TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Architektur, Institut für Bauklimatik, Professur für Bauphysik

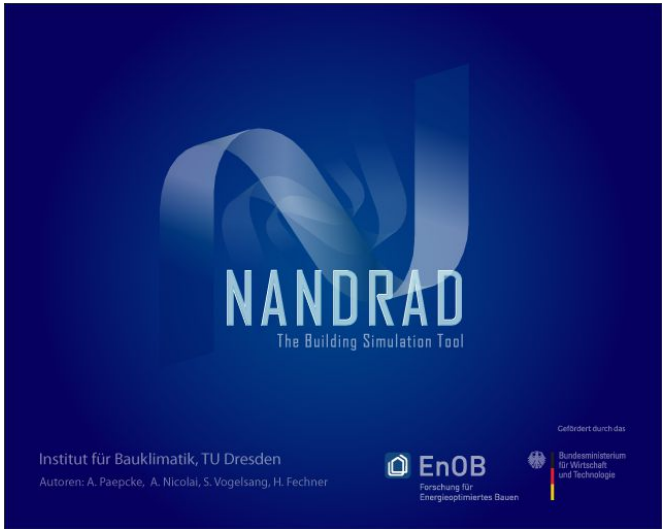
Co-Simulation between detailed building energy performance simulation and Modelica HVAC component models

Andreas Nicolai and Anne Paepcke

andreas.nicolai@tu-dresden.de

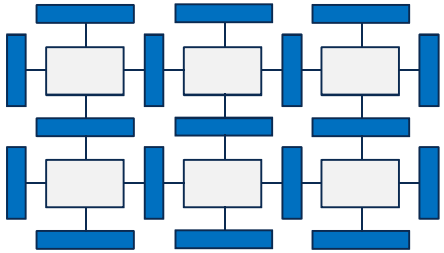
Dresden, 16.09.2016

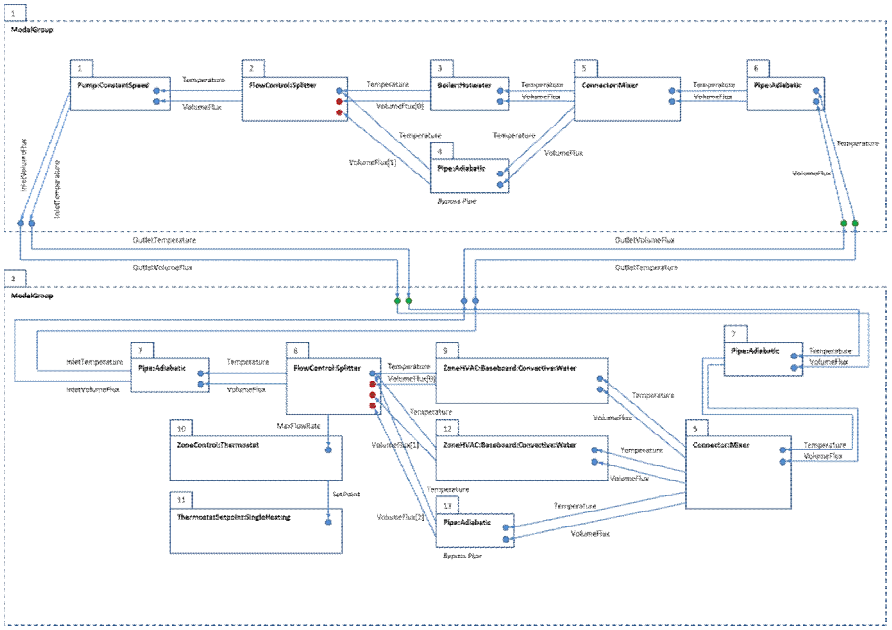
Starting point: NANDRAD - Building energy simulation engine



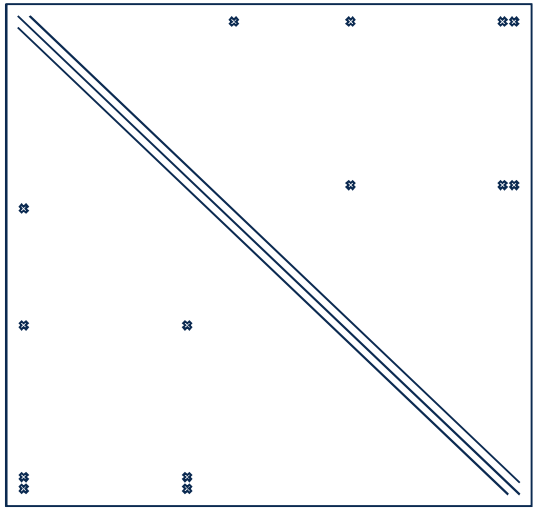
- Multi-zone building model
- Realistic building physics (**spatially resolved walls**/storage members)
- CVODE time integration engine (**error controlled** variable-order, **variable step** method)
- Algebraic relations embedded in ODE equations
- Using **Newton-Krylov**-method for non-linear system of equations
- Special **problem-specific** sparse matrix structures and preconditioners

Multi-zone model, embedded equipment loop and sparsity pattern





Instead of hard-coded equipment models rather use Modelica-based equipment models



Jacobian Matrix Sparsity Pattern

TECHNISCHE UNIVERSITÄT DRESDEN

Idea tested: get everything into Modelica via Code-generator

```
model Wall3 "A 1D Wall Model generated from NANDRAD"
  type HeatConductionCoefficient = Real(unit = "W/(m.K)", min = 0) "Effective heat cunduction coefficient";
  type ThermalConductivity = Real(unit = "W/(m2.K)", min = 0) "Thermal conductivity";
  parameter Integer nLayer = 2 "Number of material layers";
  parameter Modelica.SIunits.Temperature T0 = 293.15 "Initial temperature";
  parameter Modelica.SIunits.Density rho[nLayer] = {2100.0, 1100} "Material density";
  parameter Modelica.SIunits.SpecificHeatCapacity cp[nLayer] = {780.0, 820} "Material specific heat capacity";
  parameter ThermalConductivity lambda[nLayer] = {1.8, 0.04} "Material thermal conductivity";
  parameter Integer n = 3 "Number of discretization elements";
  parameter Modelica.SIunits.Length dx[n] = {0.1, 0.2, 0.08} "Element discretization";
  parameter Modelica.SIunits.Temperature T_left = 273.15 "Left temperature";
  parameter Modelica.SIunits.Temperature T_right = 293.15 "Right temperature";
  parameter HeatConductionCoefficient alpha_left = 8 "Left heat conduction coefficient";
  parameter HeatConductionCoefficient alpha_right = 25 "Right heat conduction coefficient";
  Modelica.SIunits.EnergyDensity u[n] "Energy density of the wall";
  Modelica.SIunits.Temperature T[n] "Wall temperature";
  Modelica.SIunits.HeatFlux q[n - 1] "Internal heat conduction flux";
  Modelica.SIunits.HeatFlux q_left "Left side heat conduction flux";
  Modelica.SIunits.HeatFlux q_right "Right side heat conduction flux";
initial equation
  // discretization elements 1 and 2 have material #1, element 3 uses material #2
  u[1] = rho[1] * cp[1] * T0;
  u[2] = rho[1] * cp[1] * T0;
  u[3] = rho[2] * cp[2] * T0;
equation
  u[1] = rho[1] * cp[1] * T[1] "Energy storage relation, element 1";
  u[2] = rho[1] * cp[1] * T[2] "Energy storage relation, element 2";
  u[3] = rho[2] * cp[2] * T[3] "Energy storage relation, element 3";
  q_left = alpha_left * (T_left - T[1]) "Left heat conduction boundary condition";
  q_right = alpha_right * (T[3] - T_right) "Right heat conduction boundary condition";
  q[1] = (dx[1] + dx[2]) / (dx[1] / lambda[1] + dx[2] / lambda[1]) * (T[1] - T[2]);
  q[2] = (dx[2] + dx[3]) / (dx[2] / lambda[1] + dx[3] / lambda[2]) * (T[2] - T[3]);
  der(u[1]) = (q_left - q[1]) / dx[1] "Energy conservation equation, element 1";
  der(u[2]) = (q[1] - q[2]) / dx[2] "Energy conservation equation, element 2";
  der(u[3]) = (q[2] - q_right) / dx[3] "Energy conservation equation, element 3";
end Wall3;
```

A lot of code like this!

Symbolic analysis (OpenModelica/ SimulationX) **really slow** on large Models

TECHNISCHE UNIVERSITÄT DRESDEN

Idea tested: get everything into Modelica via Code-generator

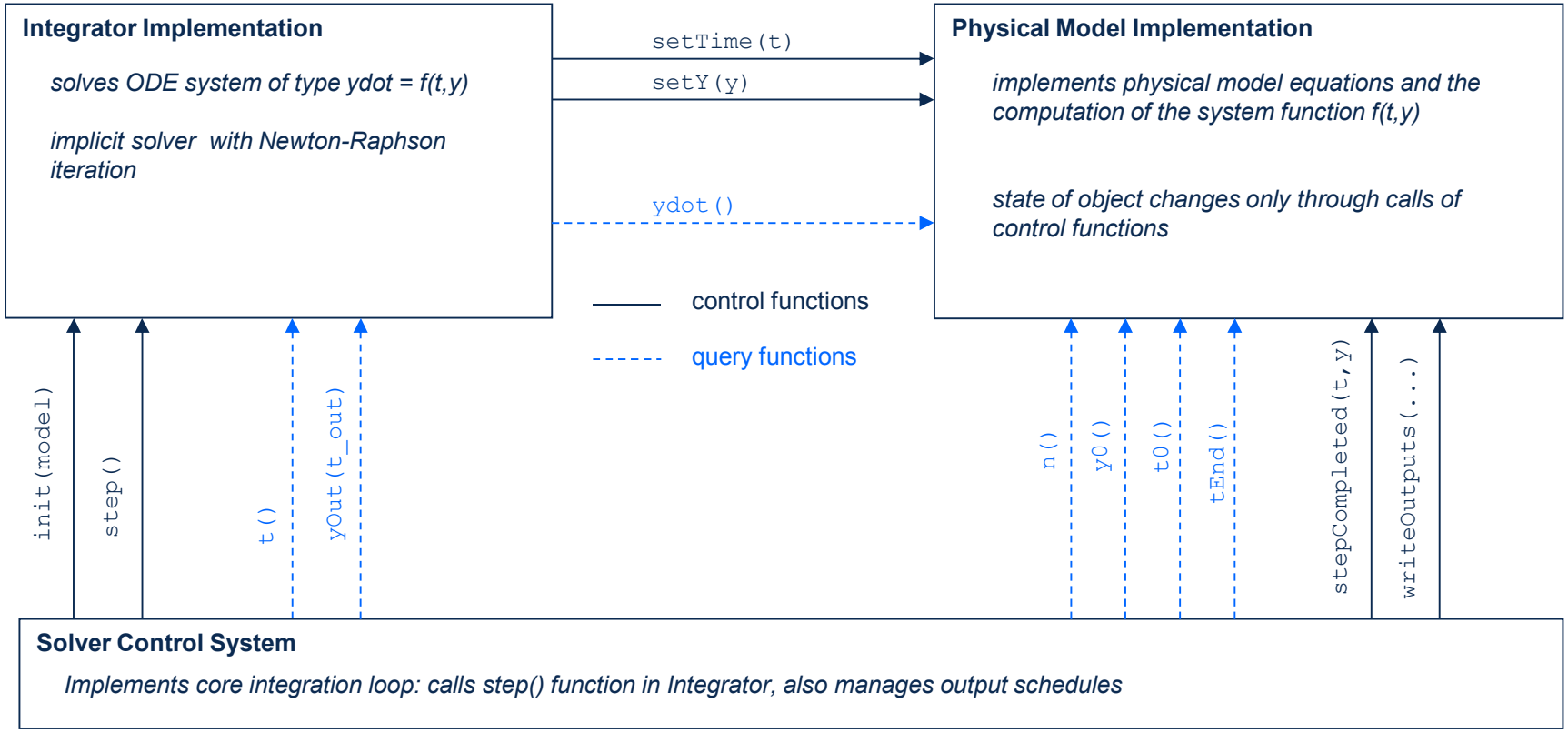
```
model Wall3 "A 1D Wall Model generated from NANDRAD"
  type HeatConductionCoefficient = Real(unit = "W/(m.K)", min = 0) "Effective heat cunduction coefficient";
  type ThermalConductivity = Real(unit = "W/(m2.K)", min = 0) "Thermal conductivity";
  parameter Integer nLayer = 2 "Number of material layers";
  parameter Modelica.SIunits.Temperature T0 = 293.15 "Initial temperature";
  parameter Modelica.SIunits.Density rho[nLayer] = {2100.0, 1100} "Material density";
  parameter Modelica.SIunits.SpecificHeatCapacity cp[nLayer] = {780.0, 820} "Material specific heat capacity";
  parameter ThermalConductivity lambda[nLayer] = {1.8, 0.04} "Material thermal conductivity";
  parameter Integer n = 3 "Number of discretization elements";
  parameter Modelica.SIunits.Length dx[n] = {0.1, 0.2, 0.08} "Element discretization";
  parameter Modelica.SIunits.
  parameter Modelica.SIunits.
  parameter HeatConductionCoe
  parameter HeatConductionCoe
  Modelica.SIunits.EnergyDens
  Modelica.SIunits.Temperatur
  Modelica.SIunits.HeatFlux q
  Modelica.SIunits.HeatFlux q
  Modelica.SIunits.HeatFlux q
initial equation
  // discretization elements 1 and 2 have material #1, element 3 uses material #2
  u[1] = rho[1] * cp[1] * T0;
  u[2] = rho[1] * cp[1] * T0;
  u[3] = rho[2] * cp[2] * T0;
equation
  u[1] = rho[1] * cp[1] * T[1] "Energy storage relation, element 1";
  u[2] = rho[1] * cp[1] * T[2] "Energy storage relation, element 2";
  u[3] = rho[2] * cp[2] * T[3] "Energy storage relation, element 3";
  q_left = alpha_left * (T_left - T[1]) "Left heat conduction boundary condition";
  q_right = alpha_right * (T[3] - T_right) "Right heat conduction boundary condition";
  q[1] = (dx[1] + dx[2]) / (dx[1] / lambda[1] + dx[2] / lambda[1]) * (T[1] - T[2]);
  q[2] = (dx[2] + dx[3]) / (dx[2] / lambda[1] + dx[3] / lambda[2]) * (T[2] - T[3]);
  der(u[1]) = (q_left - q[1]) / dx[1] "Energy conservation equation, element 1";
  der(u[2]) = (q[1] - q[2]) / dx[2] "Energy conservation equation, element 2";
  der(u[3]) = (q[2] - q_right) / dx[3] "Energy conservation equation, element 3";
end Wall3;
```

Alternative:
Export building as Functional Mockup Unit (FMU) and connect to Modelica equipment model

