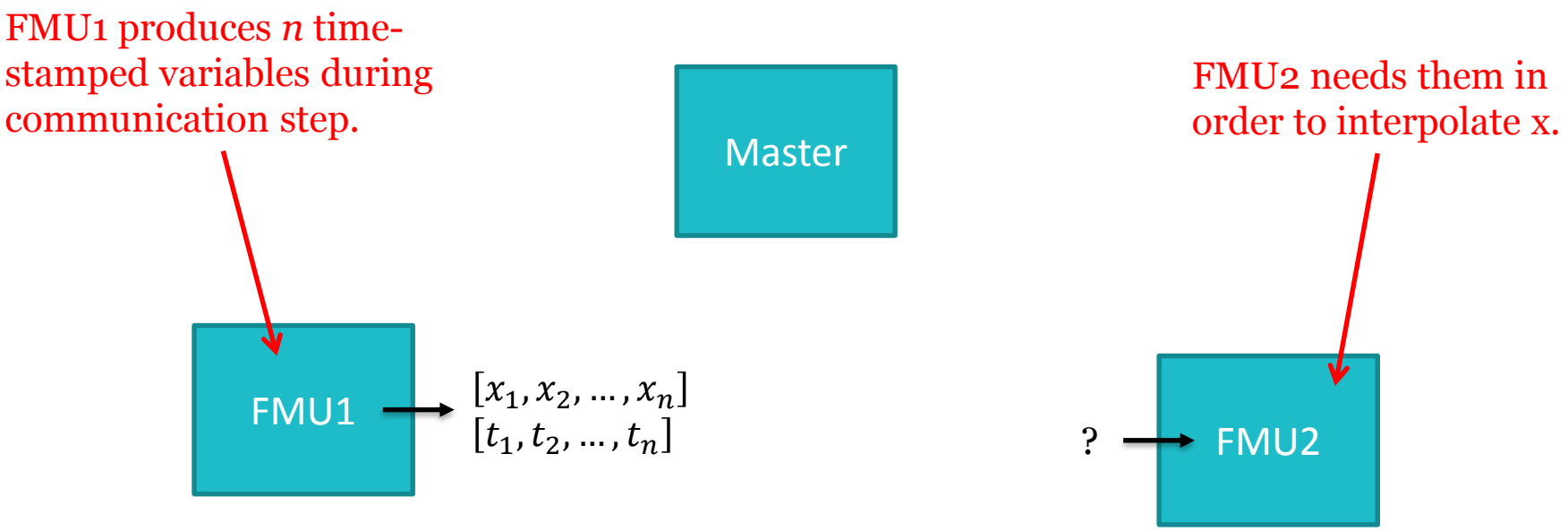
$$x(t_n \leq t \leq t_{n+1}) = x(t_n) + \frac{x(t_{n+1}) - x(t_n)}{t_{n+1} - t_n}(t - t_n) = x(t_n) + \hat{x}(t - t_n)$$

Fine-grained Interpolation



$$x(t_n \le t \le t_{n+1}) = x(t_a) + \big(x(t_b) - x(t_a)\big)\frac{t - t_a}{t_b - t_a}, \qquad \begin{cases} t_n \le t_j \le t_{n+1} \\ t_n \le t_k \le t_{n+1} \end{cases}$$

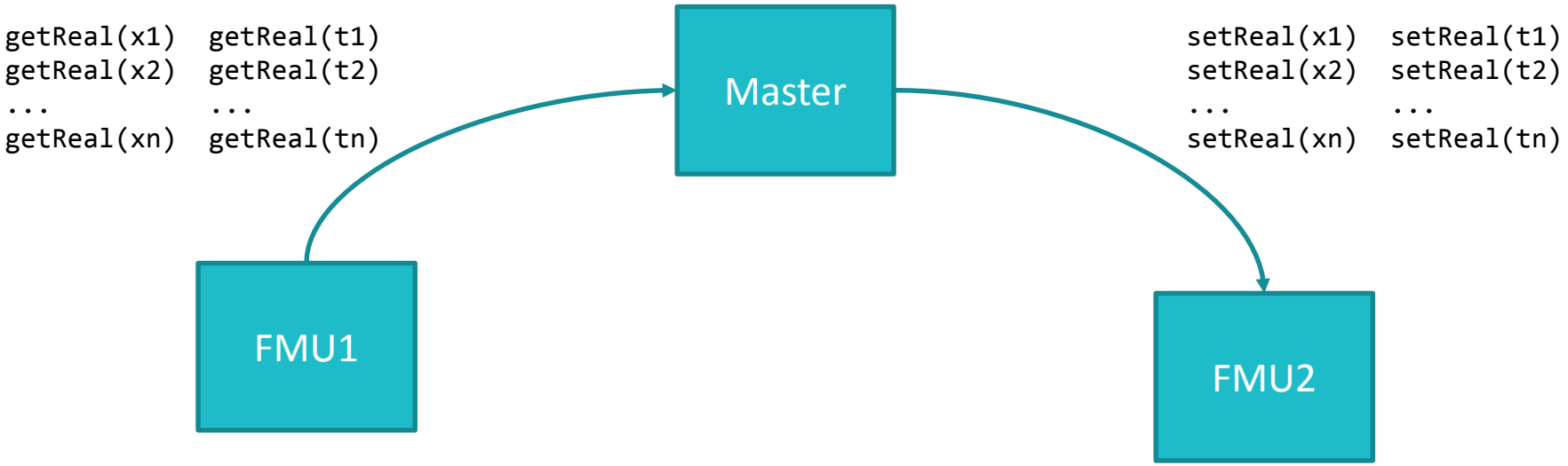
Fine-grained interpolation with FMI CS?



Fine-grained interpolation with FMI CS?

