Functional Mock-up Interface

FMI Tutorial

MODPROD 2026

LINKÖPING UNIVERSITY

50 years of free thinking shaping the future

Motivation

- Tool-independent, free-to-use standard
  - Avoid lock-in effects
  - Preserve investments
  - Facilitate collaboration
- Containerized models
  - Decouple know-how between modellers and model users
  - Protect intellectual property via block-box models
- C API
  - Compiler independent models
  - Cross-platform support

FMI History

FMI 1.0 – 2011

- Co-simulation & Model Exchange

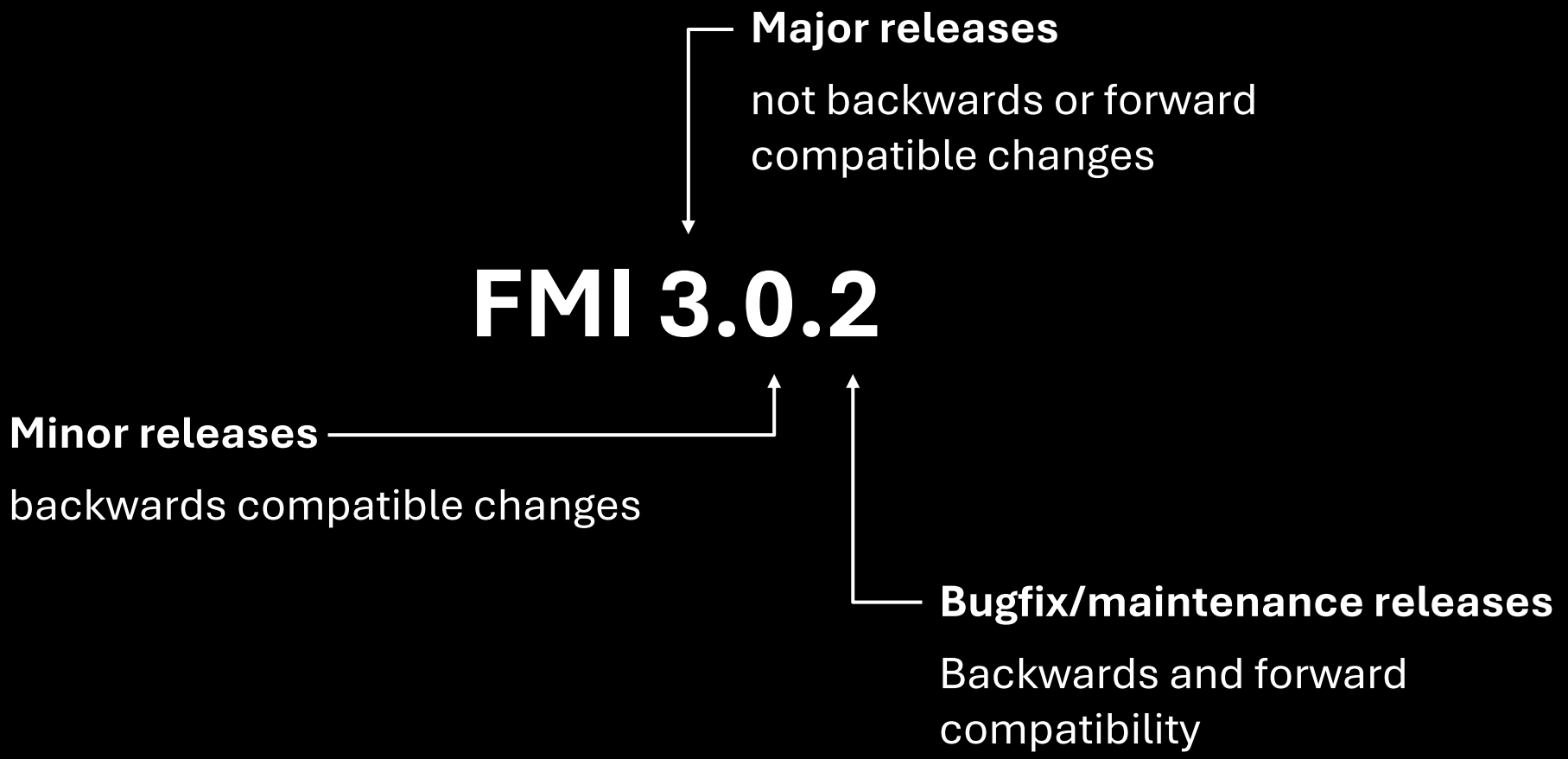
FMI 2.0 – 2013

- Harmonized co-simulation & model exchange

FMI 3.0 – 2021

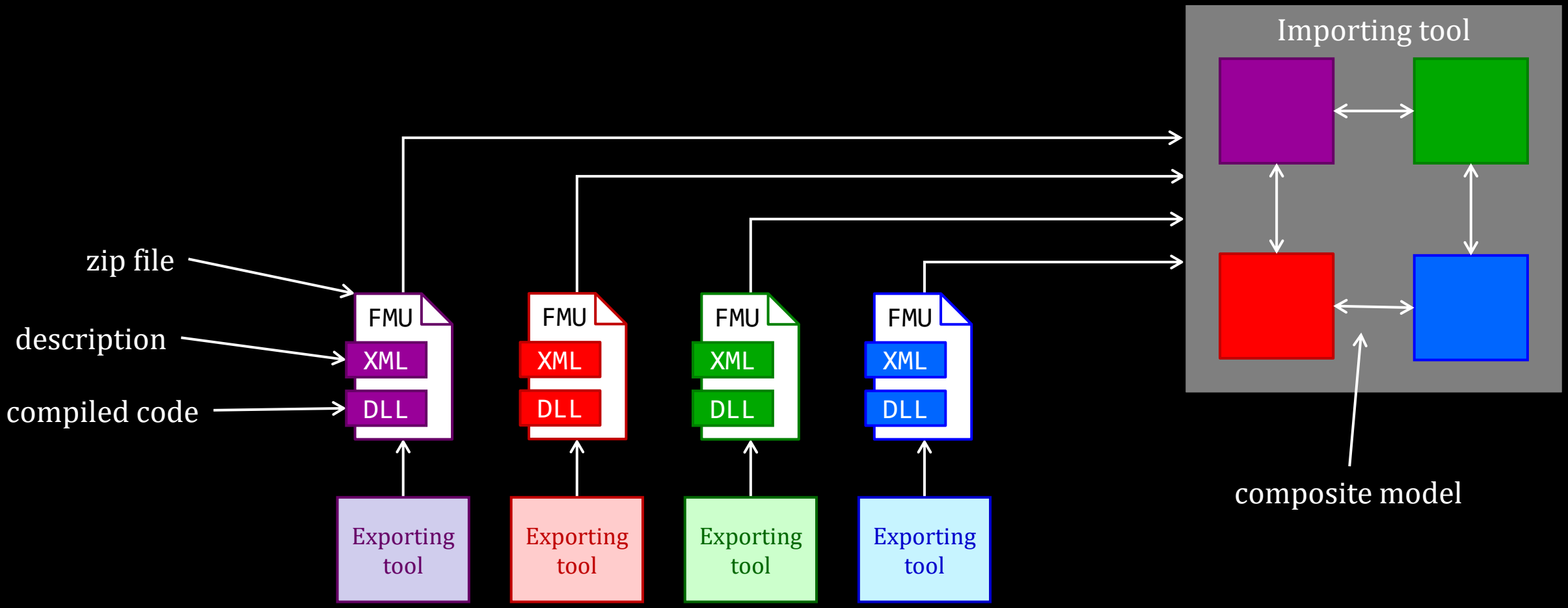
- Hybrid co-simulation
- Scheduled execution
- Support for more data types

FMI Versioning



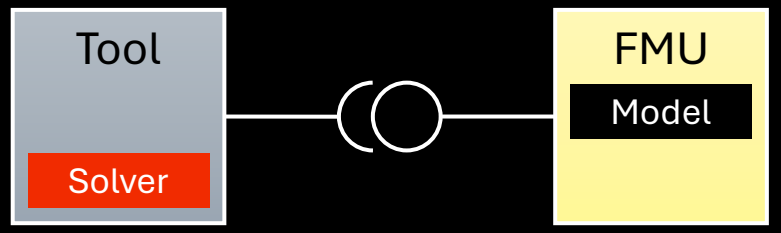
How does FMI work?

Functional Mockup Interface



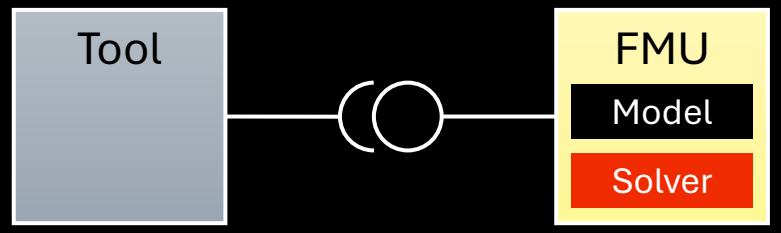
Functional Mockup Interface

Model Exchange



- FMU computes state derivatives
- States computed by importing tool

Co-Simulation



- Solver embedded in FMU
- States computed inside FMU

”Black-box”

FMI API Functions

```
fmi2Component fmi2Instantiate(name, type, id, ...)
fmi2Status fmi2SetupExperiment(tolerance, startTime, stopTime, ...)
fmi2Status fmi2EnterInitializationMode()
fmi2Status fmi2ExitInitializationMode()
fmi2Status fmi2SetReal(valueReferences, values)
fmi2Status fmi2GetReal(valueReferences, values)
fmi2Status fmi2DoStep(currentTime, stepSize, ...);
fmi2Status fmi2Terminate()
fmi2Status fmi2Reset()
fmi2FreeInstance()
```

FMI Model Description Example

```
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription variableNamingConvention="structured" generationDateAndTime="2021-11-23T09:08:37Z"
                     numberOfEventIndicators="0" guid="{21489e55-2aad-4ca4-9d91-fc5df4c443a2}"
                     description="" modelName="Position_Servo" generationTool="HopsanGenerator"
                     fmiVersion="2.0">
    <CoSimulation canHandleVariableCommunicationStepSize="true" modelIdentifier="Position_Servo"/>
    <ModelExchange modelIdentifier "Position_Servo"/>
    <ModelVariables>