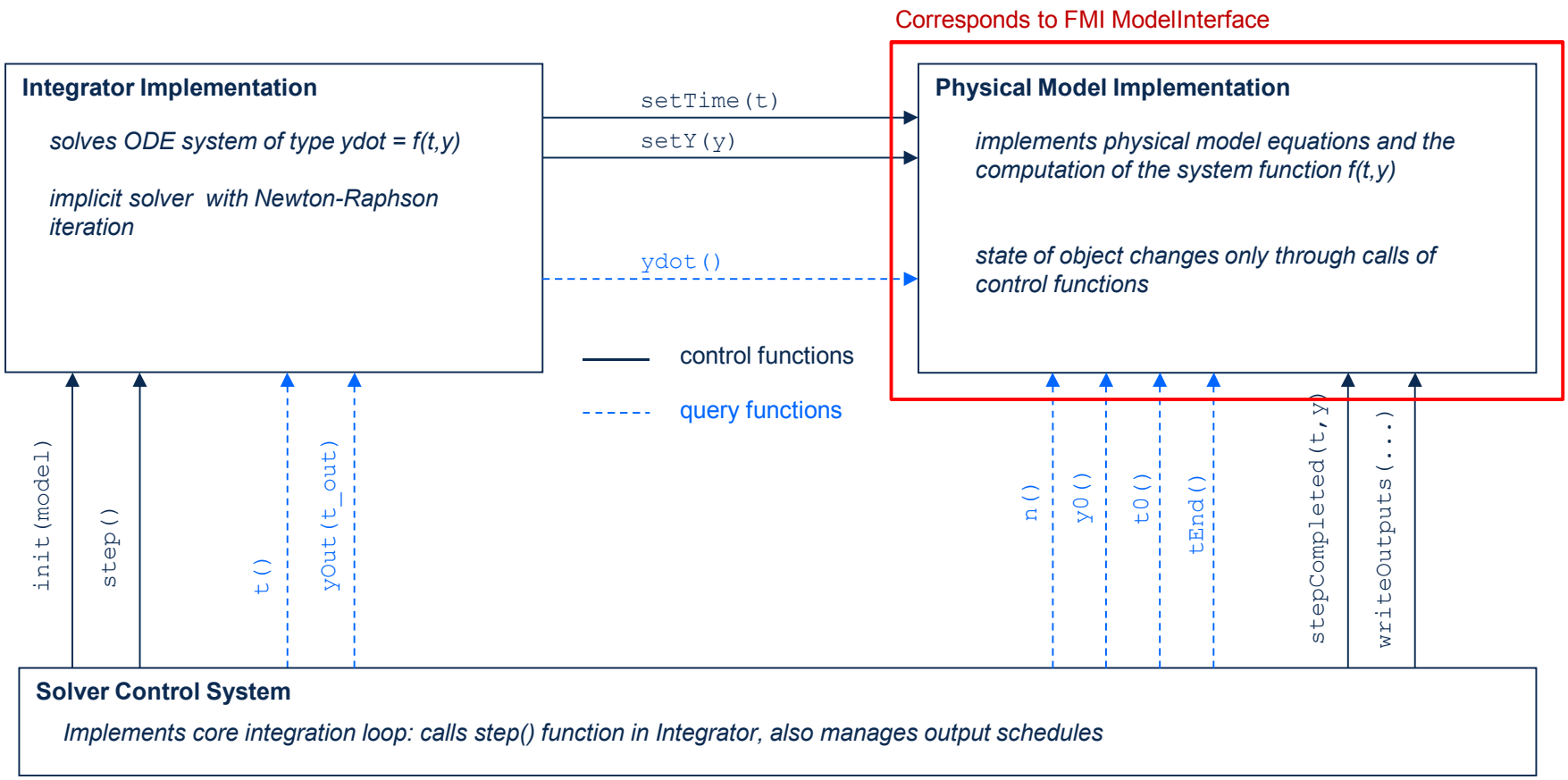
A lot of code like this!

Symbolic analysis (OpenModelica/ SimulationX) **really slow** on large Models

Software architecture of NANDRAD engine

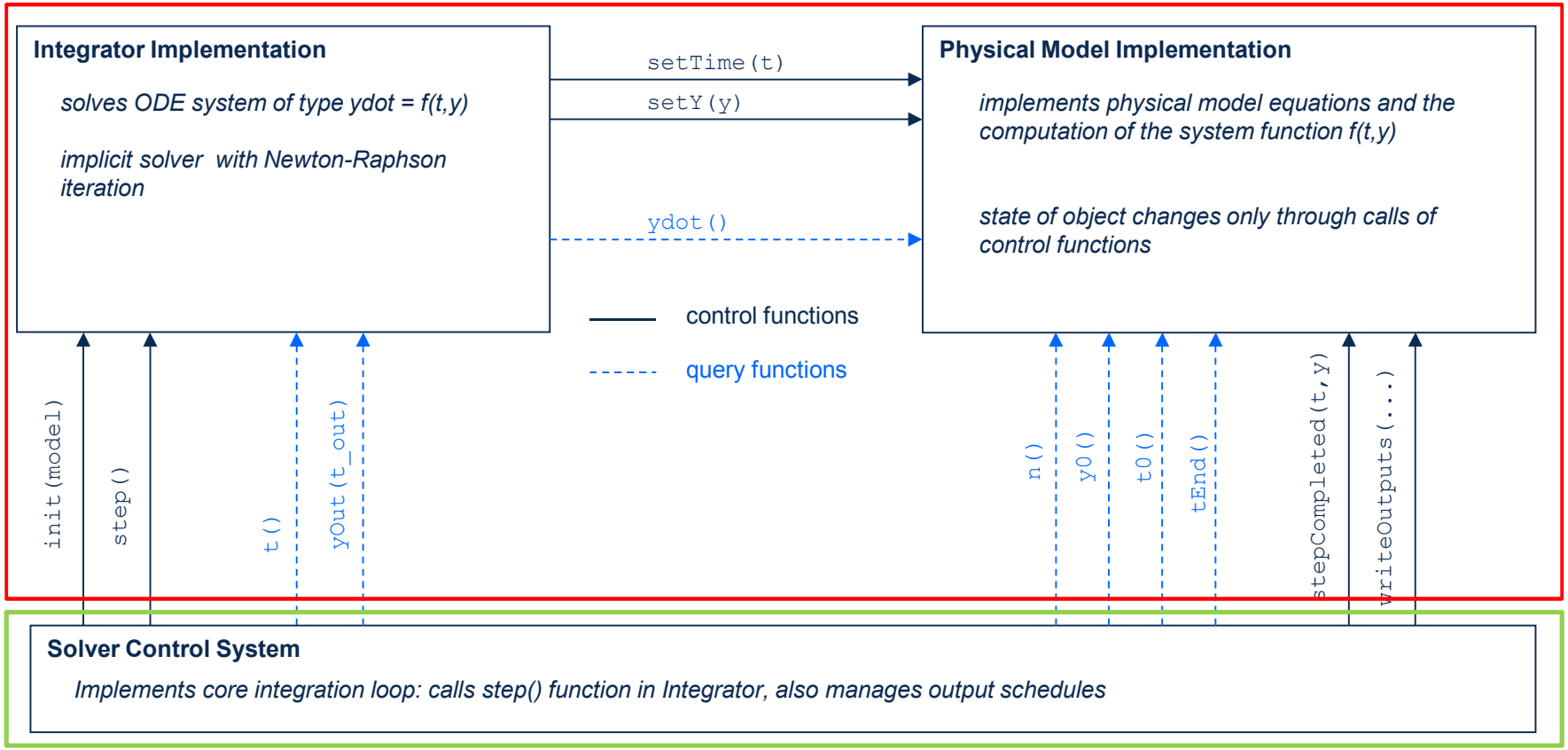


Export ODE system function via FMU for ModelExchange



Export physical model and integrator as FMI for Co-Simulation

Corresponds to FMI Co-Simulation interface



To be replaced by Co-Simulation Master

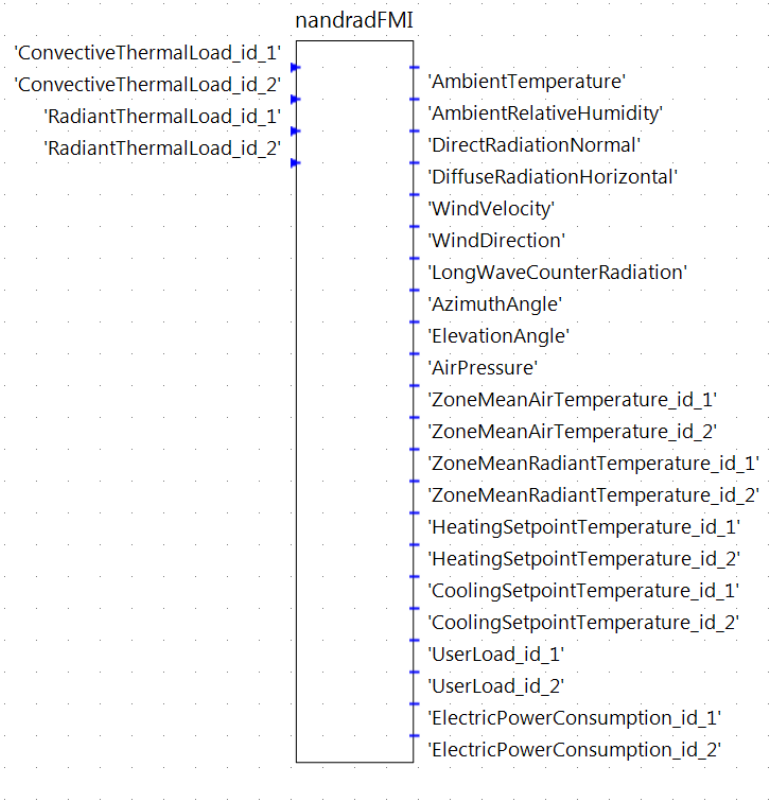
FMU Generation Process

1. Starting with BIM → Project file for standalone building simulation
2. Indicate which thermal zones shall be heated/cooled and be used in the FMU-interface
3. Run design day simulations (heating/cooling)
4. Generate report file (zone ID – variable index mapping, design loads)
5. Create ModelDescription.xml
6. Create Modelica-Wrapper and Adapter models (corresponding to project)
7. Compress project file, all database files, pre-compiled FMU-DLL and additional dependent libraries into FMU-archive

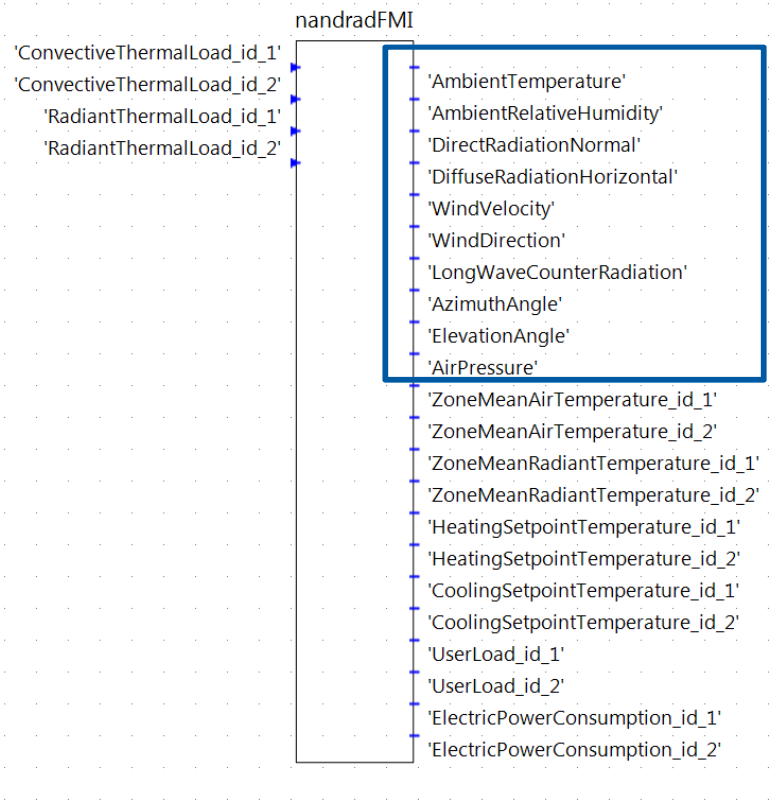
Steps 3 to 7 are done automatically by NANDRAD solver via command line:

```
NandradSolver  RowHouse_var1.nandrad  --fmu-export=RowHouse_var1.fmu
```

*(exported FMU is exported as FMI version 2, for **both ME and CS**)*



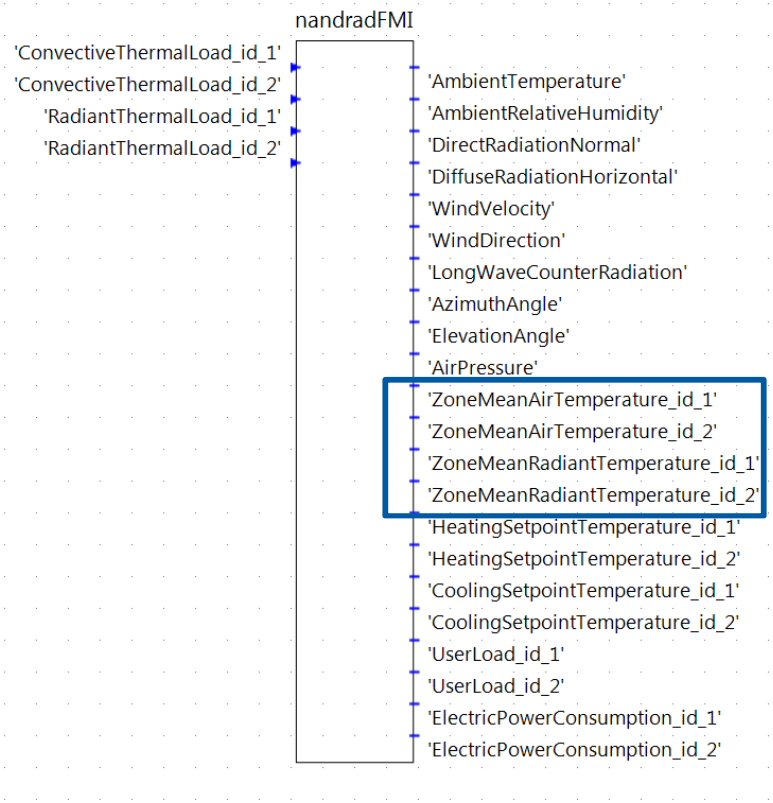
FMI Interface to NANDRAD Building Model FMU



Outputs

- Climate data

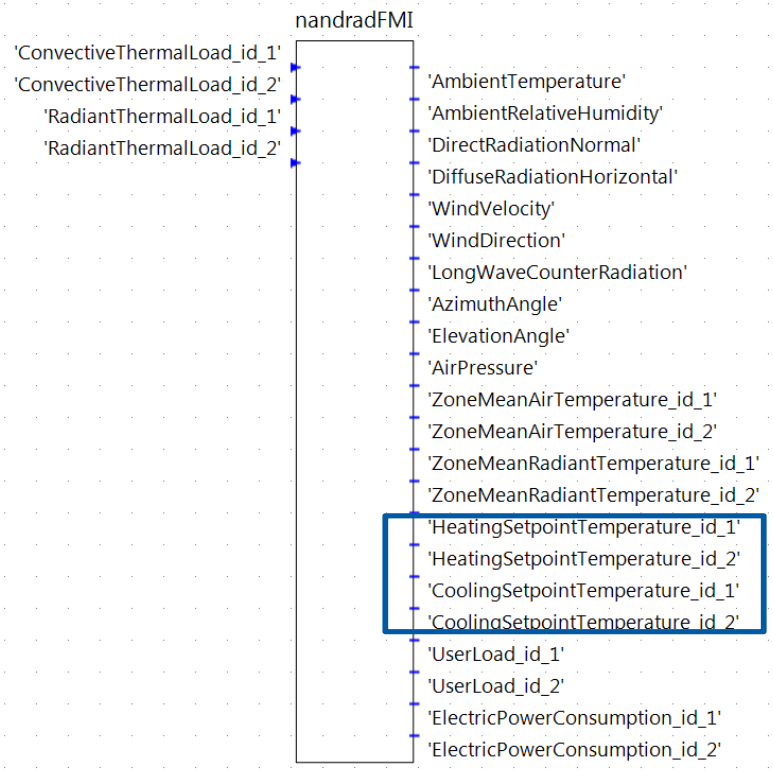
FMI Interface to NANDRAD Building Model FMU