Current standard: Using `getReal()` and `setReal()`

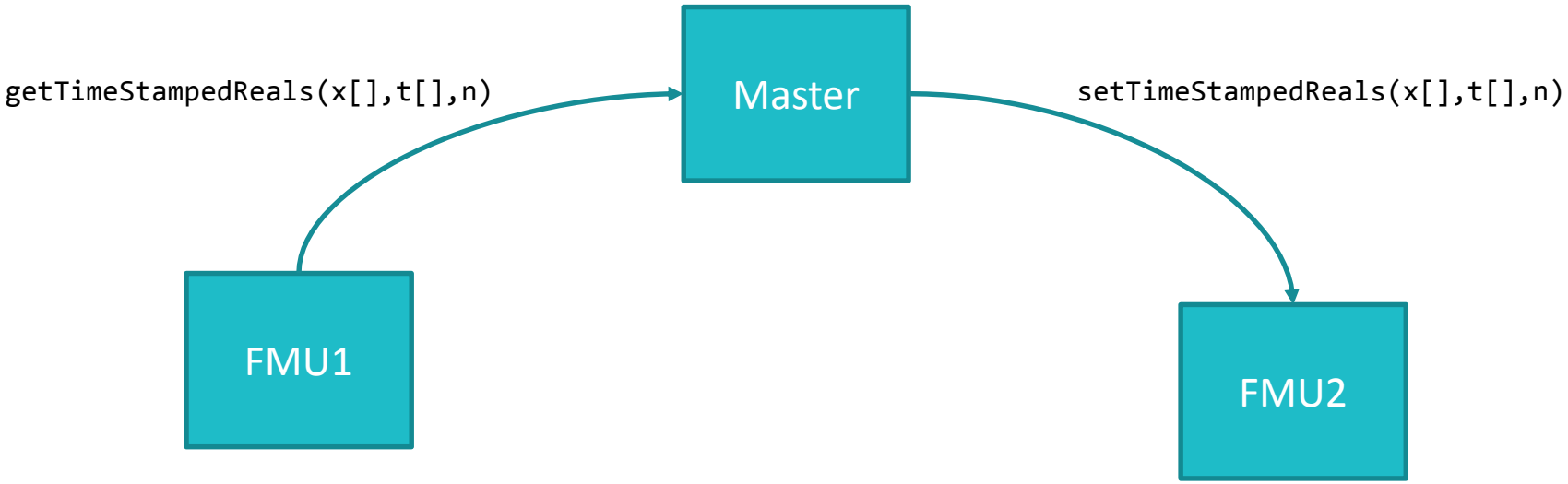
- Impractical
- Fixed number of samples



Fine-grained interpolation with FMI CS?

Extension to standard

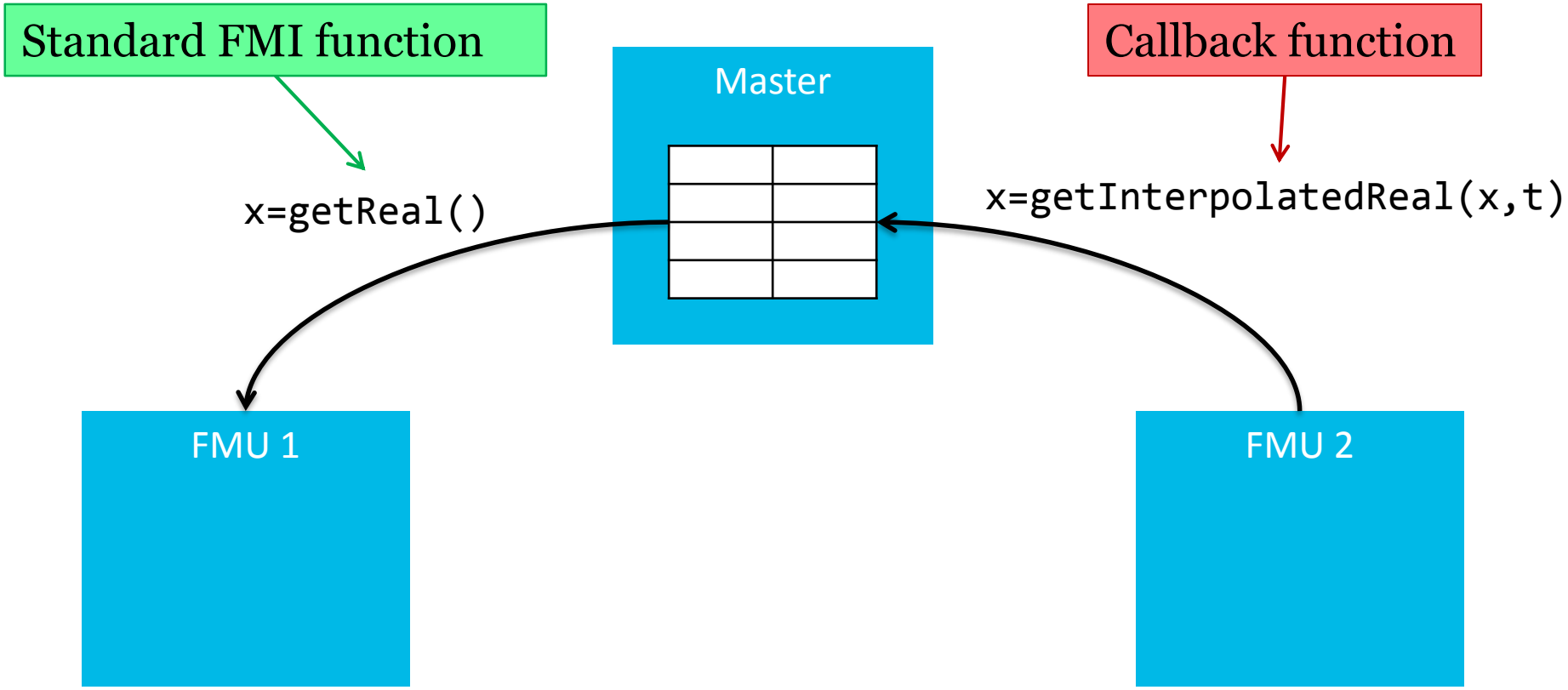
- Setting and getting time stamped variables
- Arrays of arbitrary length