        <ScalarVariable name="x" valueReference="1" description="position" initial="exact"
                causality="output" variability="continuous">
            <Real start="0"/>
        </ScalarVariable>
        <ScalarVariable name="xref" valueReference="2" description="reference position”
                        causality="input" variability="continuous">
            <Real start="0"/>
        </ScalarVariable>
        <ScalarVariable name="Kp" valueReference="3" description="Control gain"
                        causality="parameter" variability="fixed">
            <Real start="1"/>
        </ScalarVariable>
    </ModelVariables>
    <ModelStructure/>
</fmiModelDescription>
```

FMI for Co-Simulation - Example

```
fmu = fmi2Instantiate(      , ...)
fmi2SetupExperiment(fmu, ...);
fmi2EnterInitializationMode(fmu);
fmi2ExitInitializationMode(fmu);

while(time < stopTime) {
    fmi2SetReal(fmu, ...);

    fmi2DoStep(fmu, time, timeStep);

    fmi2GetReal(fmu);
    time = time + timeStep;
}

fmi2Terminate(m)
fmi2FreeInstance(fmu);
```

create instance of FMU

initialize FMU

set inputs

tell FMU to take step

get outputs

terminate and cleanup

Excluded:

- error checking

FMI for Model Exchange - Example

```
fmu = fmi2Instantiate(     , ...)
fmi2SetupExperiment(fmu, ...);
fmi2EnterInitializationMode(fmu);
fmi2ExitInitializationMode(fmu);
fmi2GetContinuousStates(fmu, states, nstates);

while(time < stopTime) {
    fmi2EnterContinuousTimeMode(fmu);
    fmi2GetDerivatives(fmu, stateDerivatives, nstates);

    time = time + timeStep;
    fmi2SetTime(fmu, time);

    fmi2SetReal(fmu, ...);

    for(int i=0; i<nstates; ++i) {
        states[i] = states[i] + timeStep*stateDerivatives[i];
    }
    fmi2SetContinuousStates(fmu, states, nstates);
    fmi2CompletedIntegratorStep(fmu, fmi2True);

    fmi2GetReal(fmu);
}

fmi2Terminate(m)
fmi2FreeInstance(fmu);
```

create instance of FMU

initialize FMU

ask FMU for initial states

ask FMU for state derivatives

advance time

set inputs

integrate states

set states to FMU

get outputs

terminate and cleanup

Excluded:

- event handling
- error checking

What kind of models does FMI support?