Outputs

- Climate data
- Zone air/ radiant temperatures

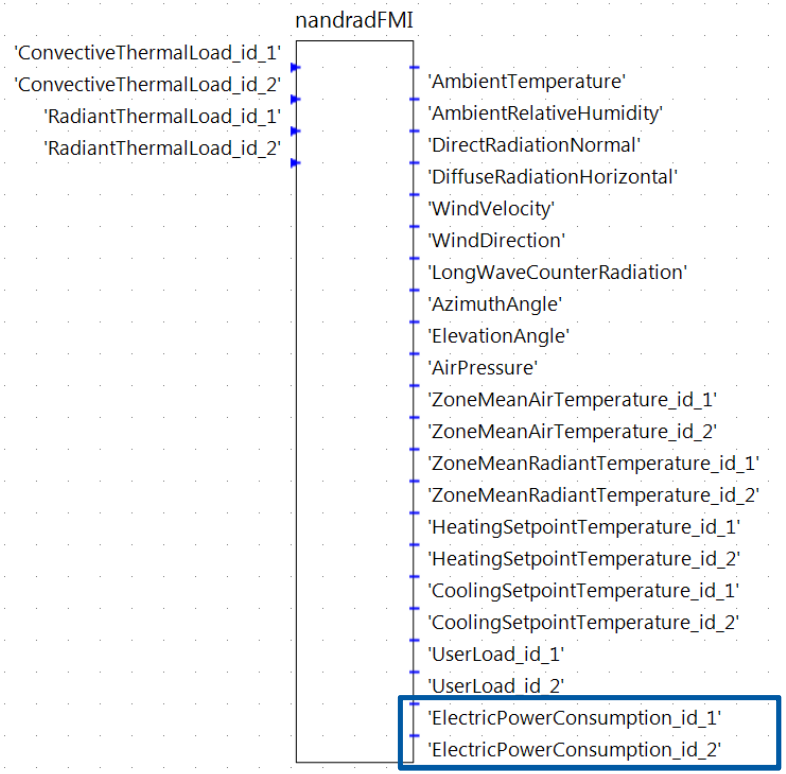
FMI Interface to NANDRAD Building Model FMU



Outputs

- Climate data
- Zone air/ radiant temperatures
- Setpoint temperatures (schedules)

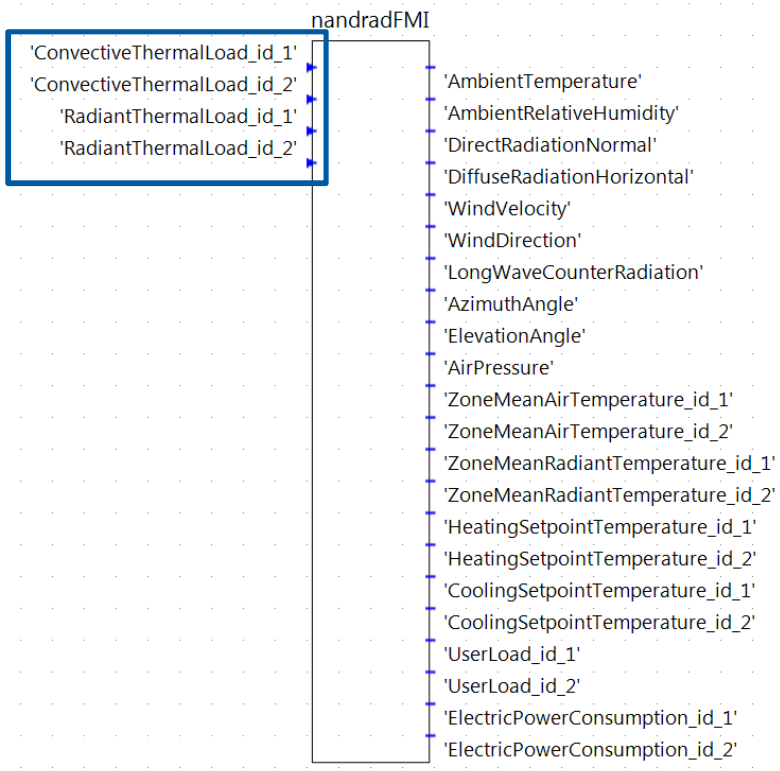
FMI Interface to NANDRAD Building Model FMU



Outputs

- Climate data
- Zone air/ radiant temperatures
- Setpoint temperatures
- Electric loads
- Warm water consumption

FMI Interface to NANDRAD Building Model FMU



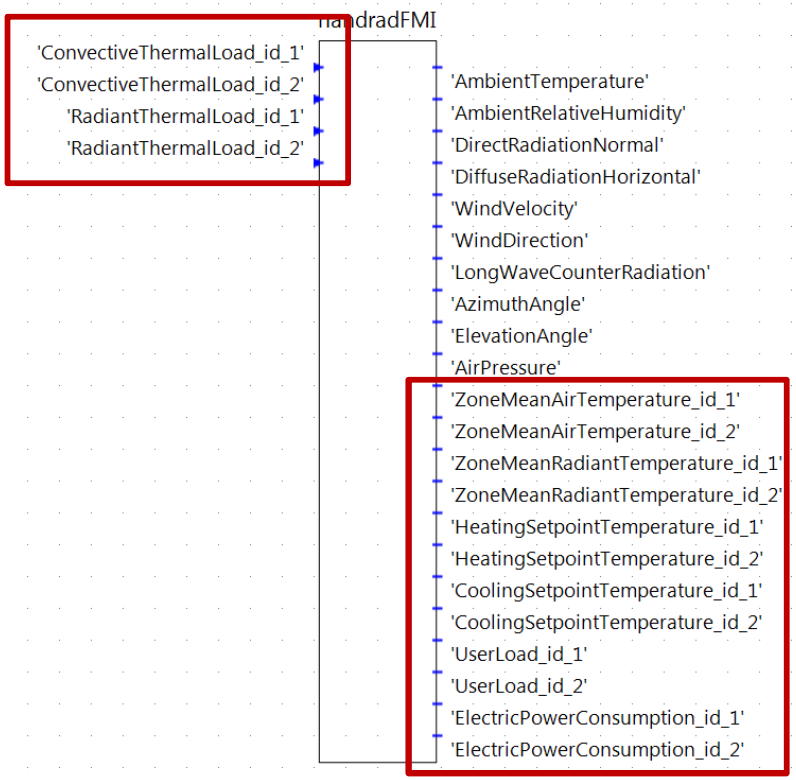
Outputs

- Climate data
- Zone air/ radiant temperatures
- Setpoint temperatures
- Electric loads

Inputs

- Convective and radiant thermal loads

FMI Interface to NANDRAD Building Model FMU



Outputs

- Climate data
- Zone air/ radiant temperatures
- Setpoint temperatures
- Electric loads

Inputs

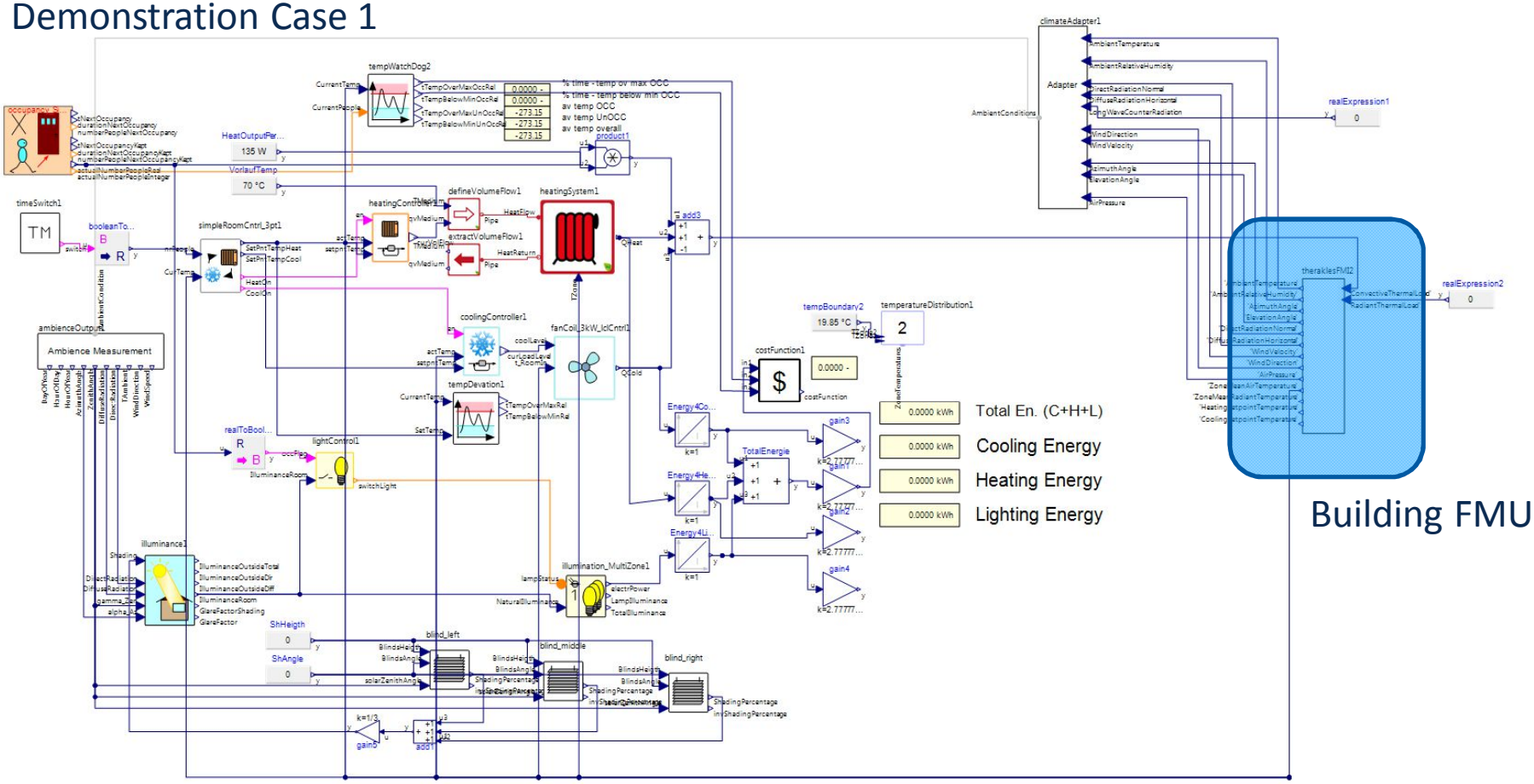
- Convective and radiant thermal loads

Zone specific inputs/outputs scale with number of zones! Manual connection ???

FMI Interface to NANDRAD Building Model FMU

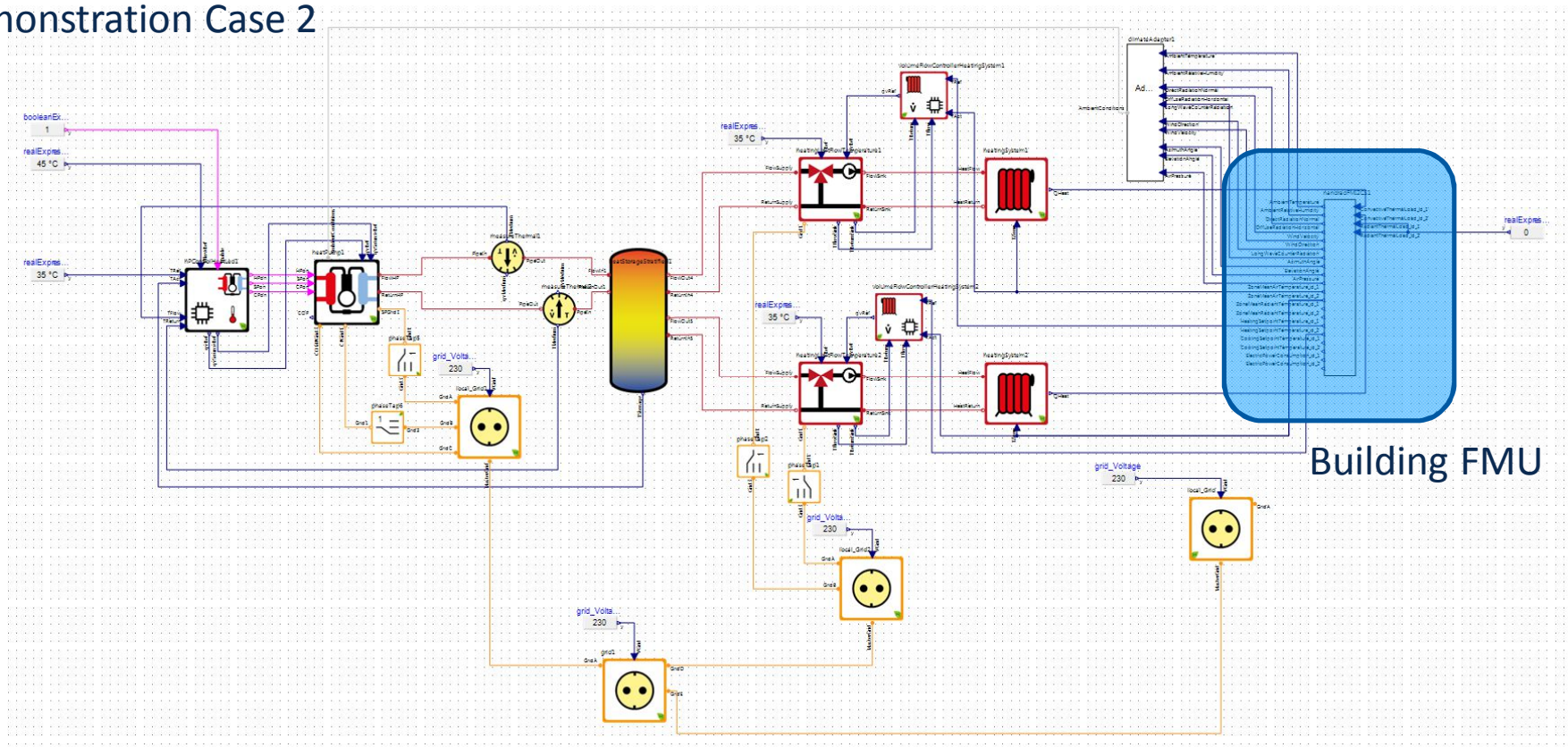
Application Scenario 1: Import into Modelica environment