Required extention to standard

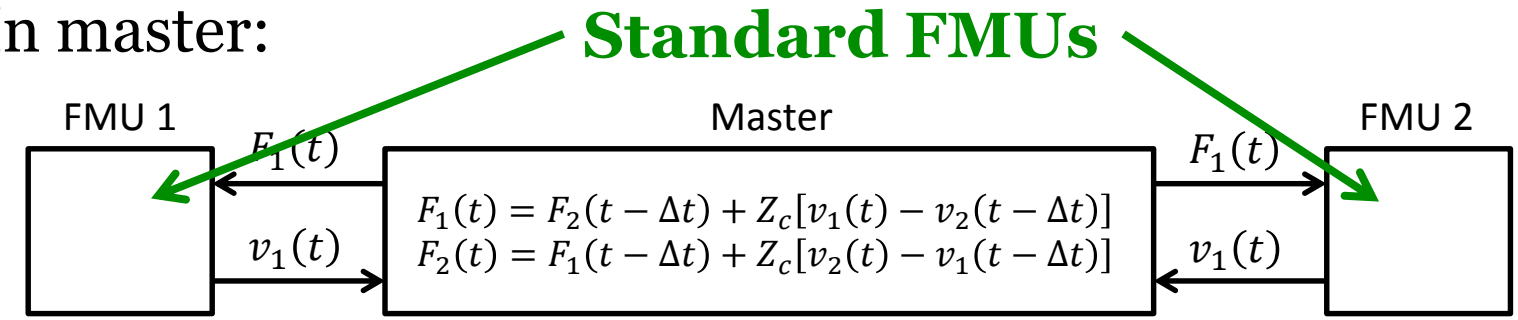
```
fmi2Status fmi2GetTimeStampedReals (fmi2Component c,
                                    const fmi2ValueReference vr,
                                    size_t nval,
                                    fmi2Real value[]
                                    fmi2Real times[]);

fmi2Status fmi2SetTimeStampedReals (fmi2Component c,
                                    const fmi2ValueReference vr,
                                    size_t nval,
                                    const fmi2Real values[]
                                    const fmi2Real times[]);
```

Alternative: Interpolation in master

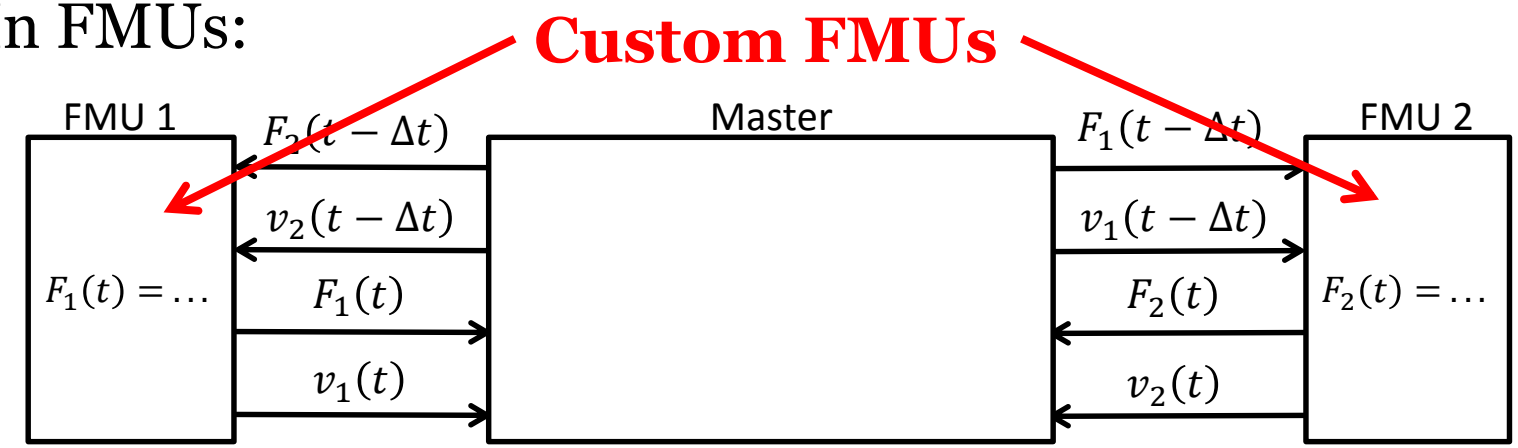


Interpolate where?

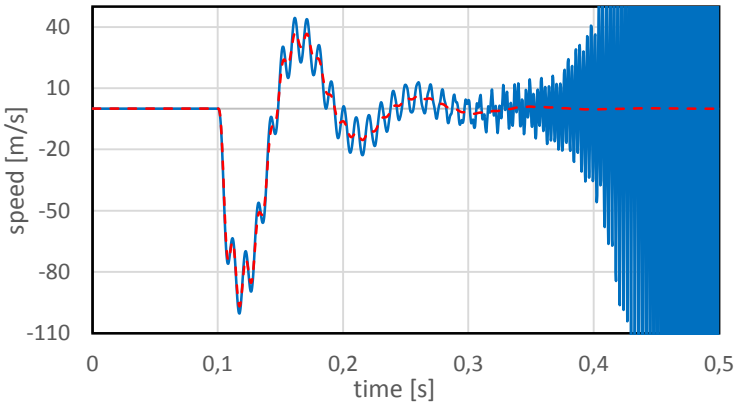
In master:



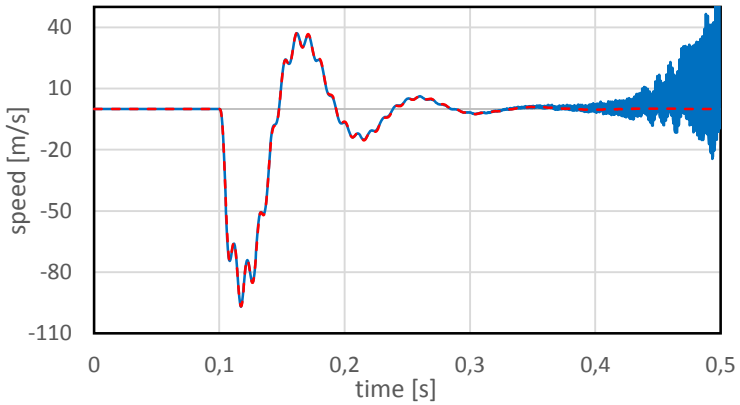
In FMUs:



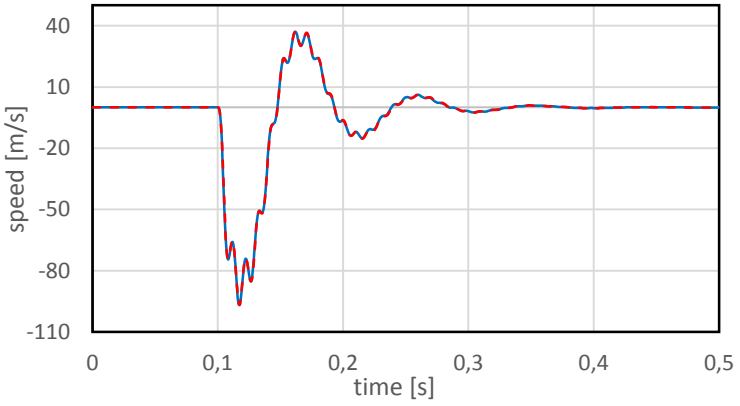
Numerical Experiments



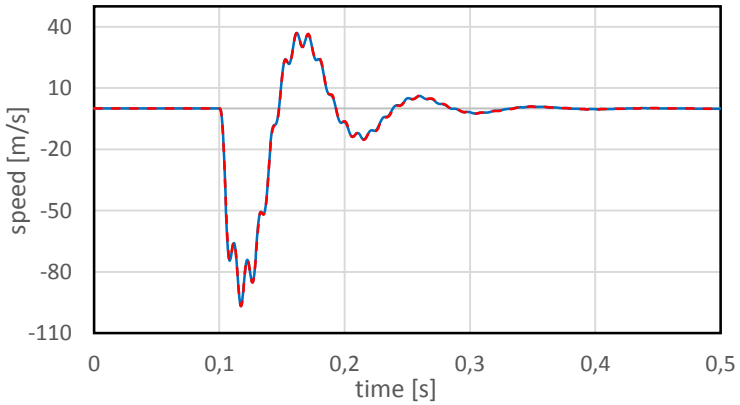
Constant extrapolation



Coarse-grained interpolation

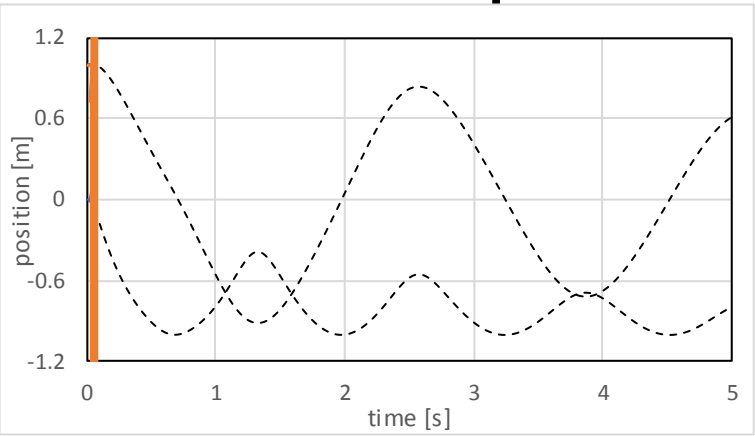


Fine-graind interpolation in FMU

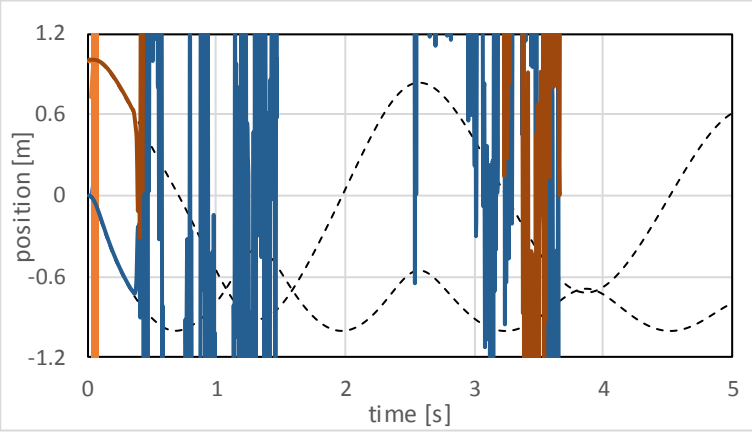


Fine-graind interpolation in master

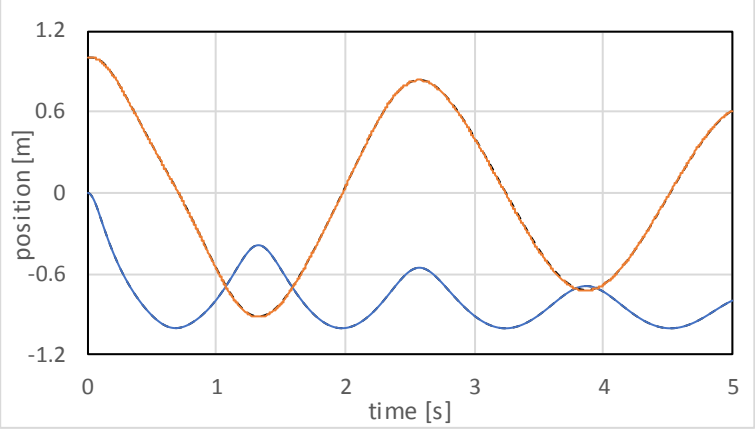
Numerical Experiments



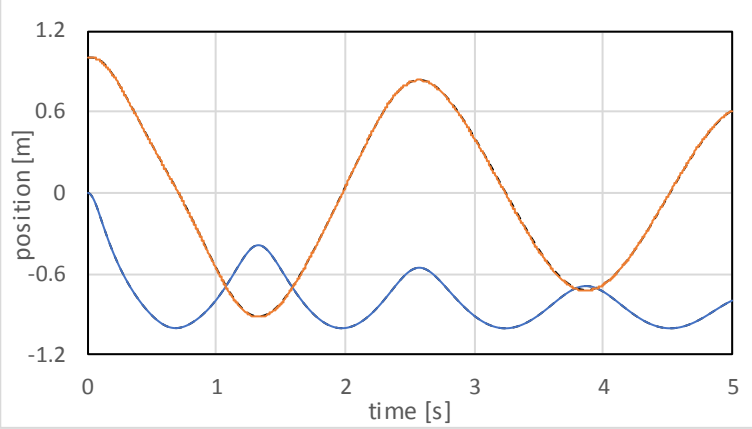
Constant extrapolation



Coarse-grained interpolation



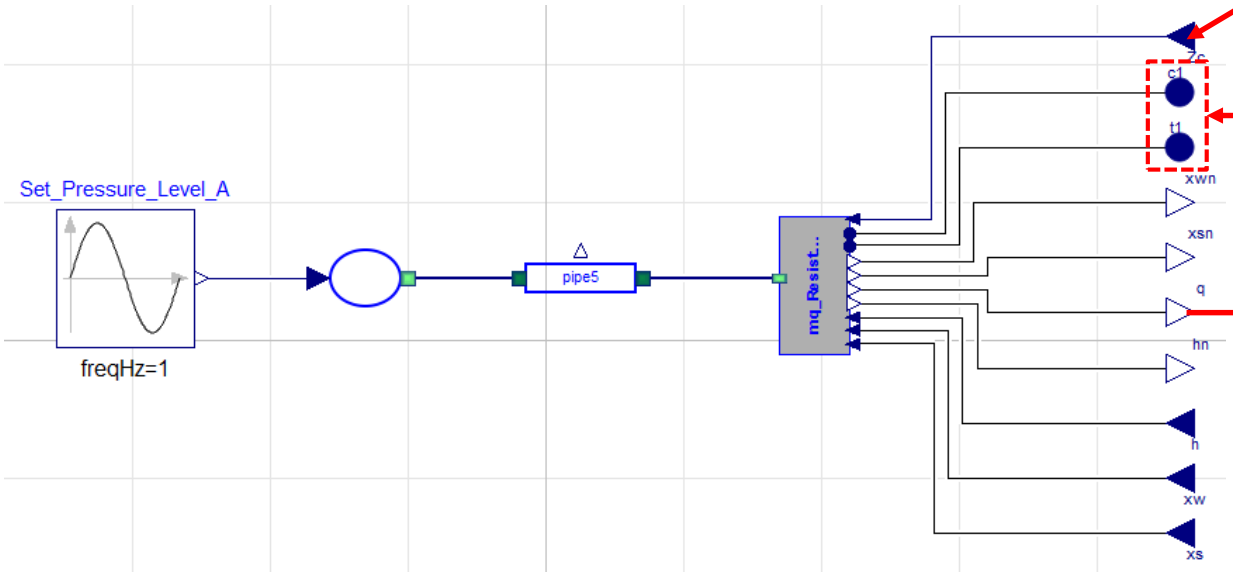
Fine-graind interpolation in FMU



Fine-graind interpolation in master

Numerical Experiments

- Hydraulic cooling system



Characteristic impedance

Interpolation data (wave variable)