Main Scope of FMI:

- Hybrid models (ODEs with events)
- Requires numerical solver
- Both continuous and discrete variables
- Independent time variable

Also supported:

- Purely algebraic models
- Complex discrete behavior with clocks and model partitions (FMI 3.0)
- …

FMI Tool Support

		FMI 1.0	FMI 2.0	FMI 3.0
Co-simulation	Export	**72**	**129**	**47**
	Import	**88**	**180**	**72**
Model Exchange	Export	**33**	**48**	**25**
	Import	**20**	**31**	**21**
Sheduled Execution	Export			**2**
	Import			**5**

Total: 271 tools

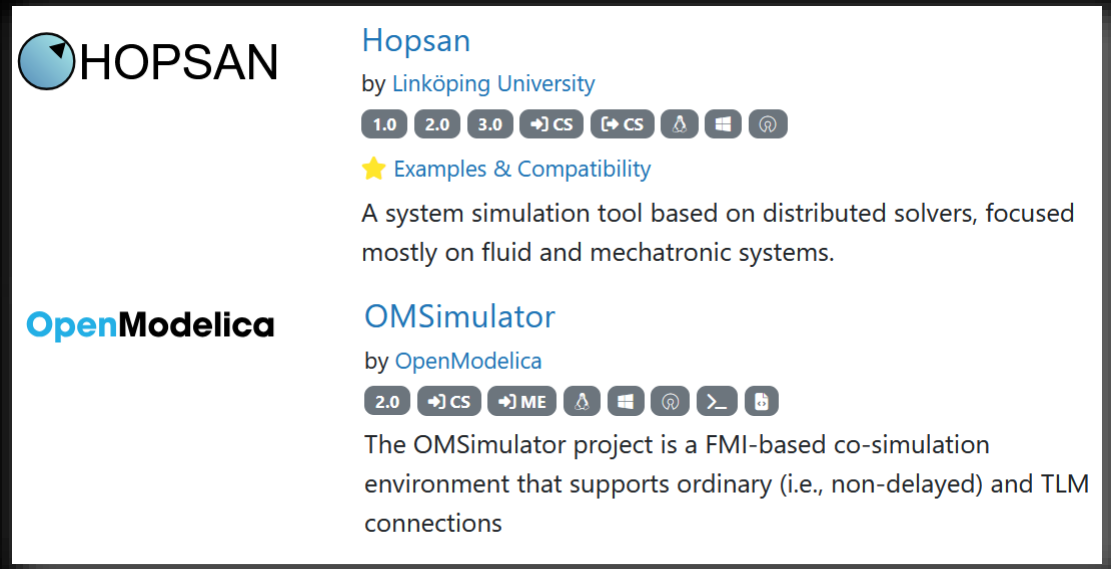
Compatibility Information

Tool vendors provide information

- Which tools have been tested?
- How were the tests performed?

Tools with compatibility information marked with a golden star

Replaces the previous cross-check system

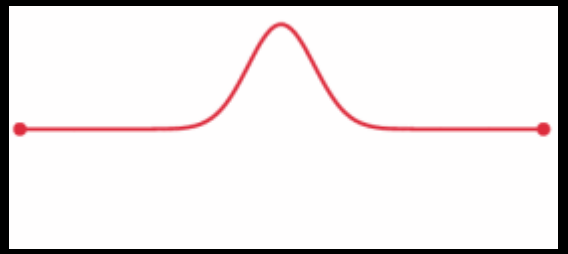


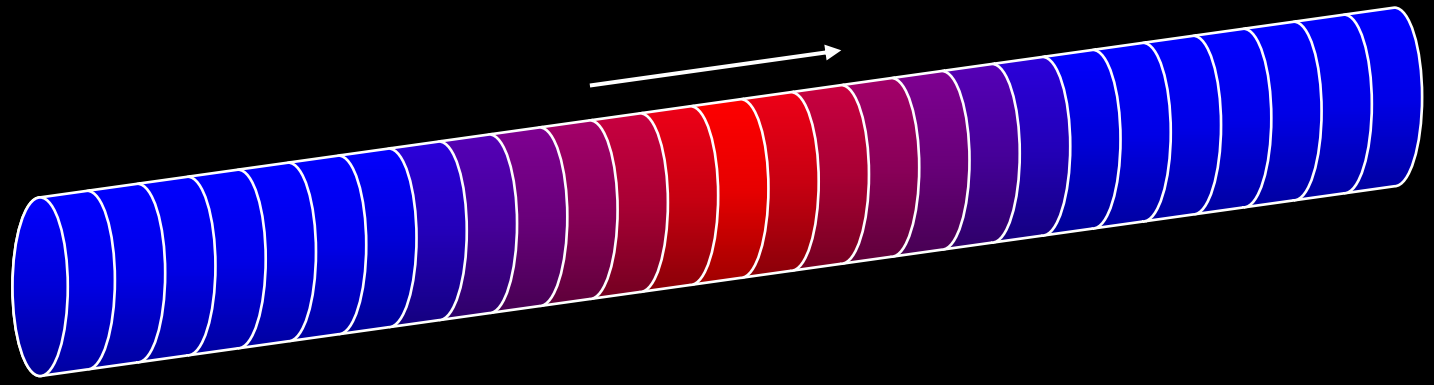
New Features in FMI 3.0

- Layered Standards
- Intermediate Update
- Hybrid Co-simulation
- Clocks
- Terminals and Icons
- Scheduled Execution
- New Data Types

FMI 1.0 & FMI 2.0	FMI 3.0
fmiReal	fmi3Float32 fmi3Float64
fmiInteger	fmi3Int8, fmi3UInt8 fmi3Int16, fmi3UInt16 fmi3Int32, fmi3UInt32 fmi3Int64, fmi3UInt64
fmiBoolean	fmi3Boolean
fmiString	fmi3String
	fmi3Binary
	fmi3Clock

Transmission Line Modelling





Transmission Line Modelling - Background

Invented twice by two independent research groups:

University of Berkeley, 1968

University of Nottingham, 1980

Distributed System Simulation With Bilateral Delay-Line Models

D. M. AUSLANDER

Assistant Professor of Mechanical Engineering, University of California, Berkeley, Calif.