Demonstration Case 1

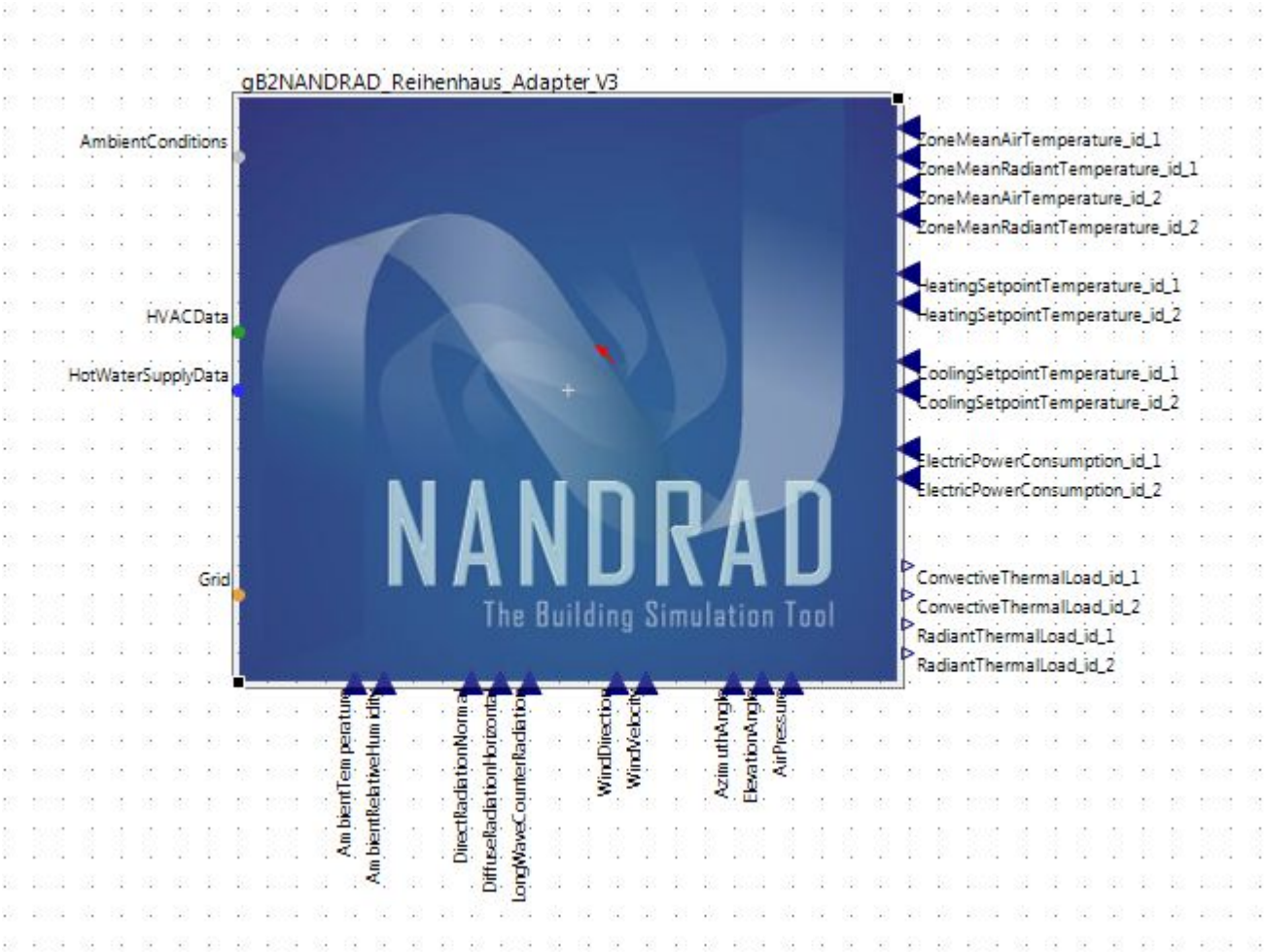


Application Scenario 1: Import into Modelica environment

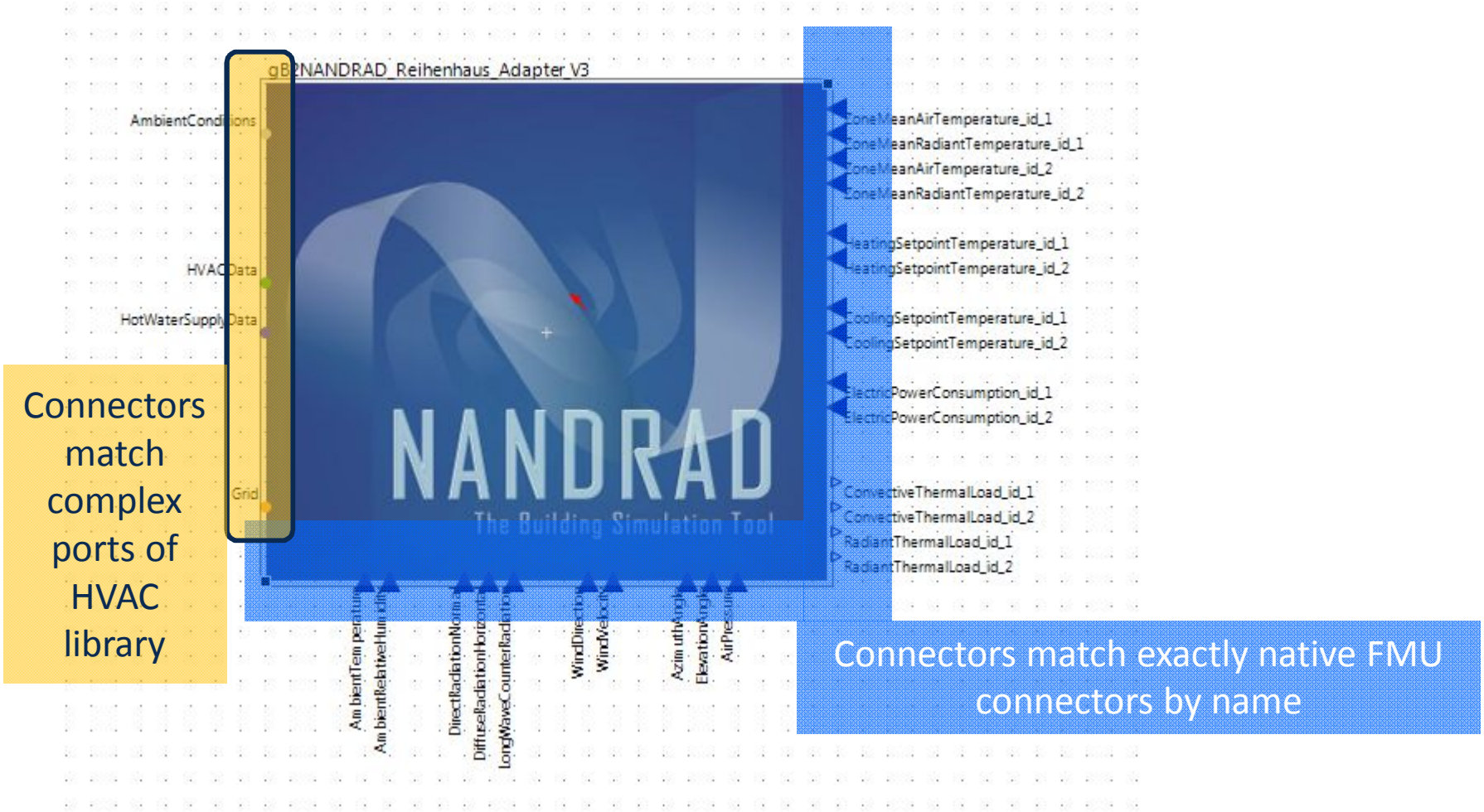
Demonstration Case 2



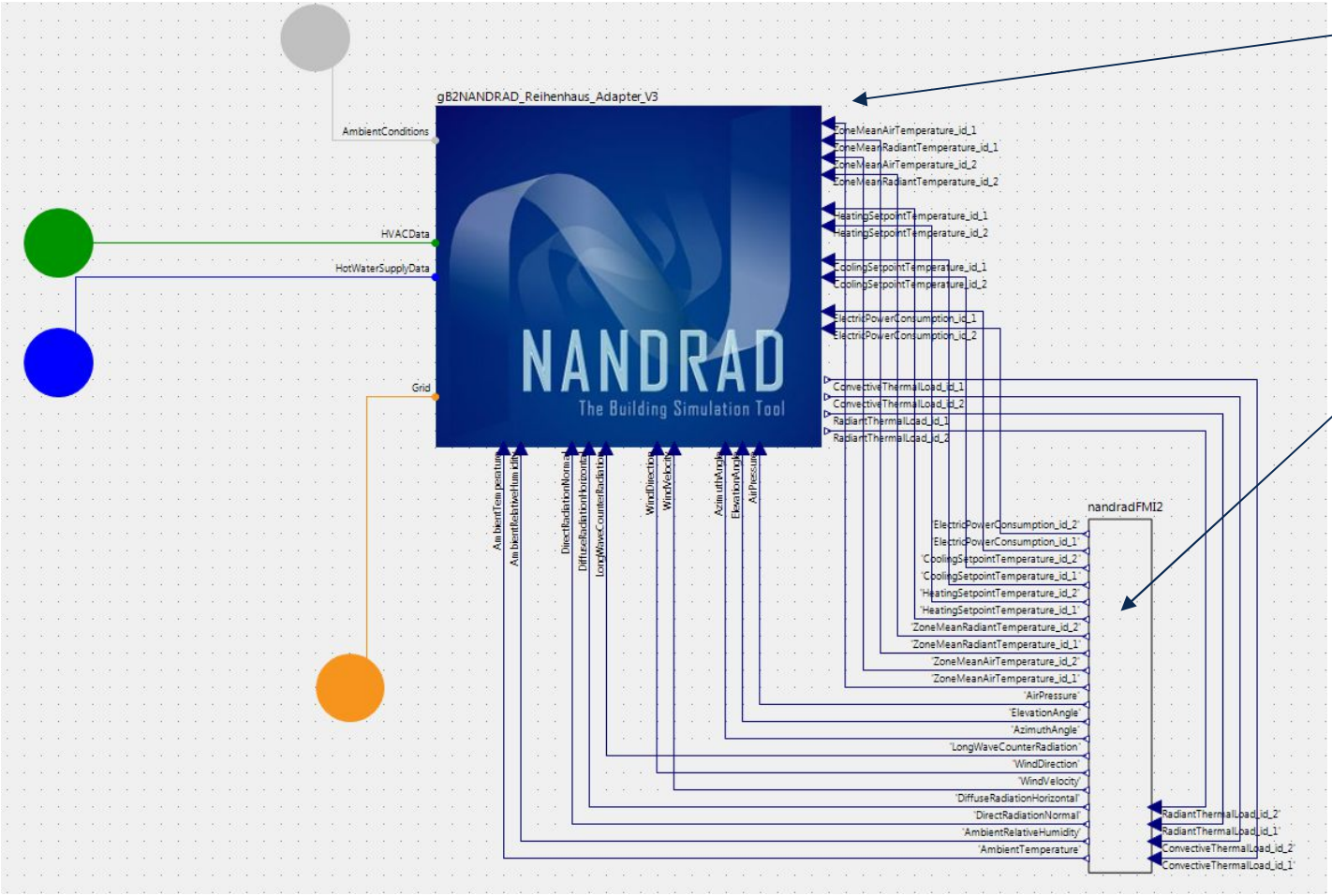
Introduce Modelica-Adapter Models (autogenerated)



Introduce Modelica-Adapter Models (autogenerated)



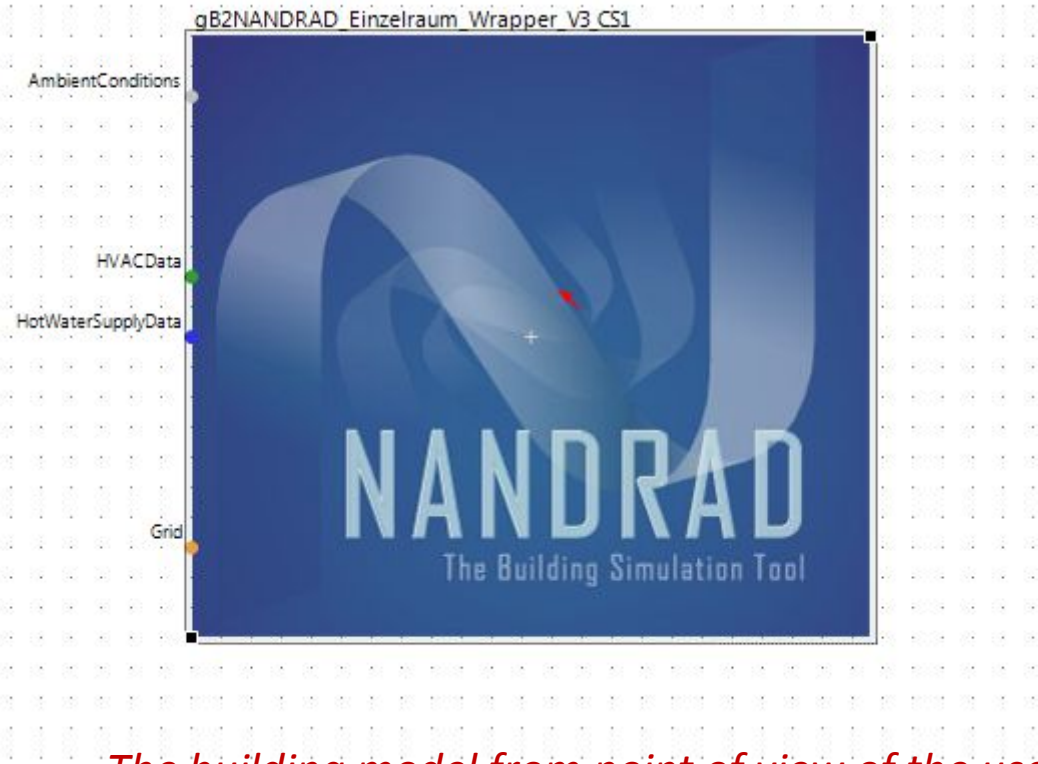
Introduce Modelica-Wrapper Models (autogenerated)