Volume flow

Numerical Experiments

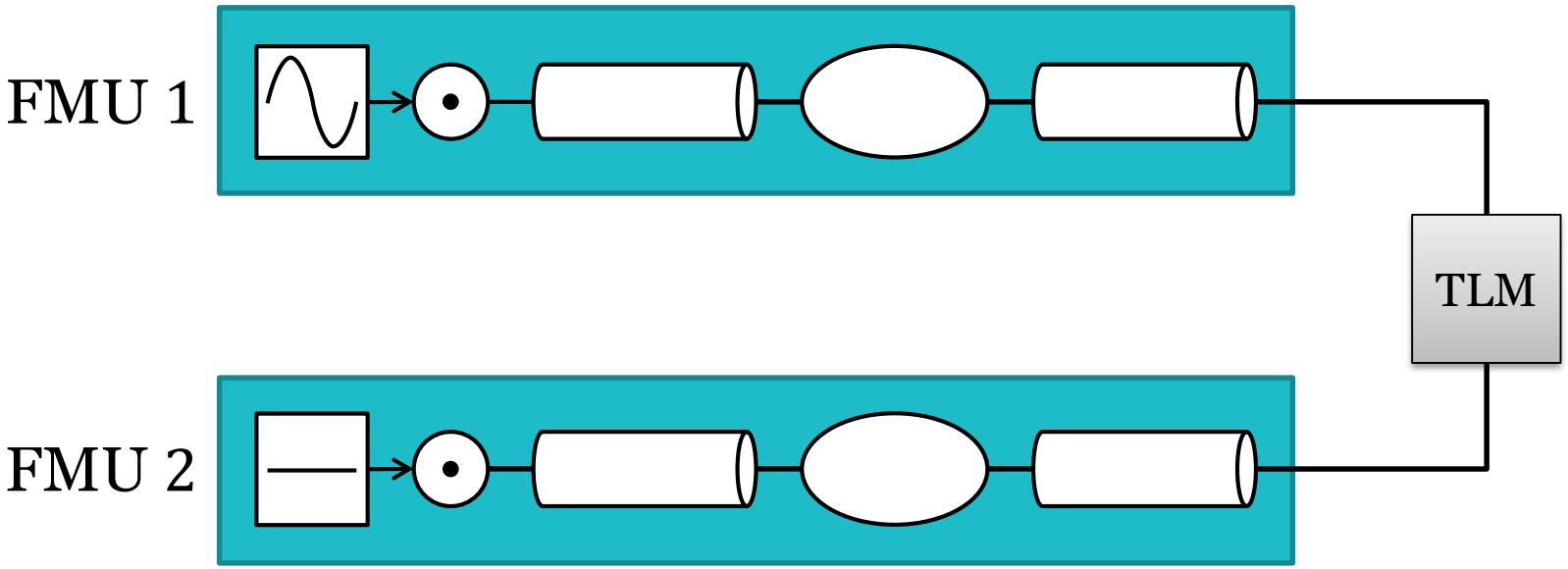
```
model Interpolate
  Modelica.Blocks.Interfaces.RealVectorInput c1[n];
  Modelica.Blocks.Interfaces.RealVectorInput t1[n];

  Modelica.Blocks.Interfaces.RealOutput pi;
  Modelica.Blocks.Interfaces.RealInput q1;

  Modelica.Blocks.Interfaces.RealInput Zc;

  parameter Integer n;

protected
  Real ci;

equation
  ci = PopulateInterpolate(t1,c1,time);
  pi = getPressure(ci,Zc,q1);

end Interpolate;
```

```
function PopulateInterpolate
  input Real t[:];
  input Real c[:];
  input Real tc;
  output Real ci;

algorithm
  ci:=Modelica.Math.Vectors.interpolate(t,c,tc);

end PopulateInterpolate;
```