Introduction

Journal of Basic Engineering

Use of the transmission-line modelling (t.l.m.) method to solve non-linear lumped networks

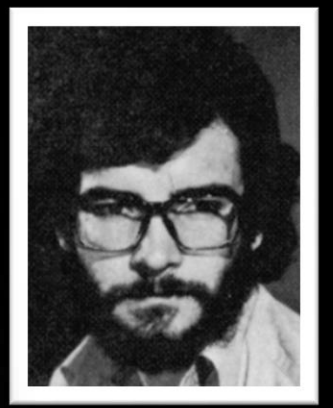
PETER B. JOHNS, M.Sc., Ph.D., C.Eng., M.I.E.E.*

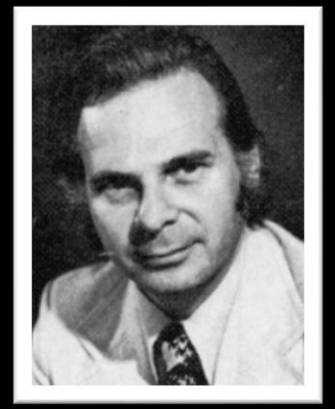
and

MARK O'BRIEN, B.Sc.†

SUMMARY

1 Introduction





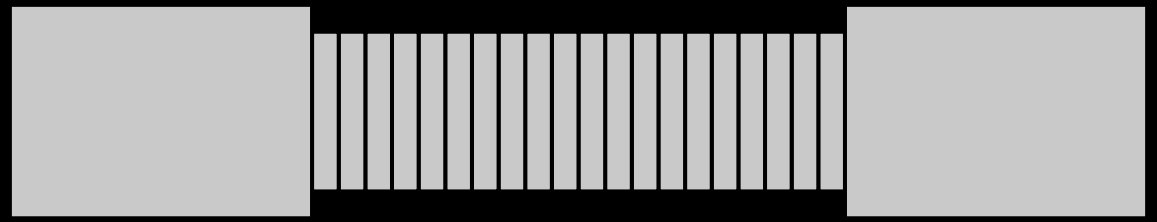
Transmission Line Modelling - Background

Every physical element has a natural time delay.

Transmission Line Modelling - Background

Every physical element has a natural time delay.

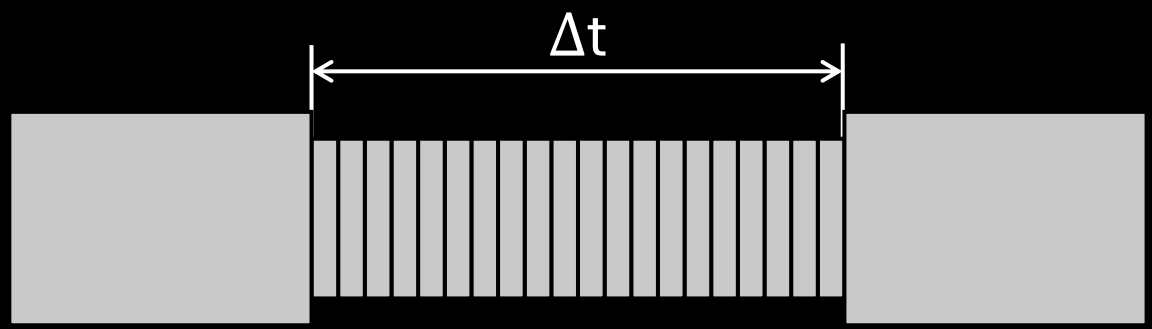
Mechanics:



Transmission Line Modelling - Background

Every physical element has a natural time delay.

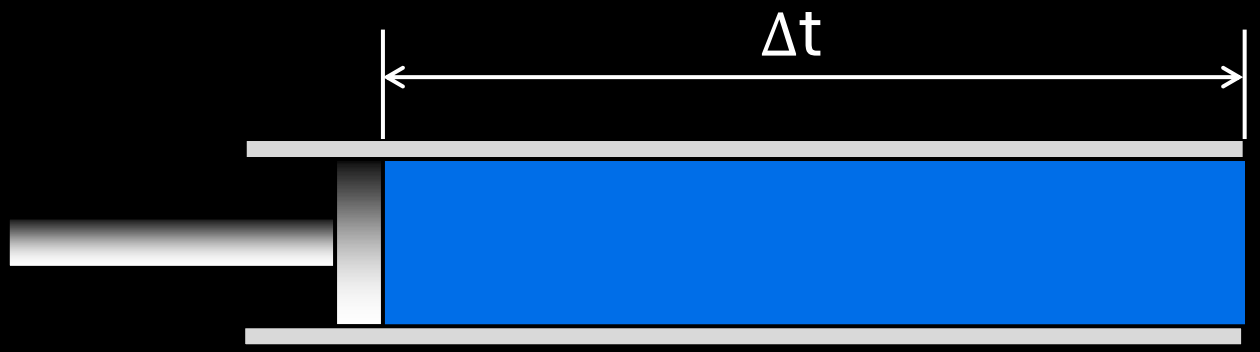
Mechanics:



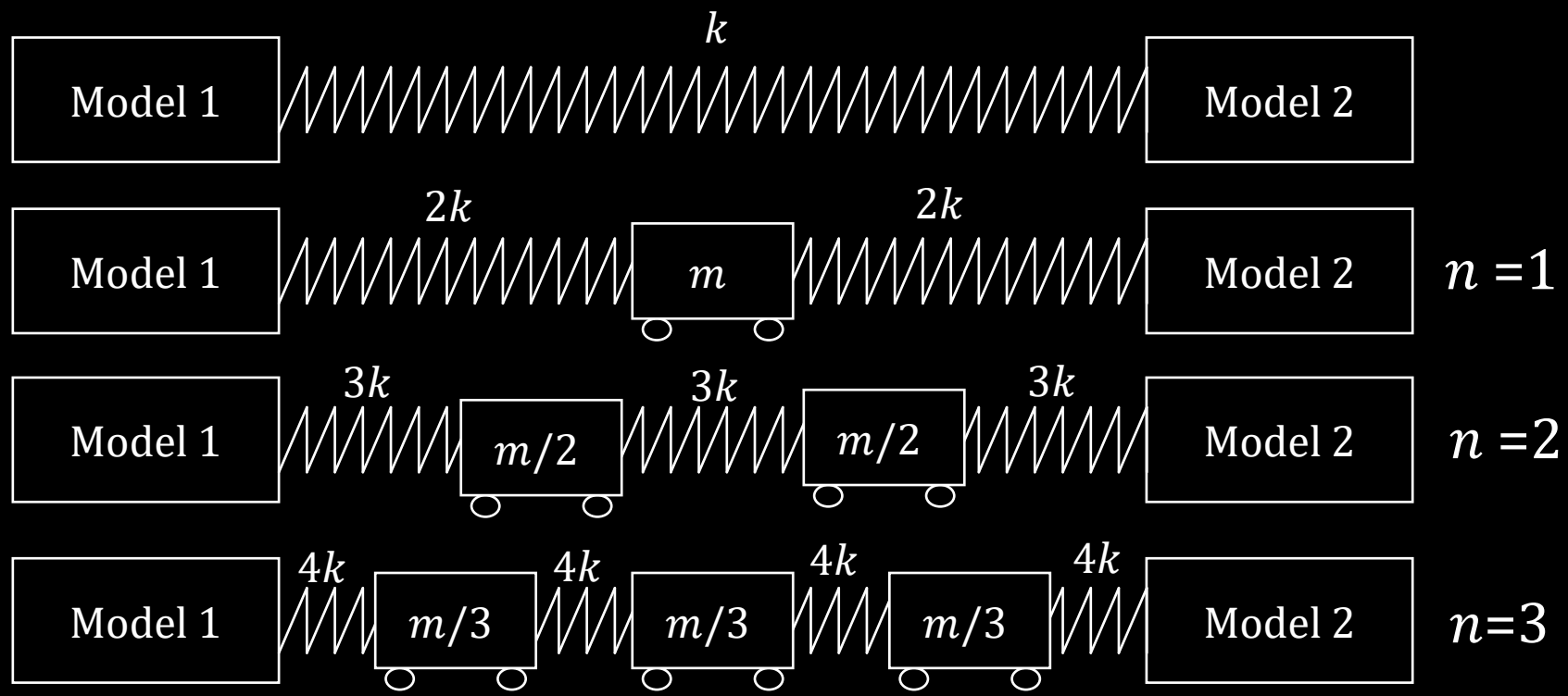
Transmission Line Modelling - Background

Every physical element has a natural time delay.

Hydraulics:

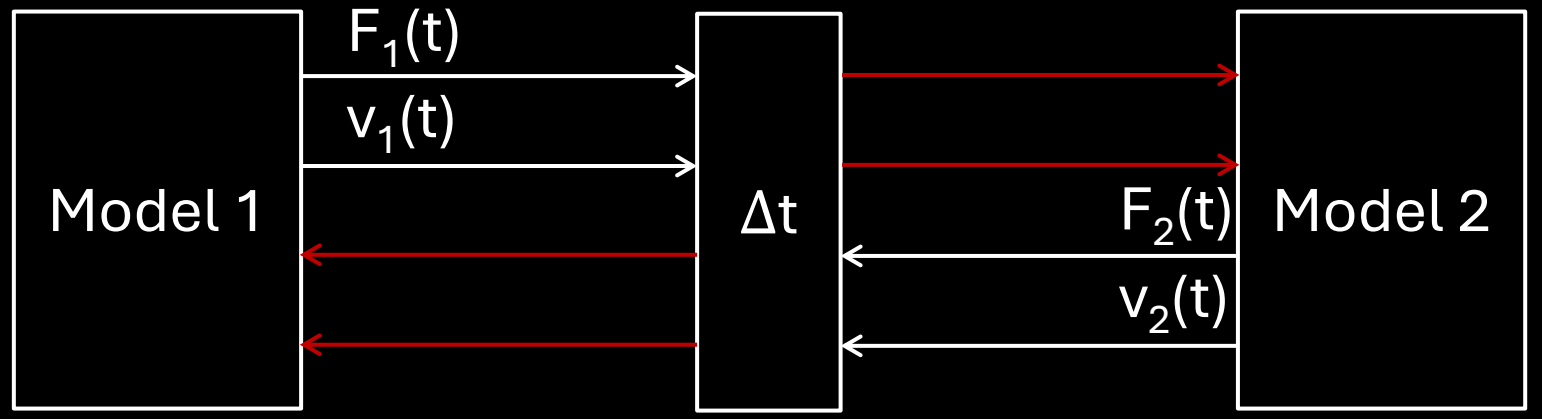


Transmission Line Modelling - Derivation



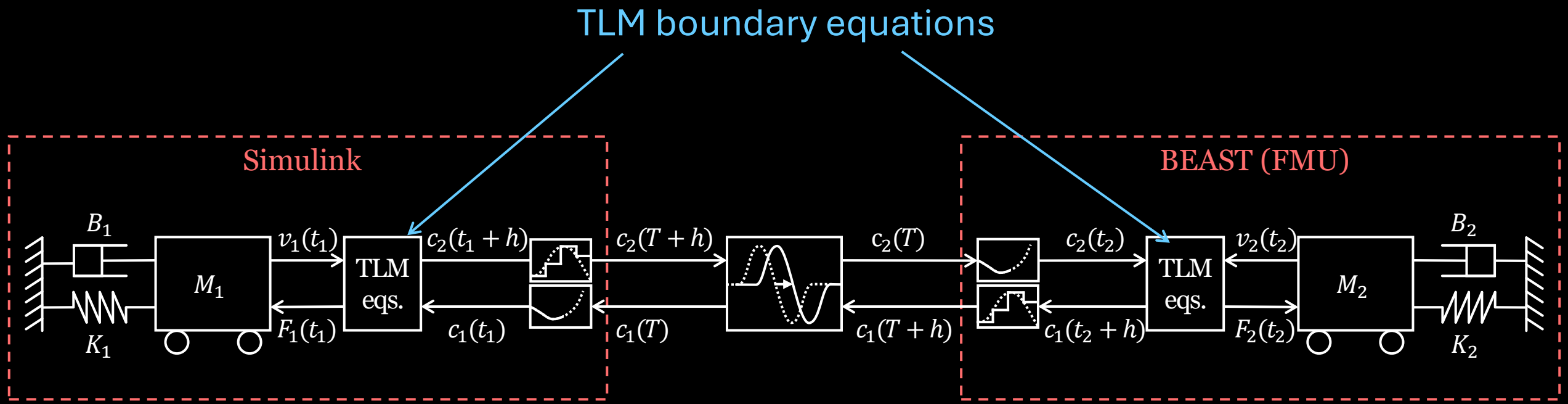
$n \to \infty$

TLM Equations



$$F_1(t) = F_2(t - \Delta t) + Z_c\left(v_1(t) + v_2(\mathrm{t} - \Delta t)\right)$$
$$F_2(t) = F_1(t - \Delta t) + Z_c\left(v_2(t) + v_1(t - \Delta t)\right)$$