Adapter model

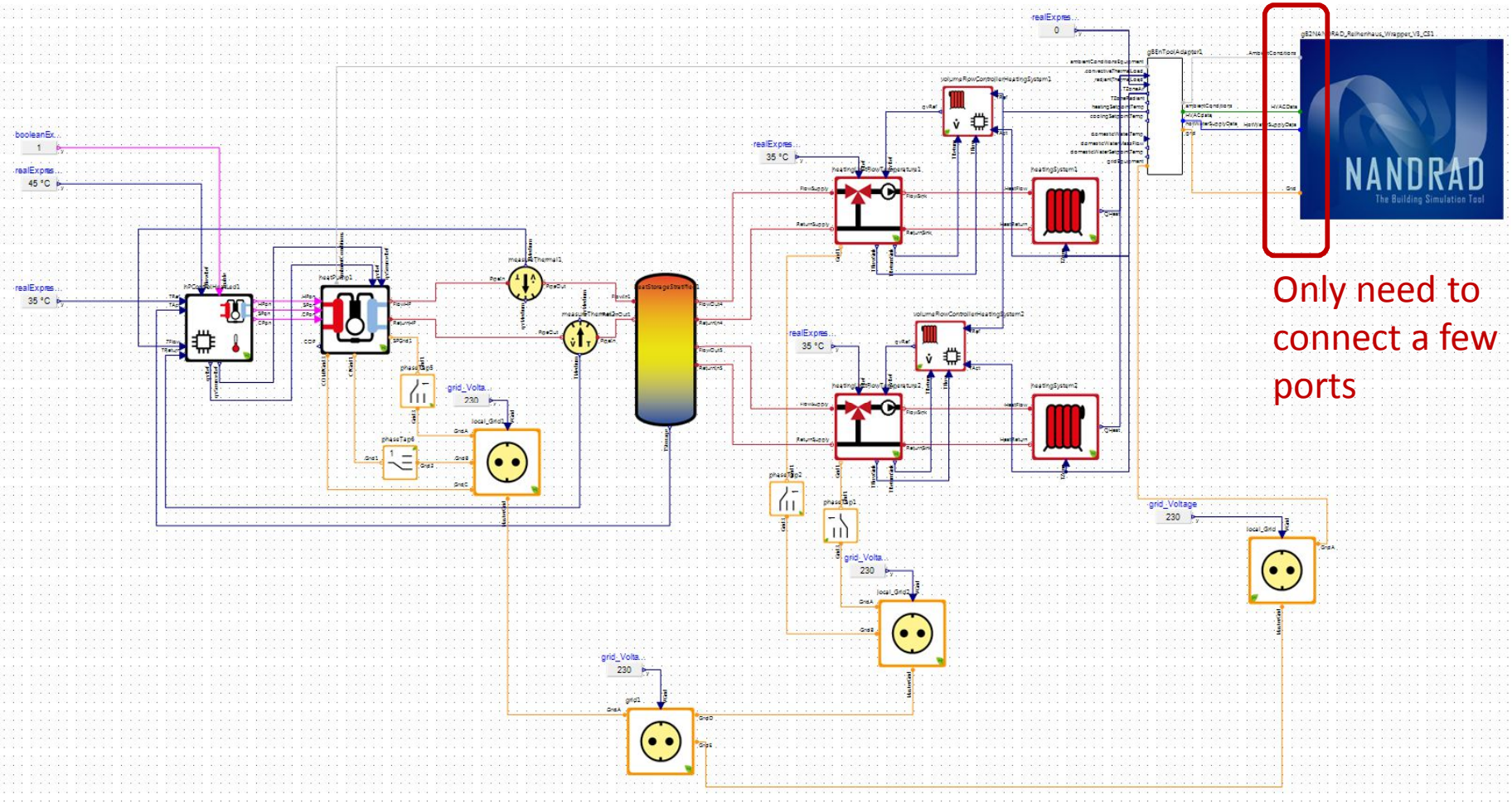
Imported FMU

Introduce Modelica-Wrapper Models (autogenerated)

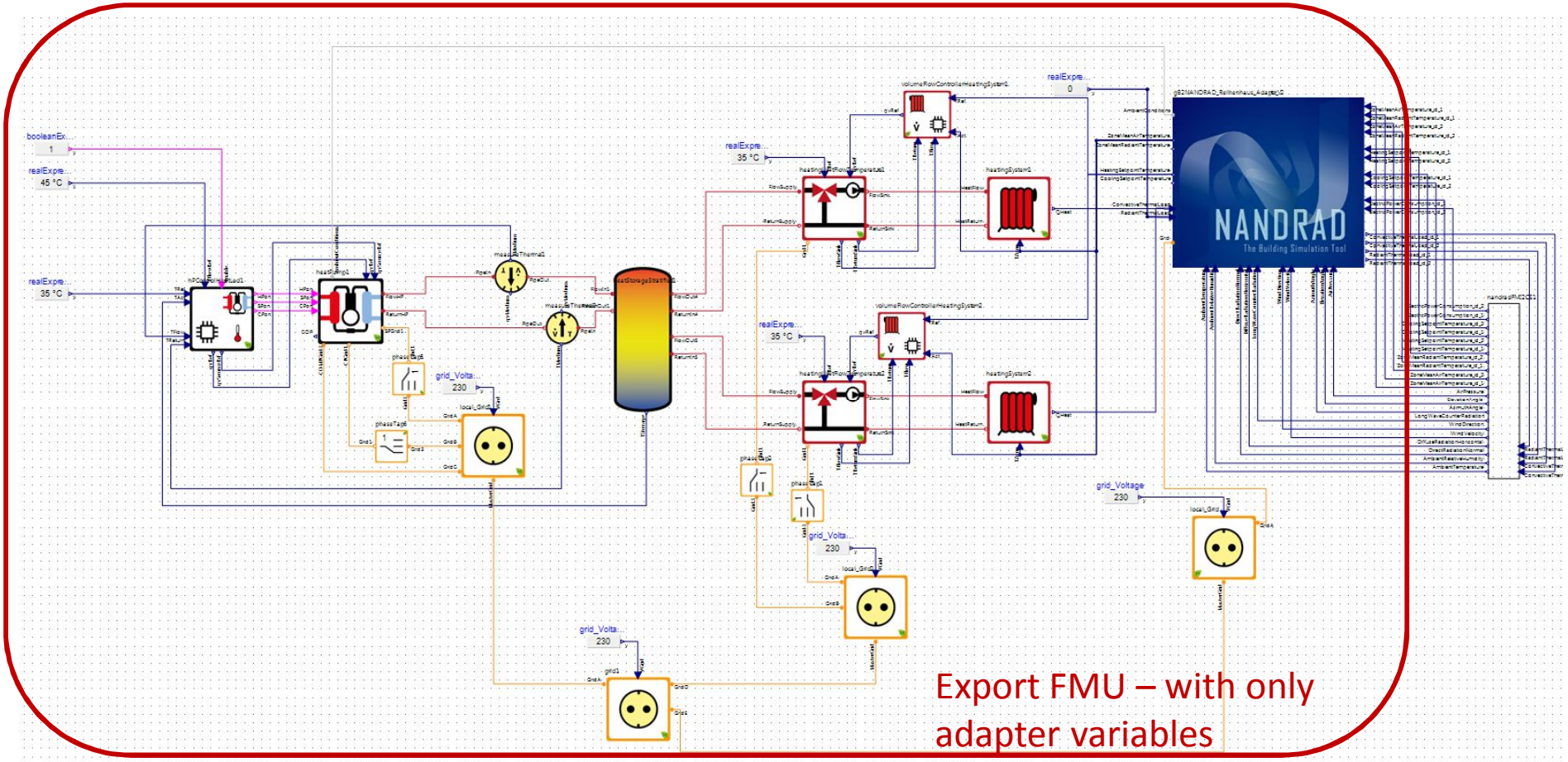


The building model from point of view of the user within the Modelica environment

Introduce Modelica-Wrapper Models (autogenerated)



Application Scenario 2: Export equipment FMU



Export FMU – with only adapter variables as FMI interface variables

Application Scenario 2: Co-Simulation with CoSim-Master

About MasterSim ...

MasterSim is an FMI Co-Simulation master and *programming library*. It supports the Functional Mockup Interface for Co-Simulation in Version 1.0 and 2.0. Using the functionality of Version 2.0, it implements various iteration algorithms that rollback FMU slaves and increase stablity of coupled simulation.

MasterSim is actively developed at the Technische Universität Dresden, Institut für Bauklimatik (see contact page).

Parts of MasterSim

MasterSim consists of three parts:

- MasterSimulator command line executable
- MasterSimulator user interface
- MasterSim programming library

Why use MasterSim?

First of all, it is **free**! The complete source code is available on its [SourceForge-project page] under an open source license.

But there are many other reasons for using MasterSim:

- MasterSim is a cross-platform development, available for Windows, Linux and MacOS
- Support for *scripted execution*
- Support for *several FMU instances* within the same co-simulation scenario
- Highly *configurable master algorithms*
- *Auto-connection feature* in user interface (very useful when connecting FMUs with many matching inputs and outputs)
- *Detailed simulation statistics* inform about where in your co-simulation is most of the simulation time spend; This is very helpful for performance tuning and selection of the optimal algorithm
- Support for parallel executed variation studies (several MasterSimulator processes), where the same FMU file is used concurrently in several simulations running in parallel

MasterSim Software Library

Library Functionality

With the library you can implement co-simulation functionality into your own simulation programs with little effort.

FMU Import functionality

The library supports extraction of F
`modelDescription.xml` files, importing st
memory. This is all neatly encapsulated in co

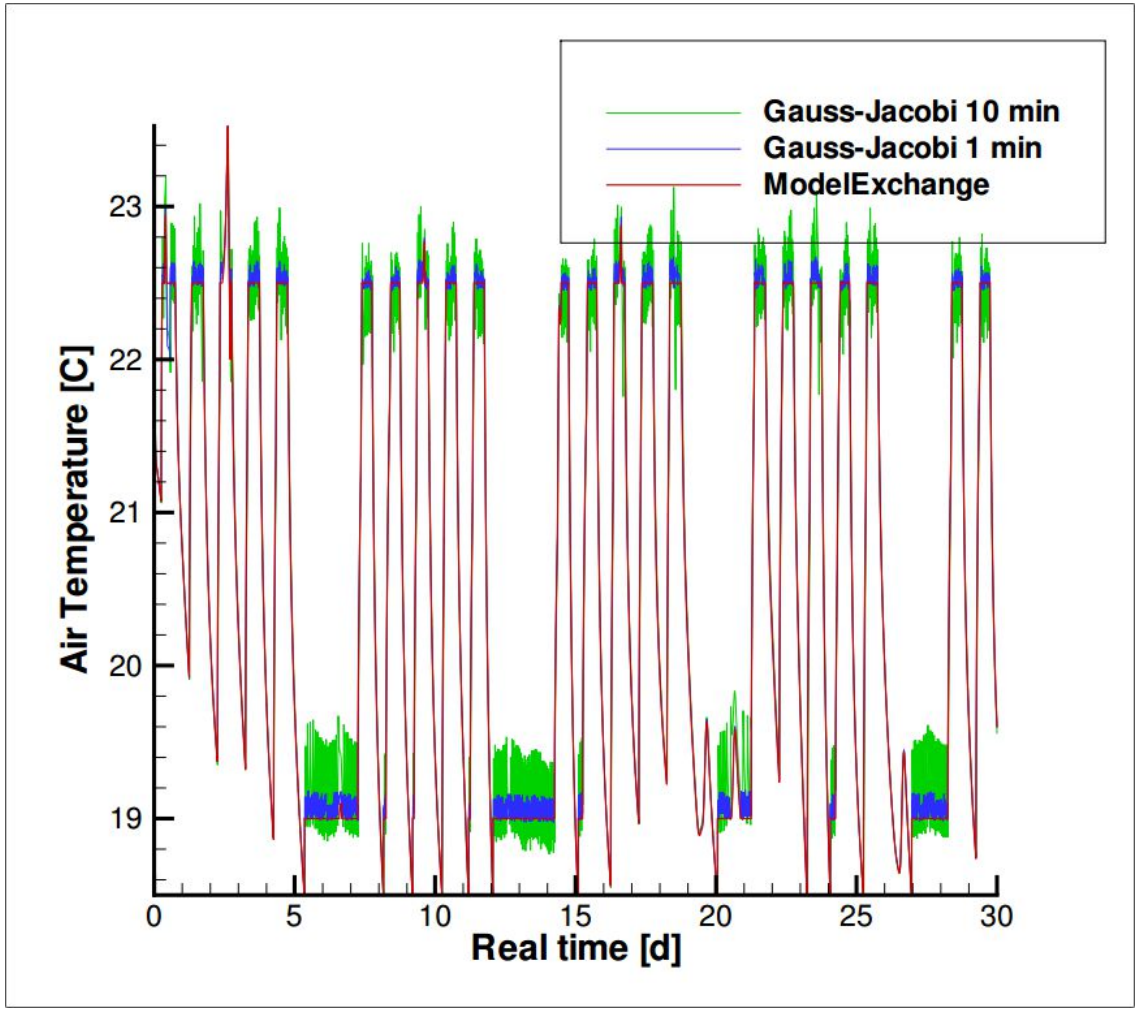
Master-Algorithms

Included in the library source code are stan
as *Gauss-Jacobi*, *Gauss-Seidel* and *Newton*.
own code without any work necessary (
numerical parameters, perhaps).

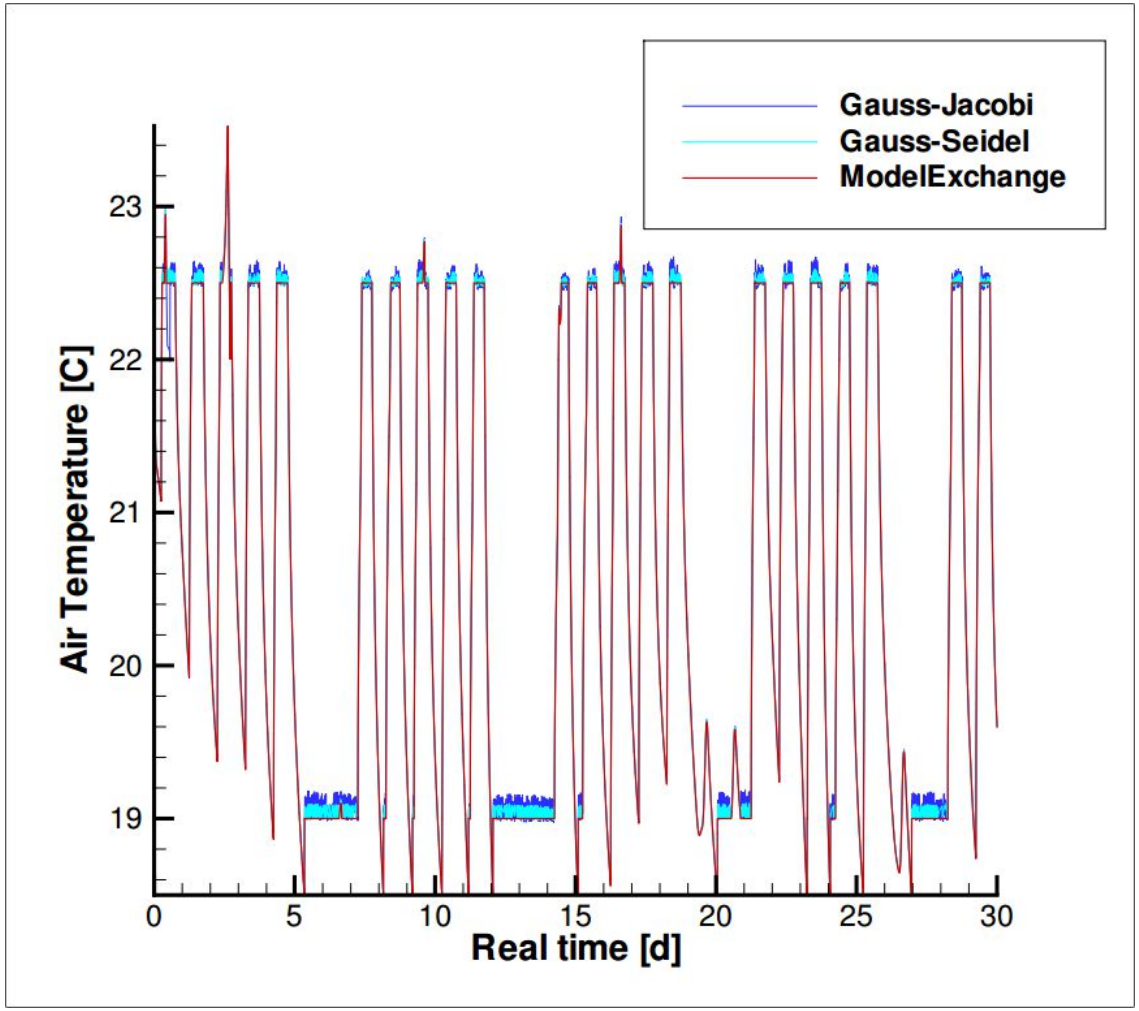
Open-Source Co-Simulation Master developed at TU Dresden, Germany