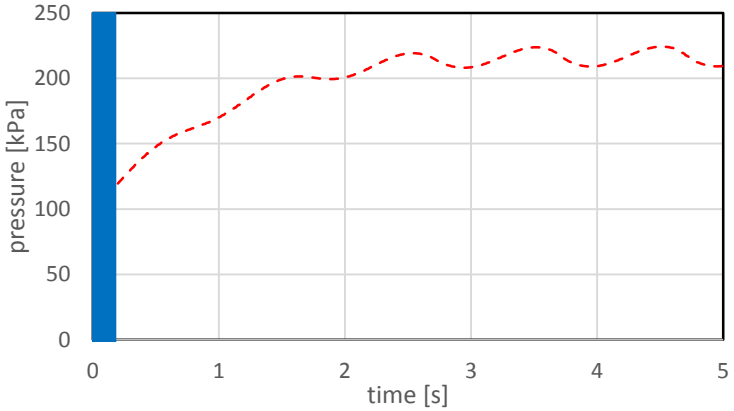
```
function getPressure
  input Real c1;
  input Real q1;
  input Real Zc;
  output Real p1;

algorithm
  p1:=Zc*q1+c1;

end getPressure;
```

Numerical Experiments

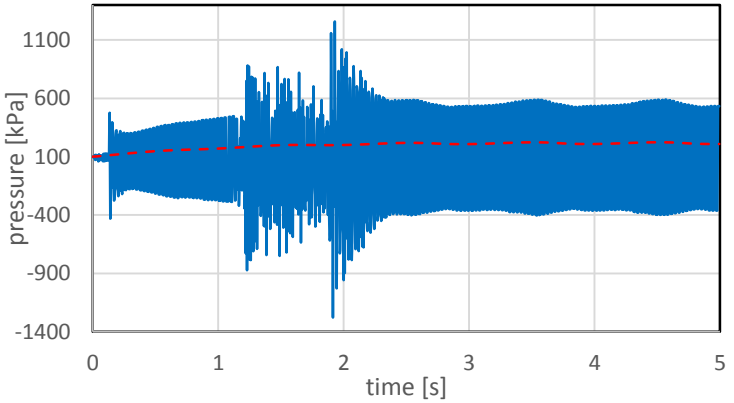
- Thermodynamic connection



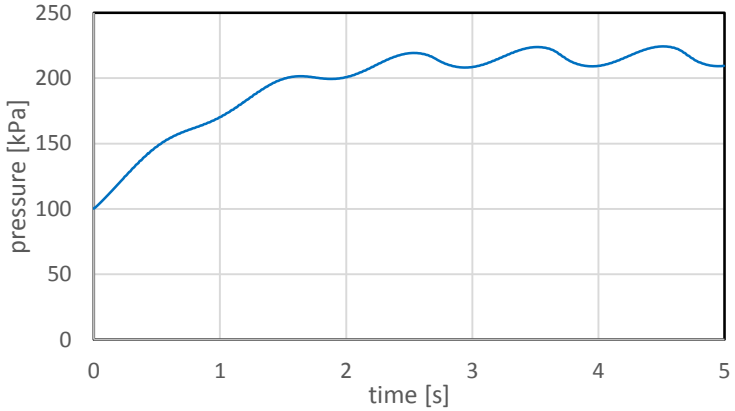
Numerical Experiments