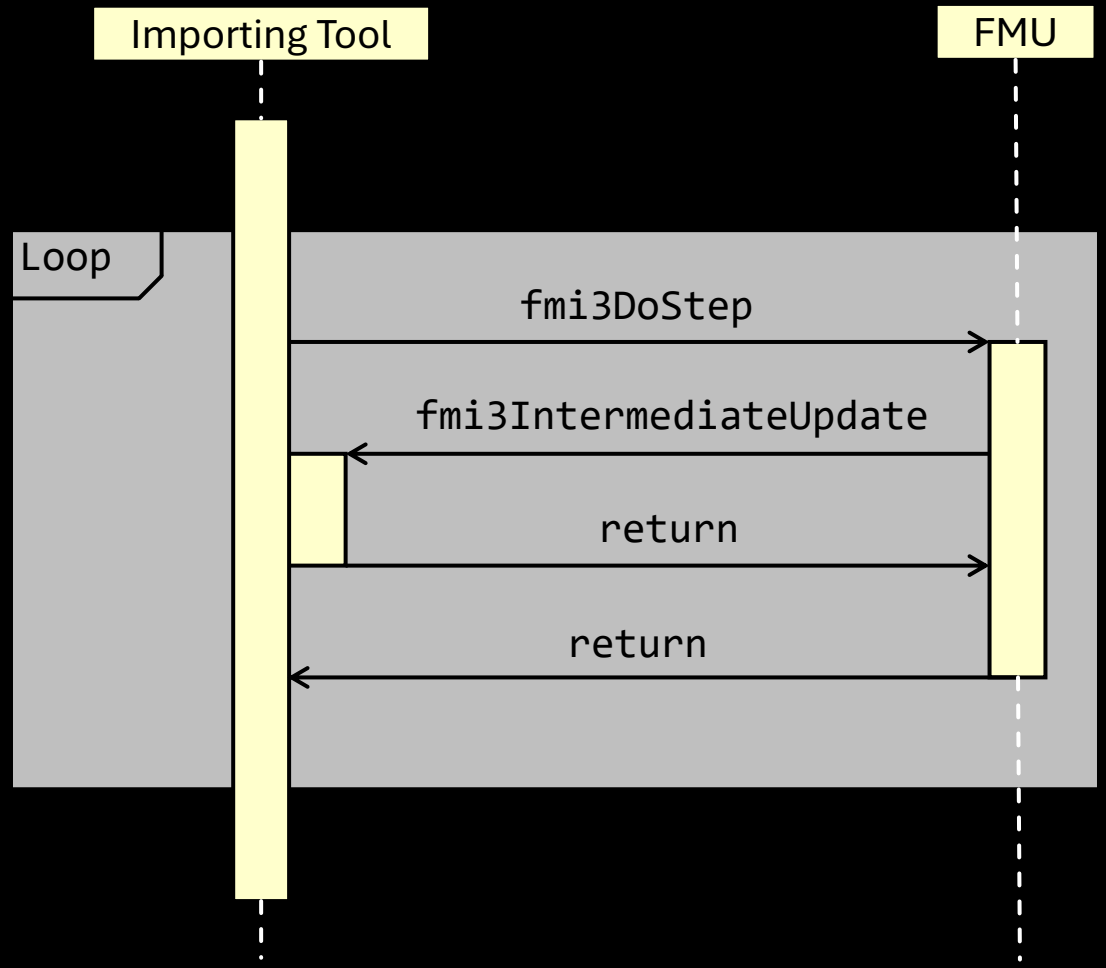
TLM with FMI 1.0 and 2.0



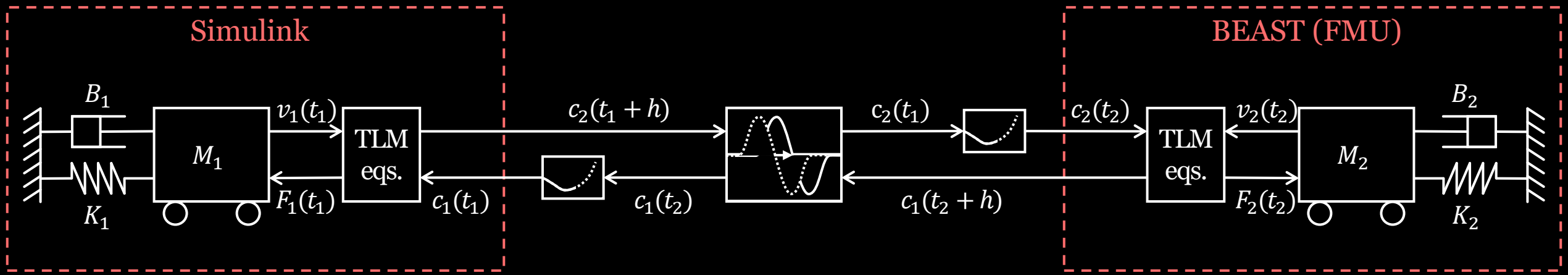
TLM with FMI 3.0

- Change proposal submitted during ITEA3 OpenCPS project
- FMUs can request and provide intermeriate variables
- Released in 2022