http://mastersim.sourceforge.net

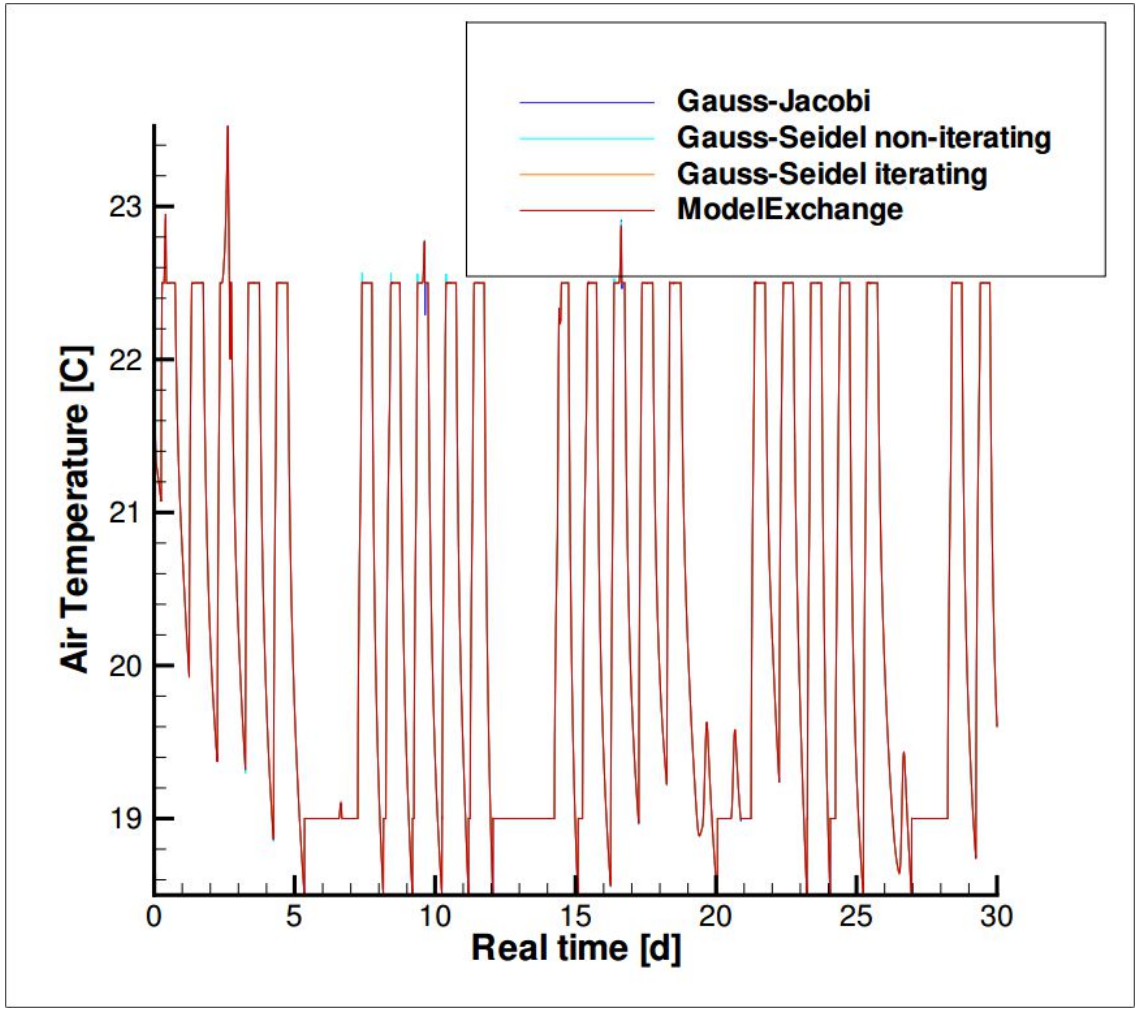
Co-Simulation: fixed-step size – stability problems



Co-Simulation: Fixed step size – Gauss-Seidel better



Co-Simulation: with adaptive step-size (error controlled)



Summary

1. Building energy simulation NANDRAD can now be used as FMI for ME/CS (version 2, with get/set state capability)
2. Use case 1 (within Modelica environment):
   a. NANDRAD-FMU substitutes simpler building models (when needed)
   b. Modelica-Wrapper-Models significantly simplifies connecting FMUs with hundreds of input/output variables
3. Use case 2 (Equipment FMU + Building FMU running in external Co-Simulation Master)
   a. Use of adapter models simplifies setup of co-simulation szenario
   b. Choice of master algorithm has great impact on stability/accuracy and performance

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Architektur, Institut für Bauklimatik, Professur für Bauphysik

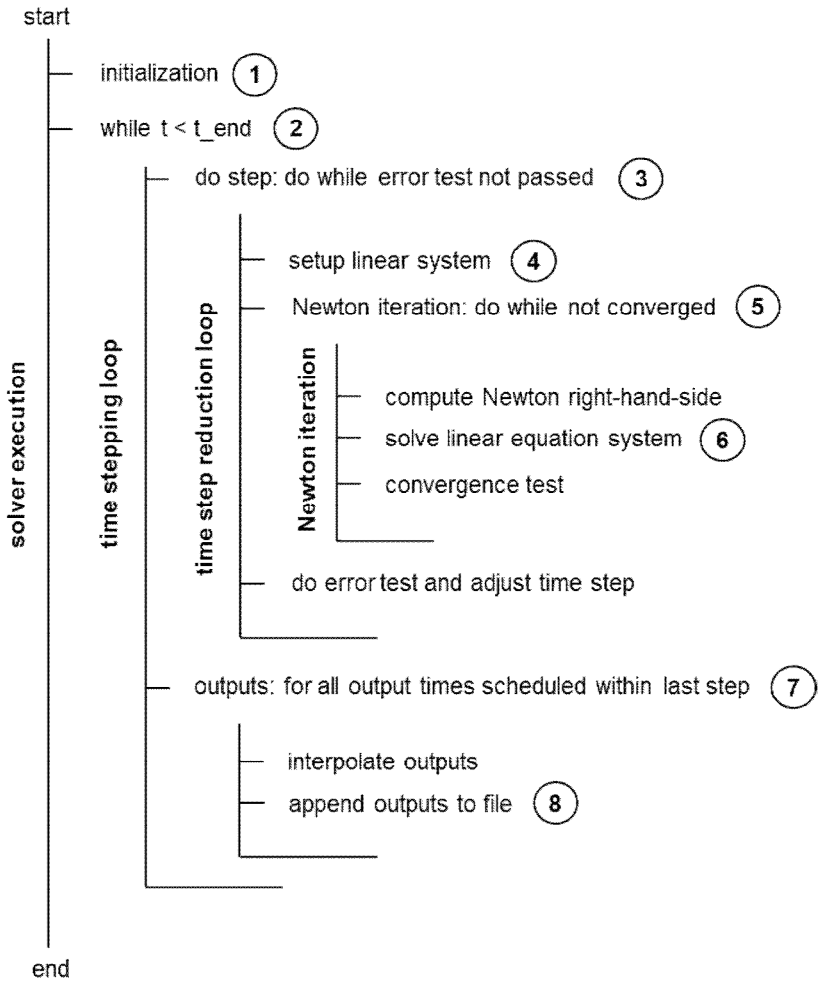
Co-Simulation between detailed building energy performance simulation and Modelica HVAC component models

Andreas Nicolai and Anne Paepcke

andreas.nicolai@tu-dresden.de

Dresden, 16.09.2016

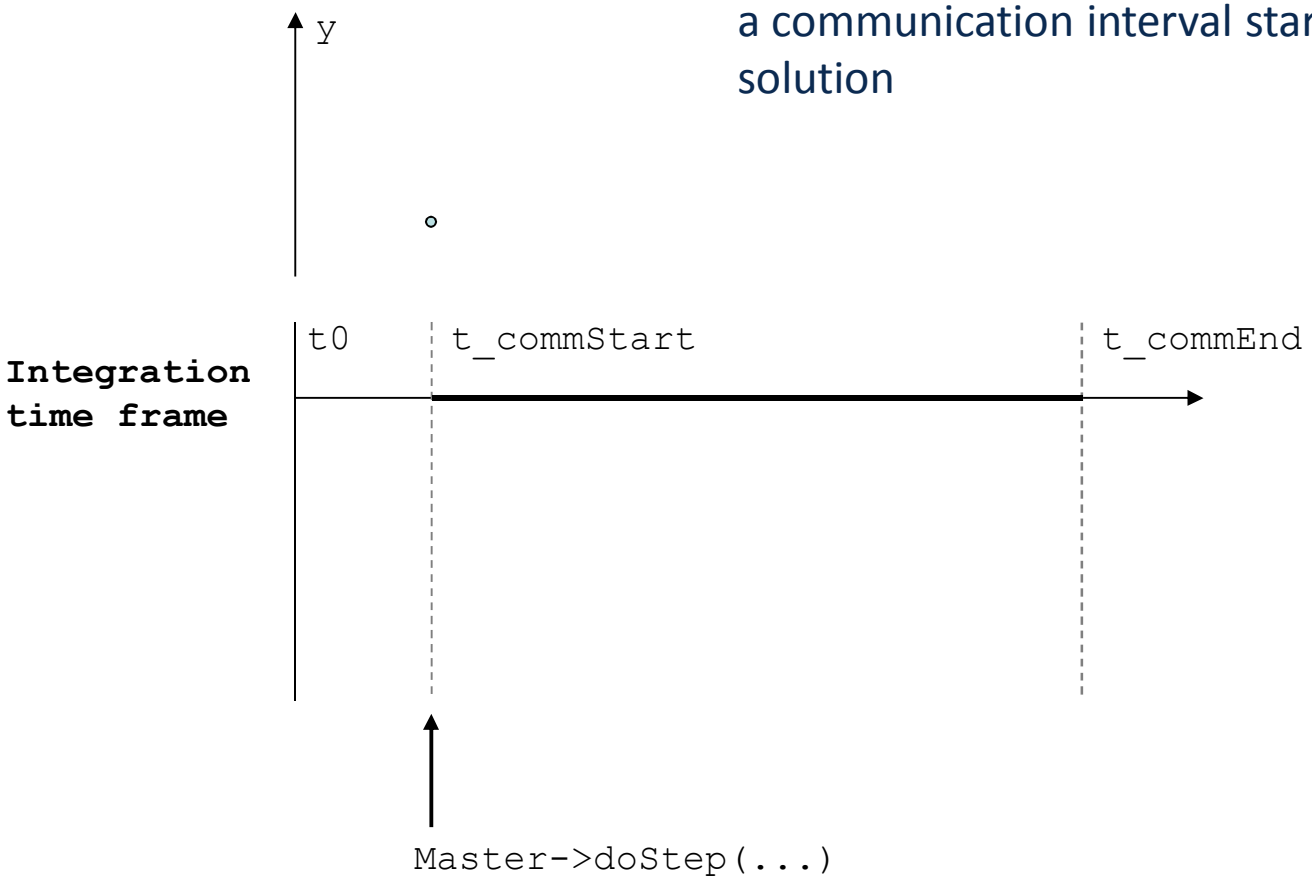
Internal functionality (only for discussion)



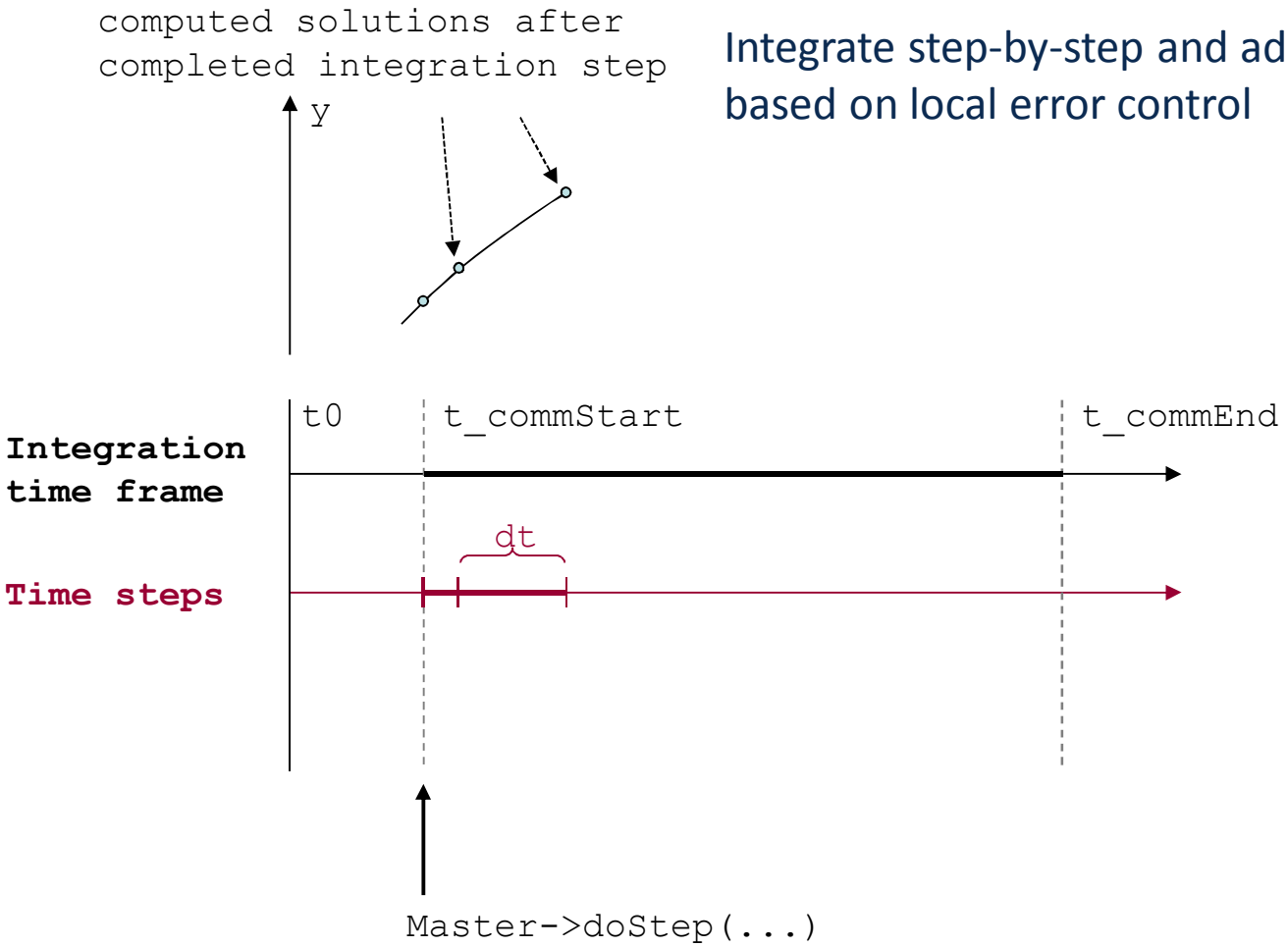
Functionality within Co-Simulation Master