Constant extrapolation

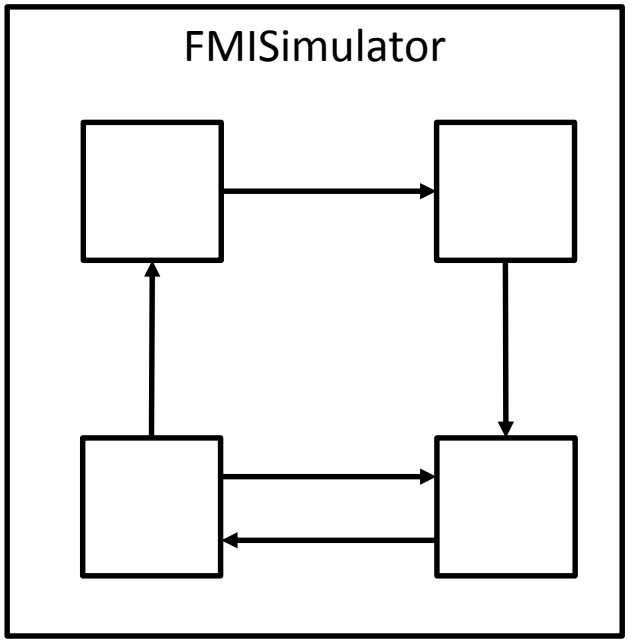


Coarse-grained interpolation

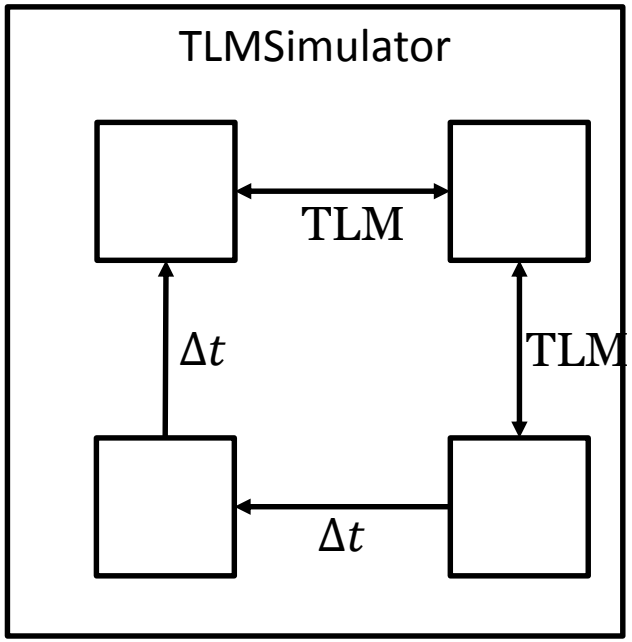


Fine-graind interpolation in FMU

Merging TLMSimulator and FMISimulator

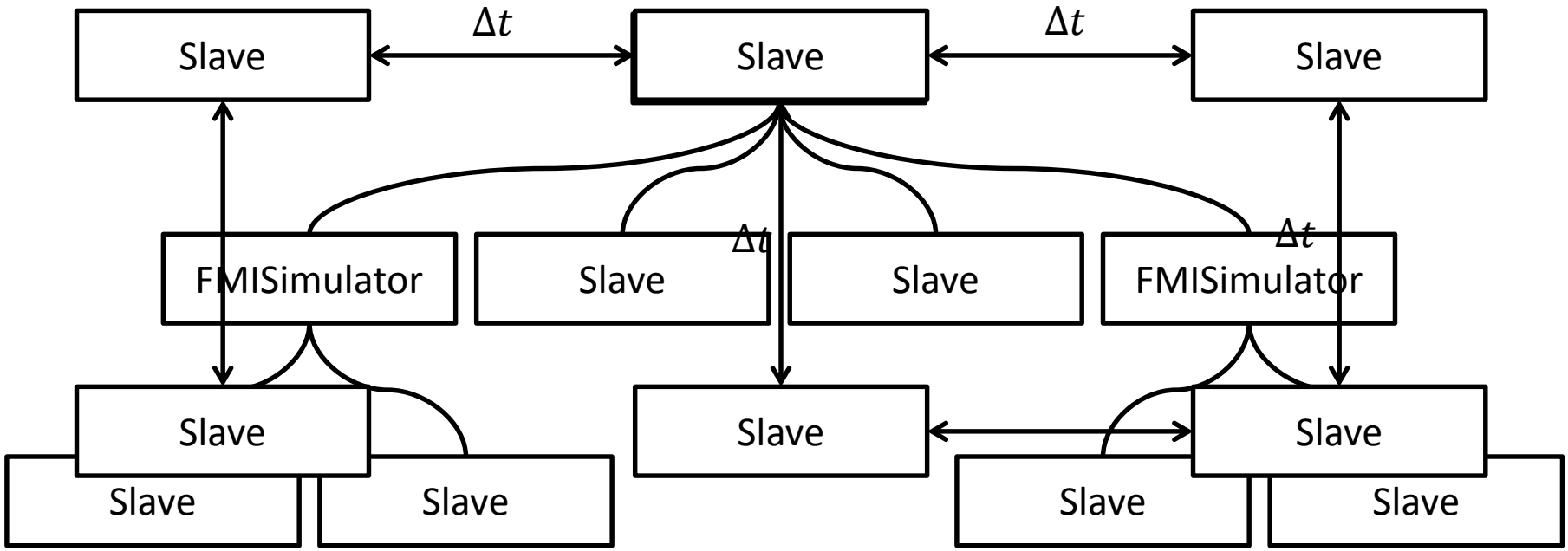


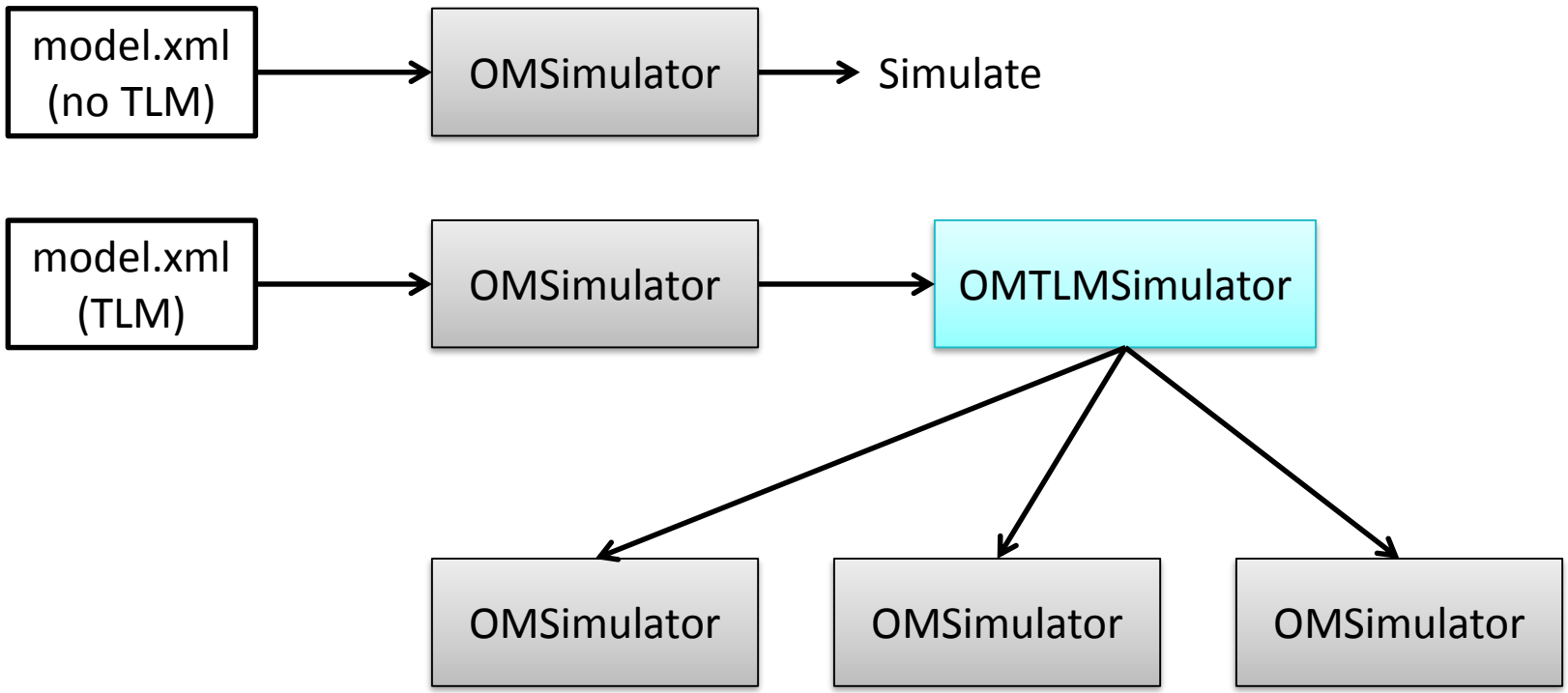
Non-delayed signal connections,
algebraic loops