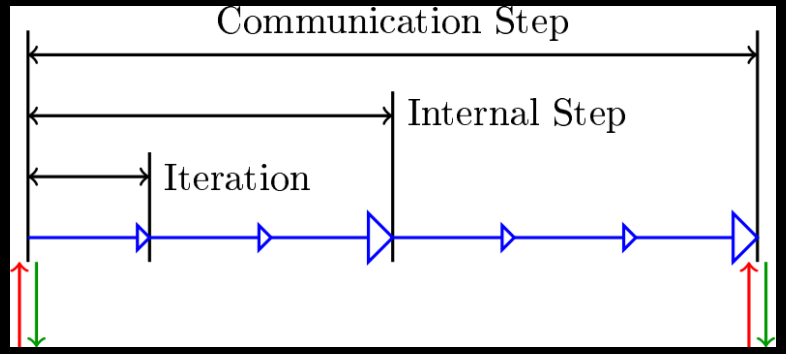
openCPS



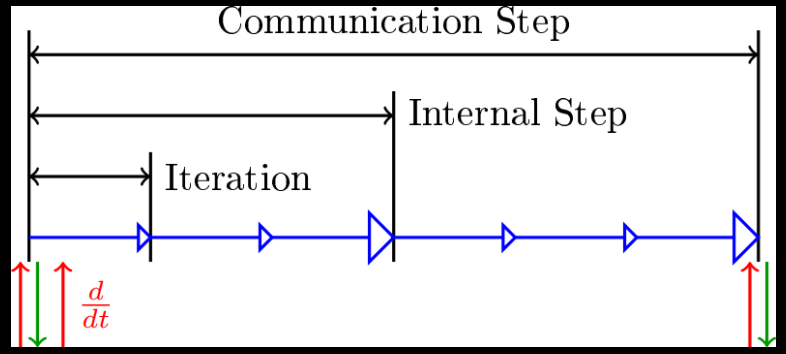
TLM with FMI 3.0



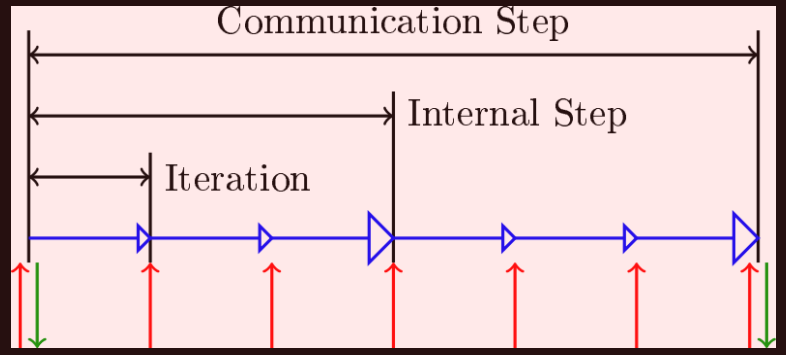
Communication Patterns



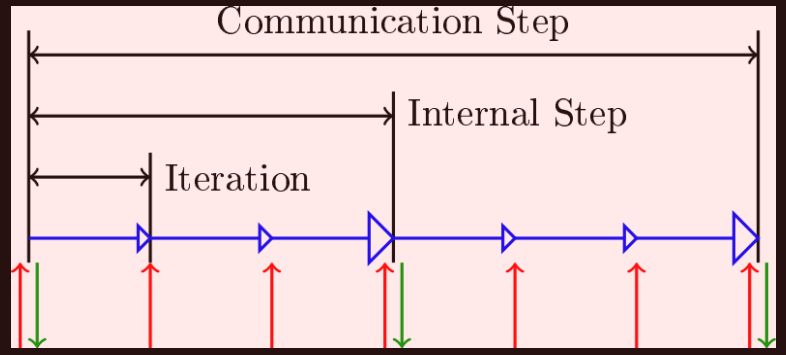
Zero-order hold



Input derivatives



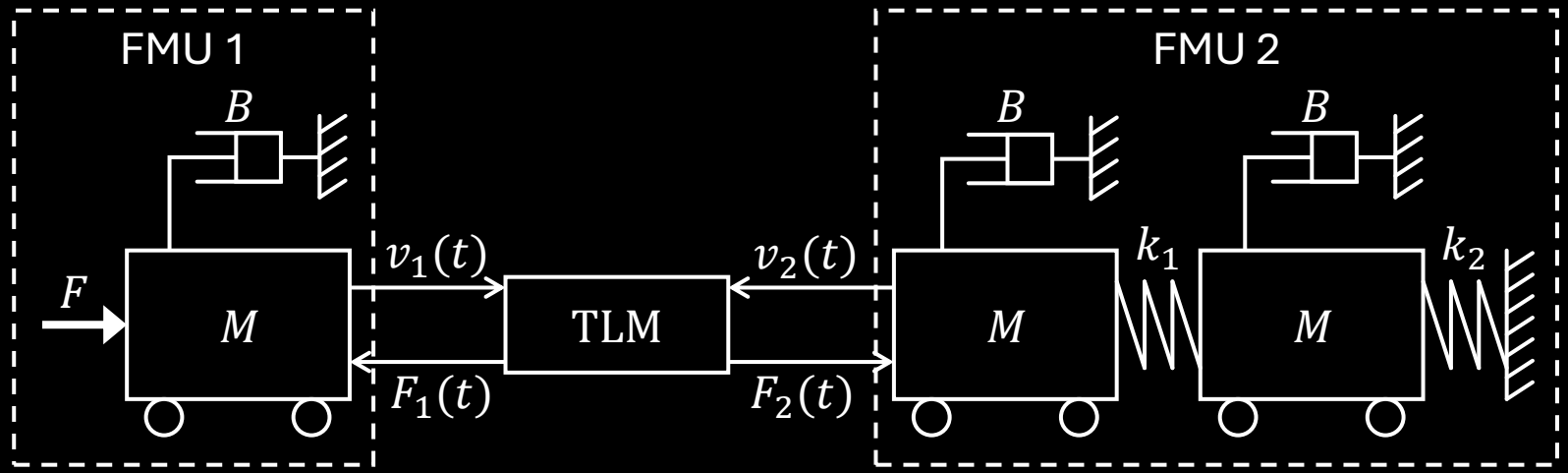
Intermediate inputs



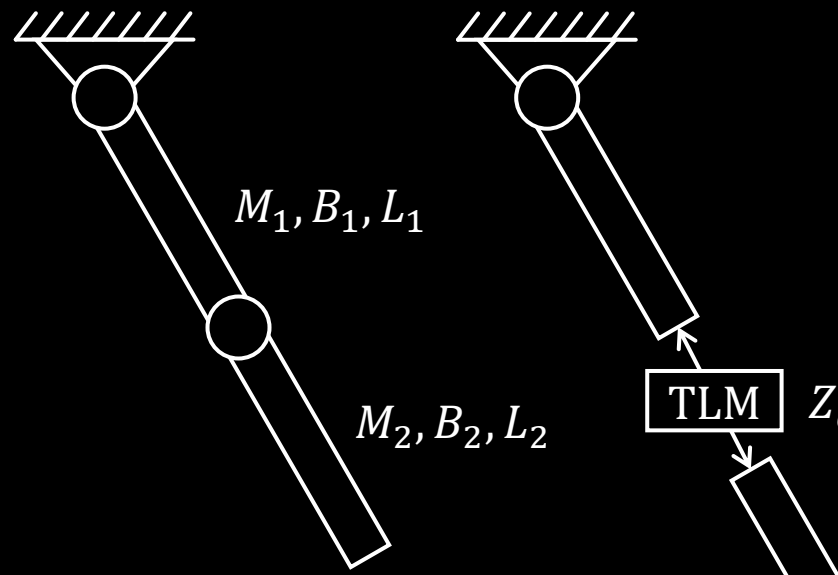
Interemediate inputs/outputs

FMI3

1D Test Model

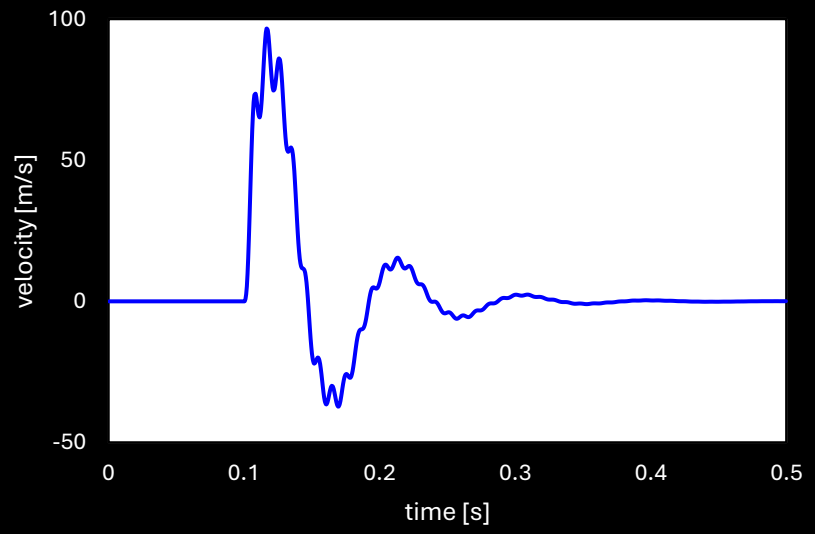


2D Test Model

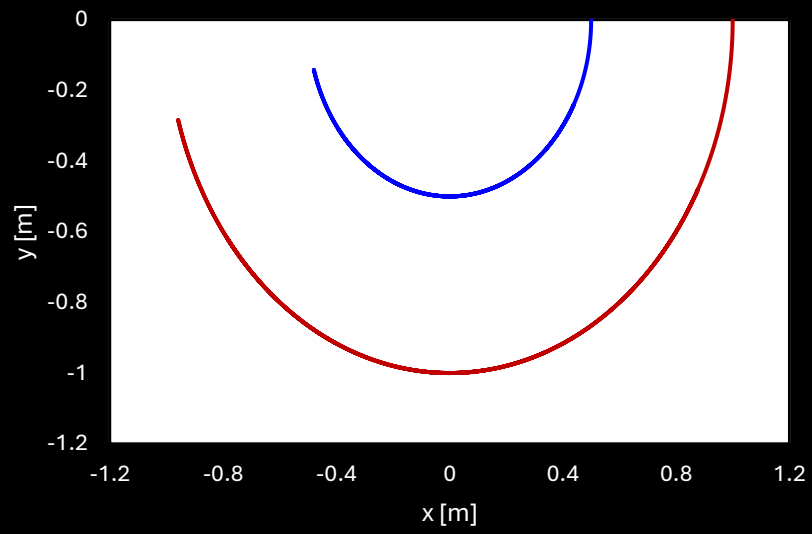


Experimental Results

1D spring-mass-damper model



2D pendulum model



Both models stable with large communcation steps!