1. Model acts as a Co-Simulation slave
2. Interacts with Master through get/set functions
3. Time integration is done within a communication step when Master calls doStep()

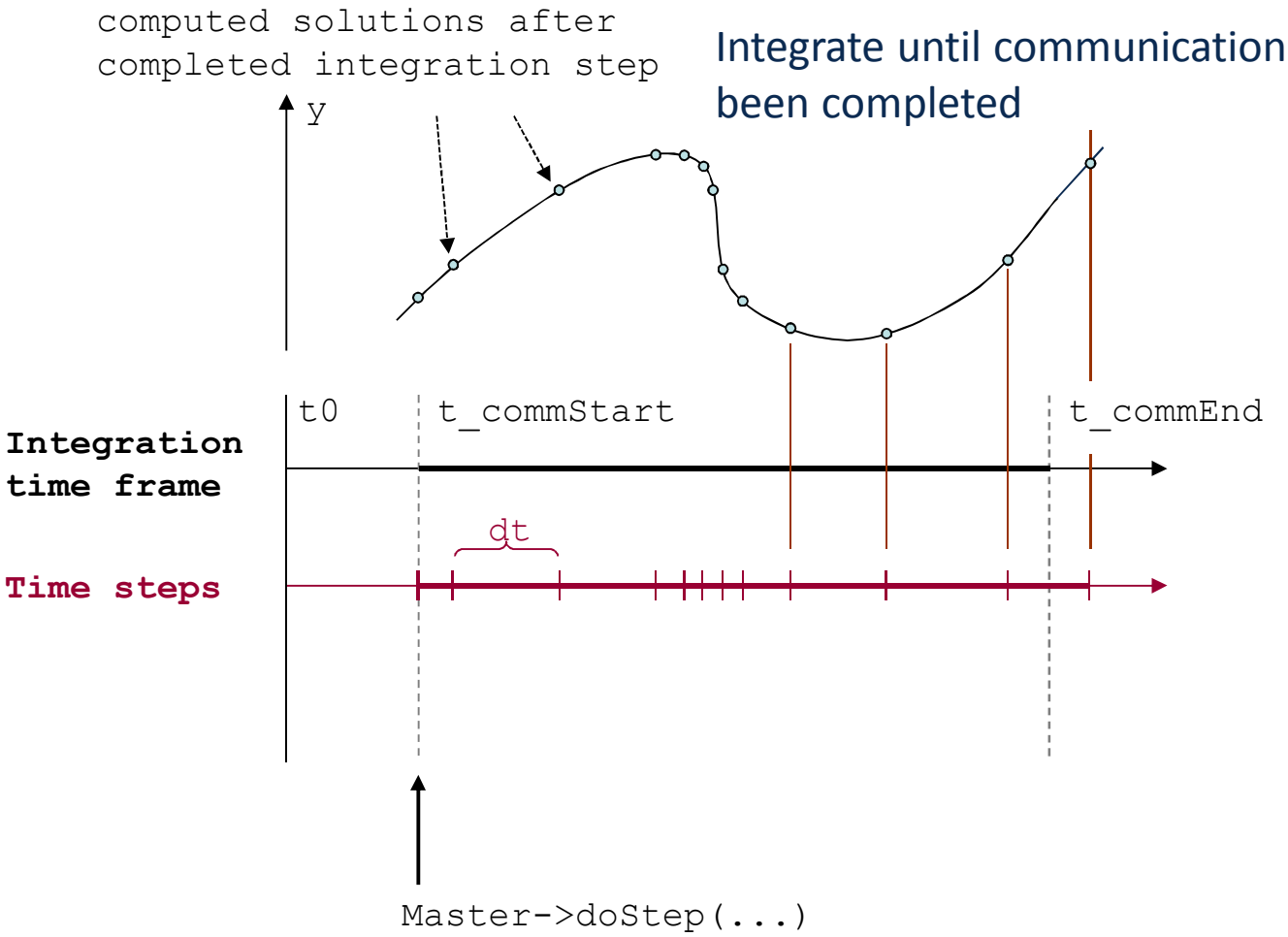
Master requests slave to integrate over a communication interval starting from a known solution



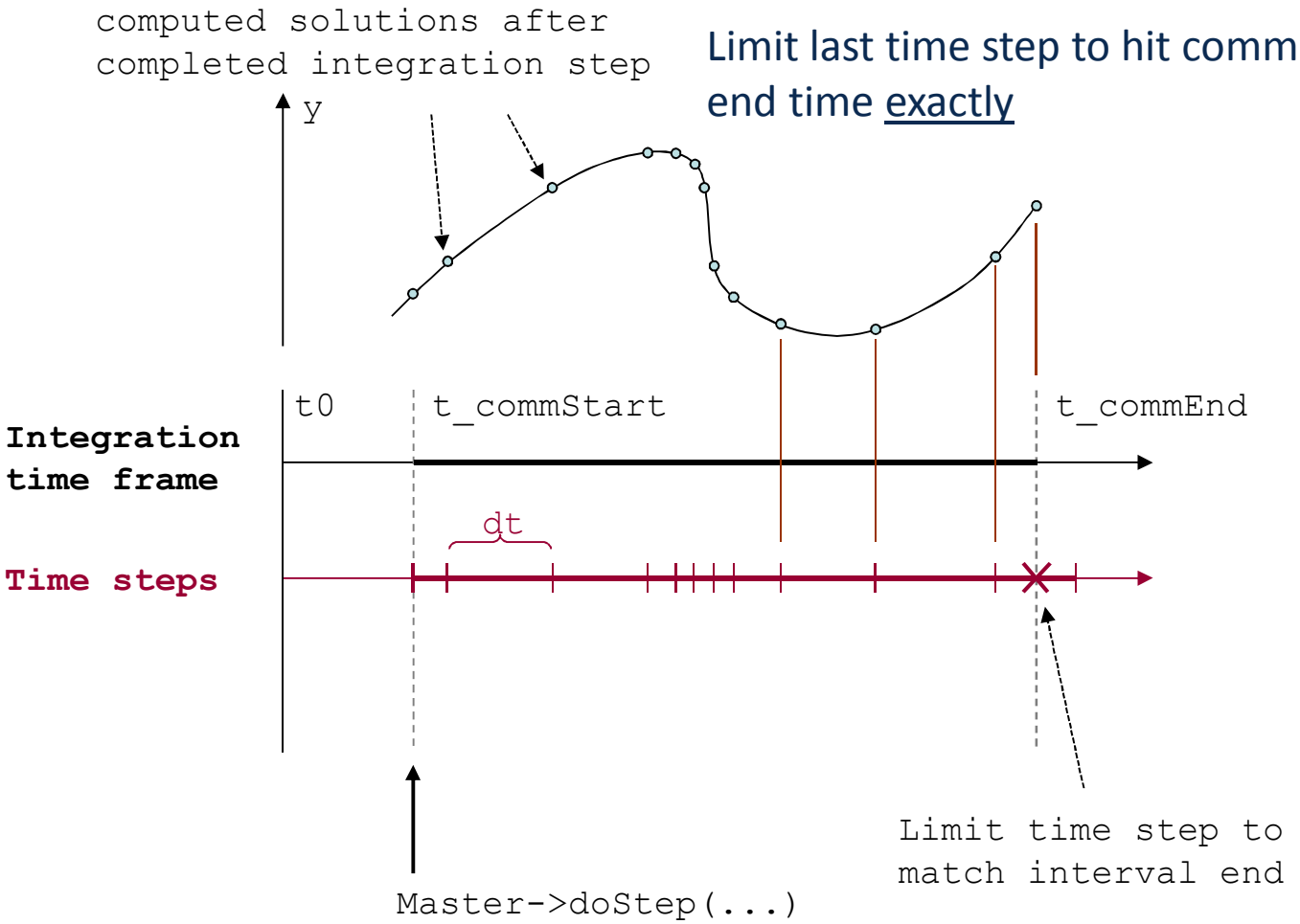
Integrate step-by-step and adapt time step size based on local error control



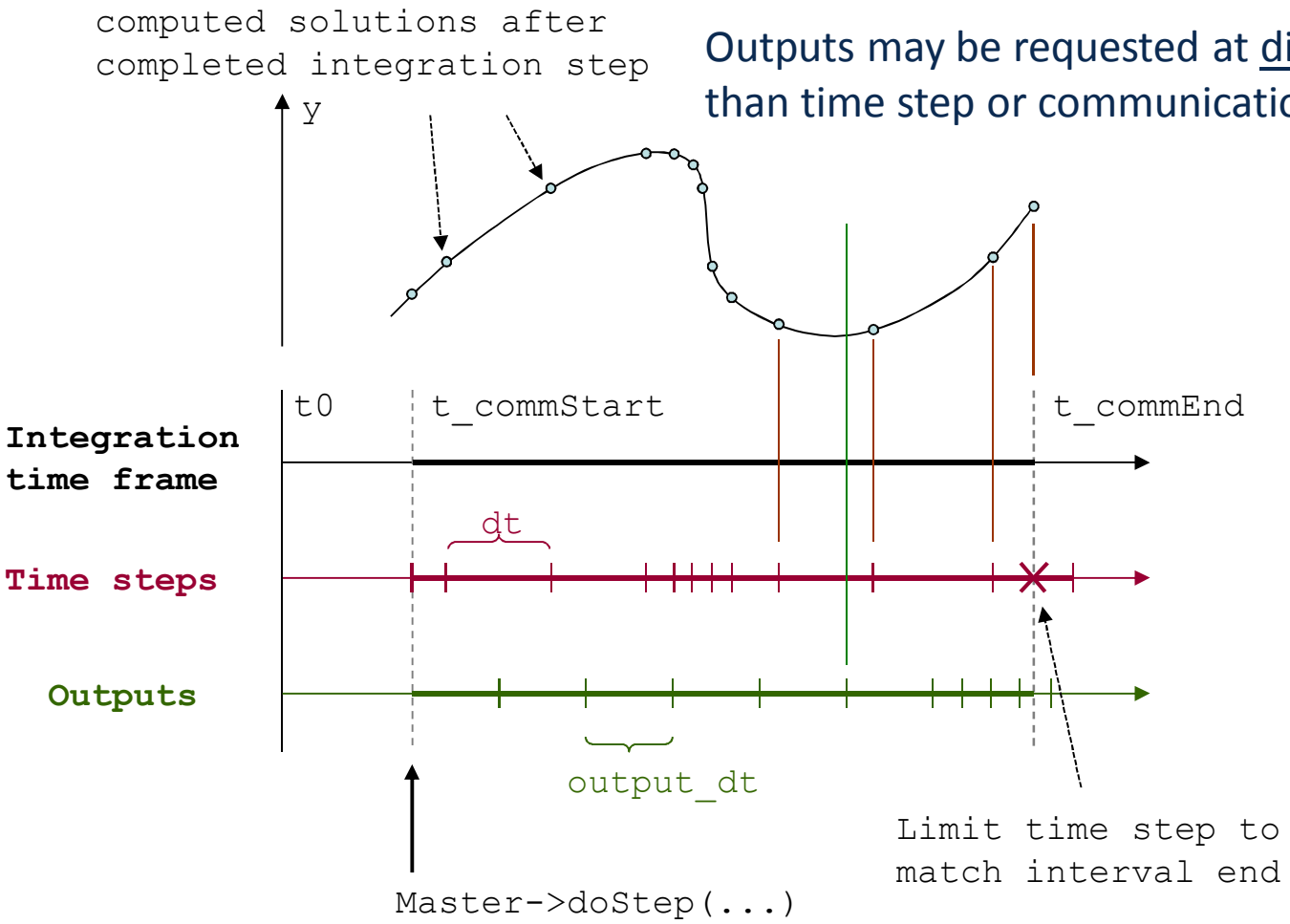
Integrate until communication interval has been completed



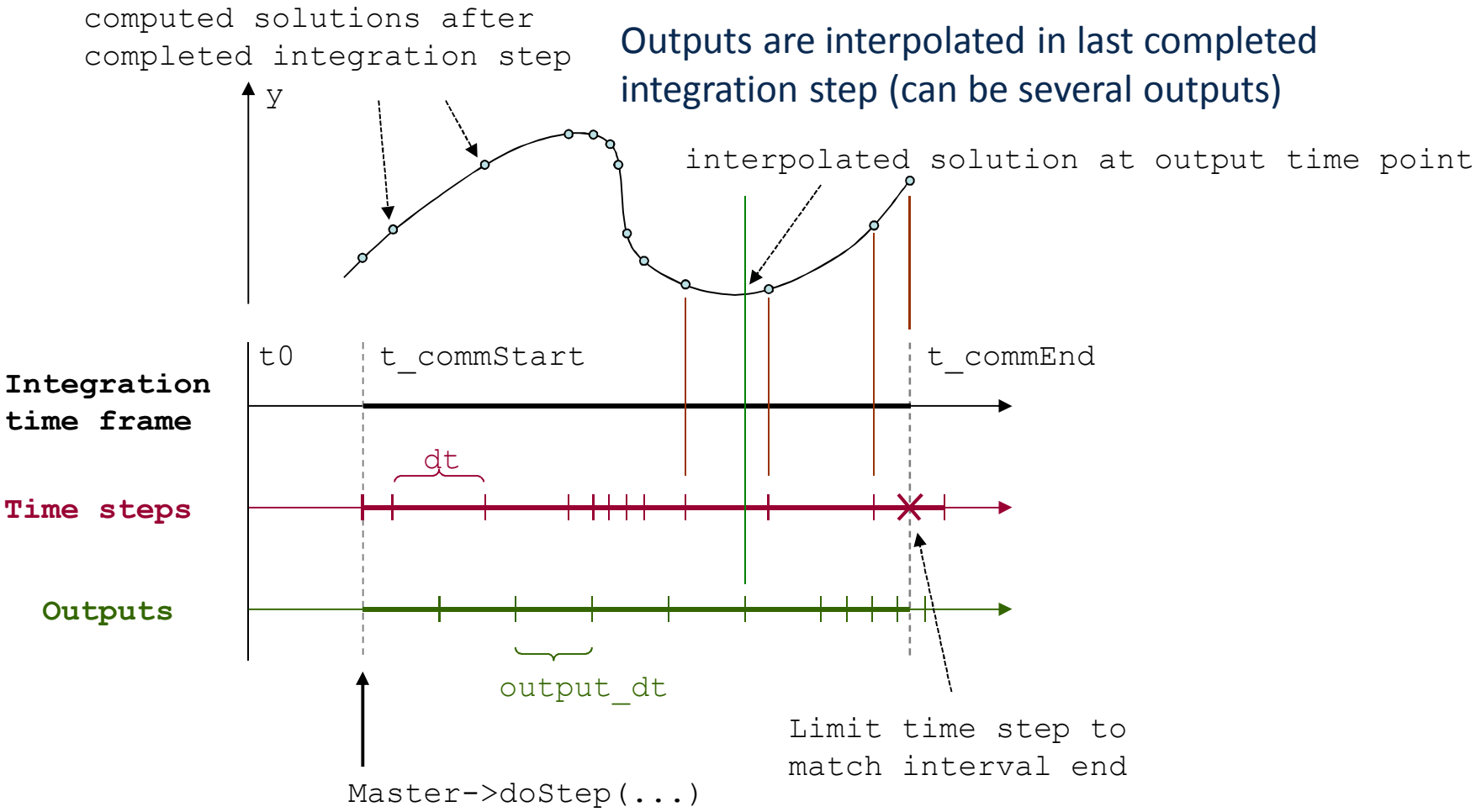
Limit last time step to hit communication interval end time exactly