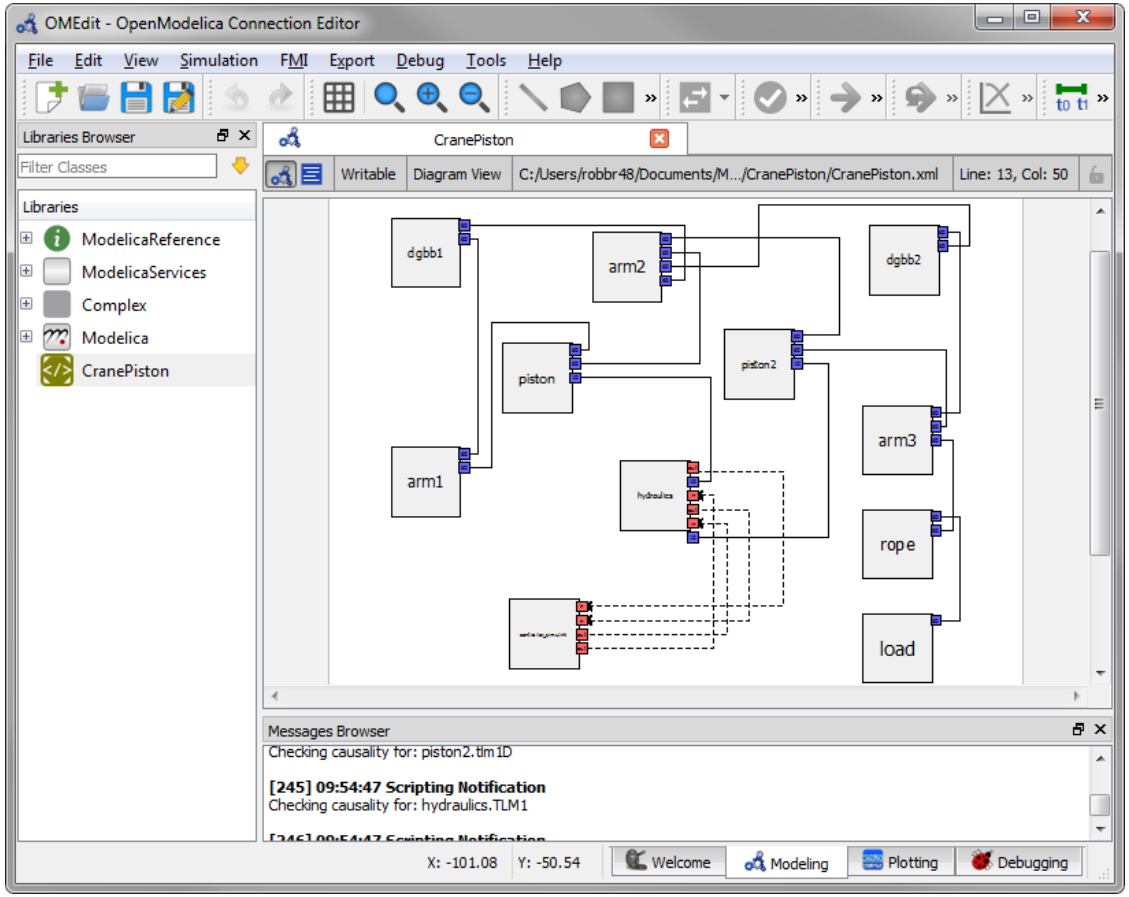
TLM connections,
delayed signal connections

Merging TLMSimulator and FMISimulator

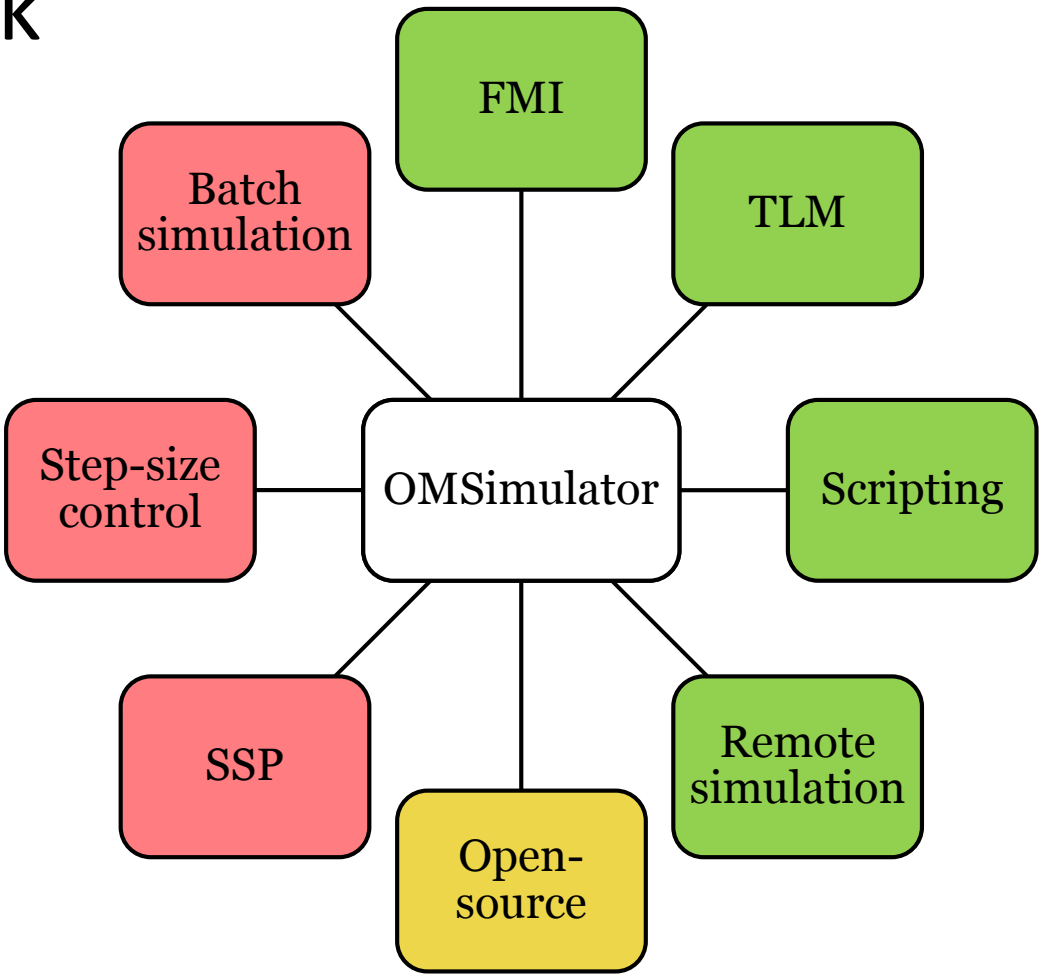



model.xml
(no TLM)
OMSimulator
Simulate
model.xml
(TLM)
OMSimulator
OMTLMSimulator
OMSimulator
OMSimulator
OMSimulator

Graphical connection editor in OMEdit



Outlook


FMI
Batch simulation
TLM
Step-size control
OMSimulator
Scripting
SSP
Open-source
Remote simulation

Robert Braun

www.liu.se

LINKÖPING UNIVERSITY