Outputs may be requested at different times than time step or communication interval end time



Outputs are interpolated in last completed integration step (can be several outputs)



Things to consider…

1. Limit last time step
2. When interpolating model state backwards in last step, restore model state to end of communication interval before returning control to master

… otherwise pretty straight forward …

Iterative Master algorithms

- Gauss-Jacobi, Gauss-Seidel, Newton
- Master requests Slave to store its state
- Master can request Slave to store multiple states
- Master requests Slave to reset its state back to a previously stored state (roll-back)
- Related functions:
  - fmi2Serialize(), fmi2Deserialize(), fmi2SerializedFMUstateSize(), fmi2GetFMUState(), fmi2SetFMUState()

Getting and Setting FMU State

State is defined by:

- Integrator (conservation variables of ODEs)
- LES solver (for direct solvers factorized Jacobian)
- Jacobian matrix (unfactorized)
- Preconditioner (unfactorized and in case of ILU also factorized)
- Model state (e.g. hysteresis variables)

Getting and Setting FMU State

Fragmented memory structures
→ Direct copy not meaningful (and not fast)

→ Copy into sequential memory first

Implement first serialize and deserialize functionality

Getting FMU State via Serialization

Typical implementation of fmi2GetFMUState():

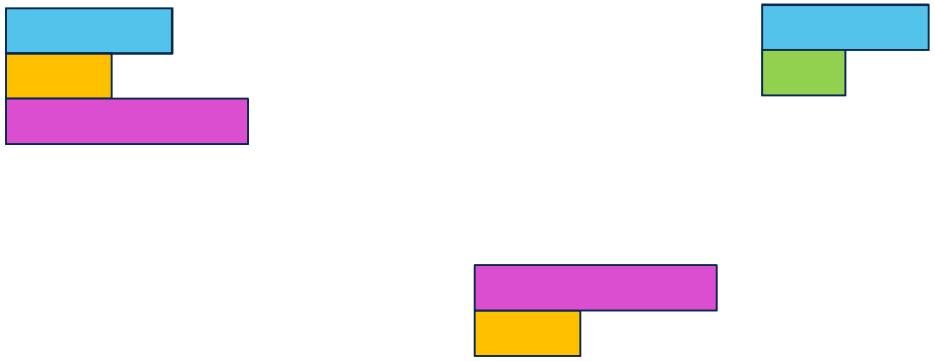
```
// check if new alloc is needed
if (*FMUstate == NULL) {
    // alloc new memory
    fmi2FMUstate fmuMem = malloc(modelInstance->m_fmuStateSize);
    // remember this memory array
    modelInstance->m_fmuStates.insert(fmuMem);
    // store size of memory in first 8 bytes of fmu memory
    *(size_t*)(fmuMem) = modelInstance->m_fmuStateSize;
    // return newly created FMU mem
    *FMUstate = fmuMem;
}
else {
    // check if FMUstate is in list of stored FMU states
    if (modelInstance->m_fmuStates.find(*FMUstate) == modelInstance->m_fmuStates.end()) {
        modelInstance->logger(fmi2Error, "error", "fmi2GetFMUstate is called with invalid "
          "FMUstate (unknown or already released pointer).");
        return fmi2Error;
    }
}

// now copy FMU state into memory array
modelInstance->serializeFMUstate(*FMUstate);

return fmi2OK;
```

Serialization into sequential memory

Fragmented memory structures

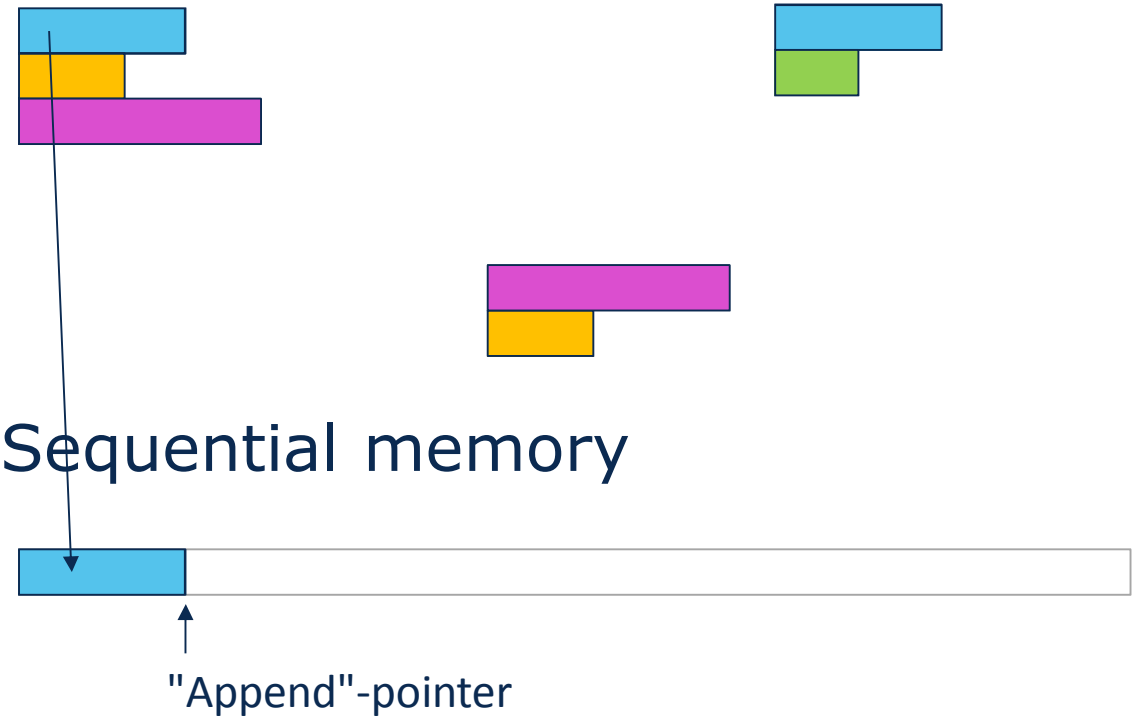


Sequential memory

"Append"-pointer

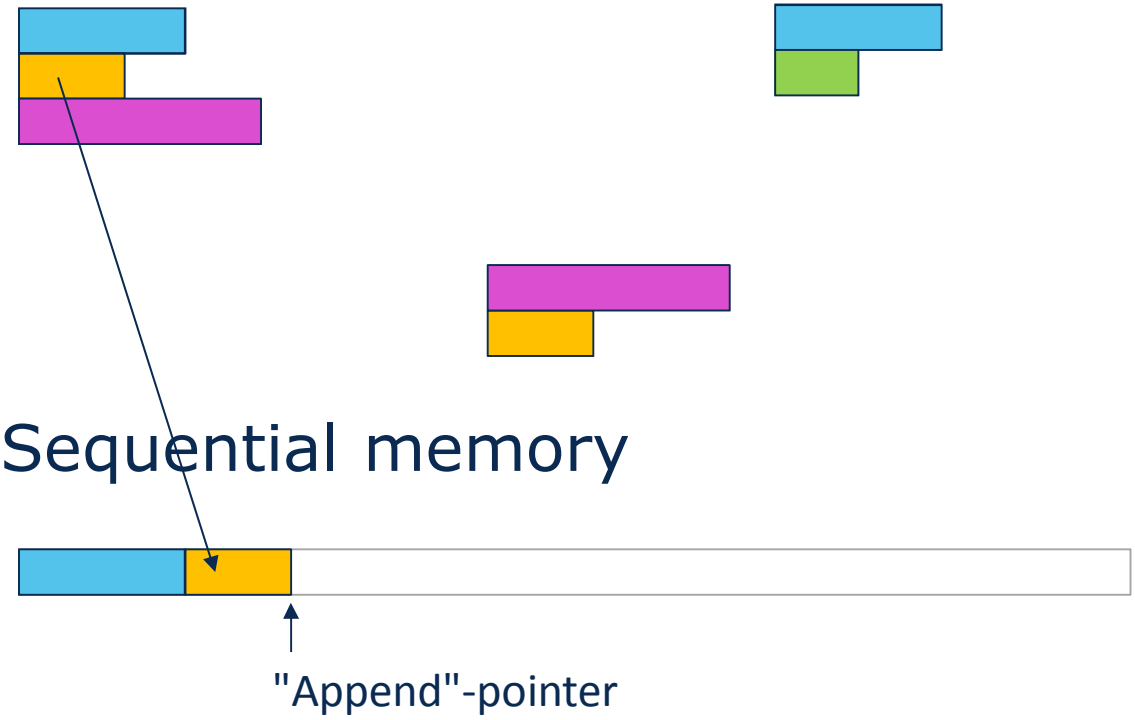
Serialization into Sequential Memory

Fragmented memory structures



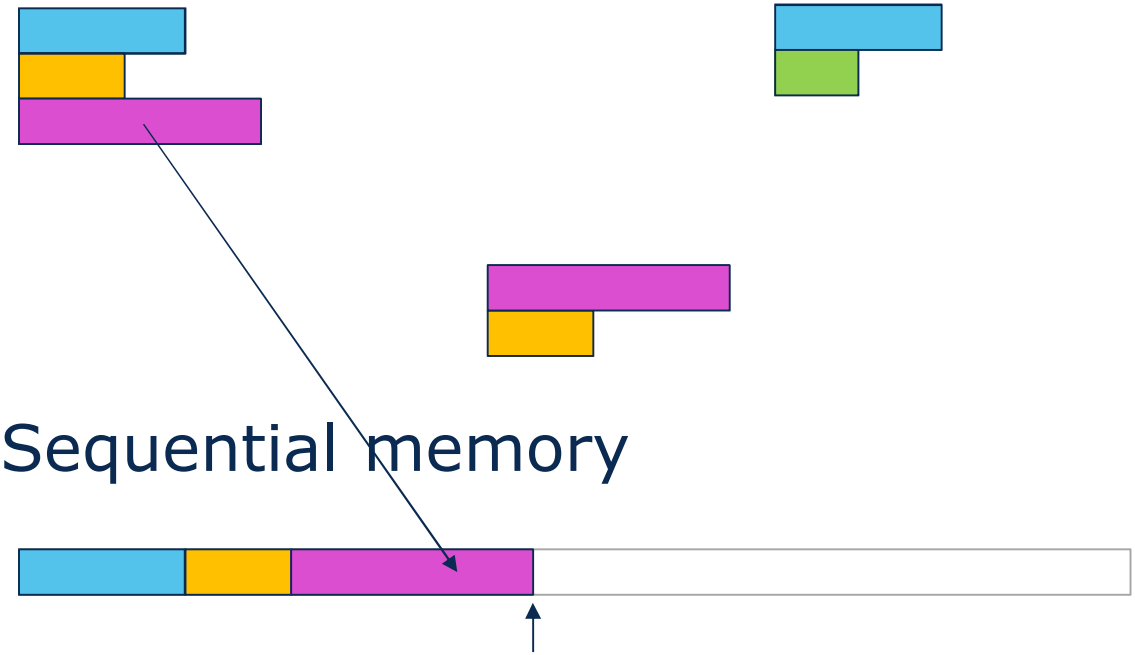
Serialization into Sequential Memory

Fragmented memory structures



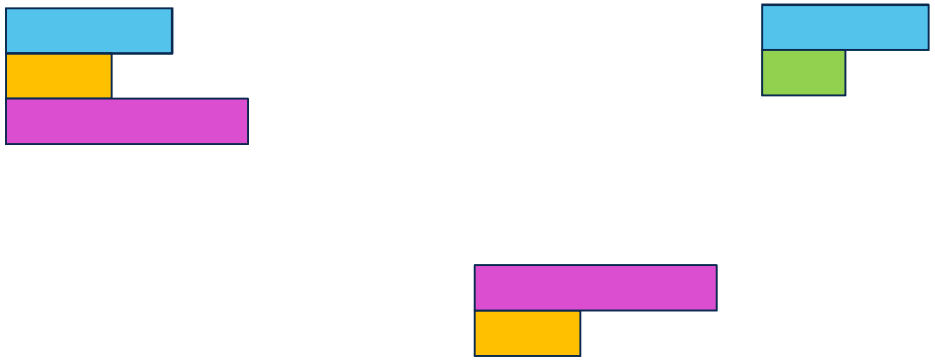
Serialization into Sequential Memory

Fragmented memory structures



Serialization into Sequential Memory

Fragmented memory structures



Sequential memory

Serialization into Sequential Memory

Maintainable code → Single code for:

- Size calculation
- Copy into sequential memory
- Copy back from sequential memory

For C compatibility (e.g. for CVODE integrator) use Macros

Serialization into Sequential Memory

```
1. #define SERIALIZE(type, storageDataPtr, value)\
2. {\
3.   *(type *)(storageDataPtr) = (value);\
4.   (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
5. }
6.
7. #define DESERIALIZE(type, storageDataPtr, value)\
8. {\
9.   (value) = *(type *)(storageDataPtr);\
10.  (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
11. }
```

Serialization into Sequential Memory

```
1. #define SERIALIZE(type, storageDataPtr, value)\
2. {\
3.   *(type *)(storageDataPtr) = (value);\
4.   (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
5. }
6.
7. #define DESERIALIZE(type, storageDataPtr, value)\
8. {\
9.   (value) = *(type *)(storageDataPtr);\
10.   (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
11. }
```

```
1. #define CVODE_SERIALIZE_A(op, type, storageDataPtr, value, mem_size)\

2.   switch (op) {\
3.   case SUNDIALS_SERIALIZATION_OPERATION_SERIALIZE:\
4.     SERIALIZE(type, storageDataPtr, value)\
5.     break;\
6.   case SUNDIALS_SERIALIZATION_OPERATION_DESERIALIZE:\
7.     DESERIALIZE(type, storageDataPtr, value)\
8.     break;\
9.   case SUNDIALS_SERIALIZATION_OPERATION_SIZE:\
10.     mem_size += sizeof(type);\
11.     break;\
12.   }
```

Serialization into Sequential Memory

Single code for mapping memory depending on selected operation type (op):

```
CVODE_SERIALIZE_A(op, booleantype, *storageDataPtr,
    cv_mem->cv_tstopset, memSize)
CVODE_SERIALIZE_A(op, realtype, *storageDataPtr,
    cv_mem->cv_tstop, memSize)

/* current order */
CVODE_SERIALIZE_A(op, int, *storageDataPtr,
    cv_mem->cv_q, memSize)
/* order to be used on the next step = q-1, q, or q+1 */
CVODE_SERIALIZE_A(op, int, *storageDataPtr,
    cv_mem->cv_qprime, memSize)
```

Aspects of creating a coupled building energy simulation

1. Adding FMI ME+CS interface to a stand-alone simulation engine
2. Implementing FMU v2 setting/getting-state capability
3. FMU generation process + resource handling
4. Application scenarios
5. Interface conventions
6. Usability (Modelica-Wrapper/Adaptors)
7. Demonstration cases (findings and experience gained)