TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Architektur, Institut für Bauklimatik, Professur für Bauphysik

Transformation of a building energy simulation tool with variable time step solver into an FMU for CoSimulation v.2 with serialization capabilities

Andreas Nicolai and Anne Paepcke

andreas.nicolai@tu-dresden.de

Dresden, 16.09.2016

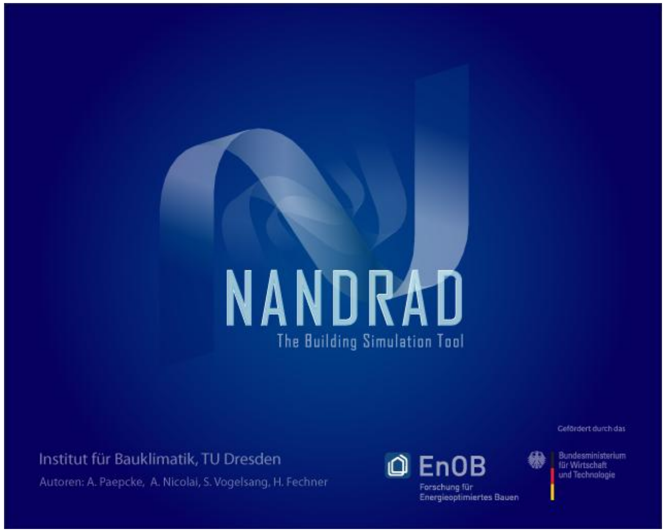
TECHNISCHE UNIVERSITÄT DRESDEN

Overview

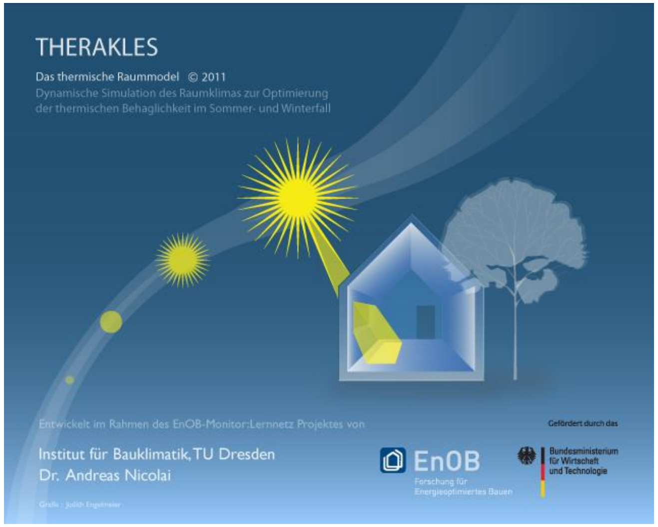
1. Some background on our tools
2. Principle functionality of variable time stepping integrators
3. Setting/Getting FMU States
4. Serialization into sequential memory
5. Output handling when iterating over FMUs

TECHNISCHE UNIVERSITÄT DRESDEN

Simulation tools – NANDRAD and THERAKLES



Multi-zone building energy simulation model

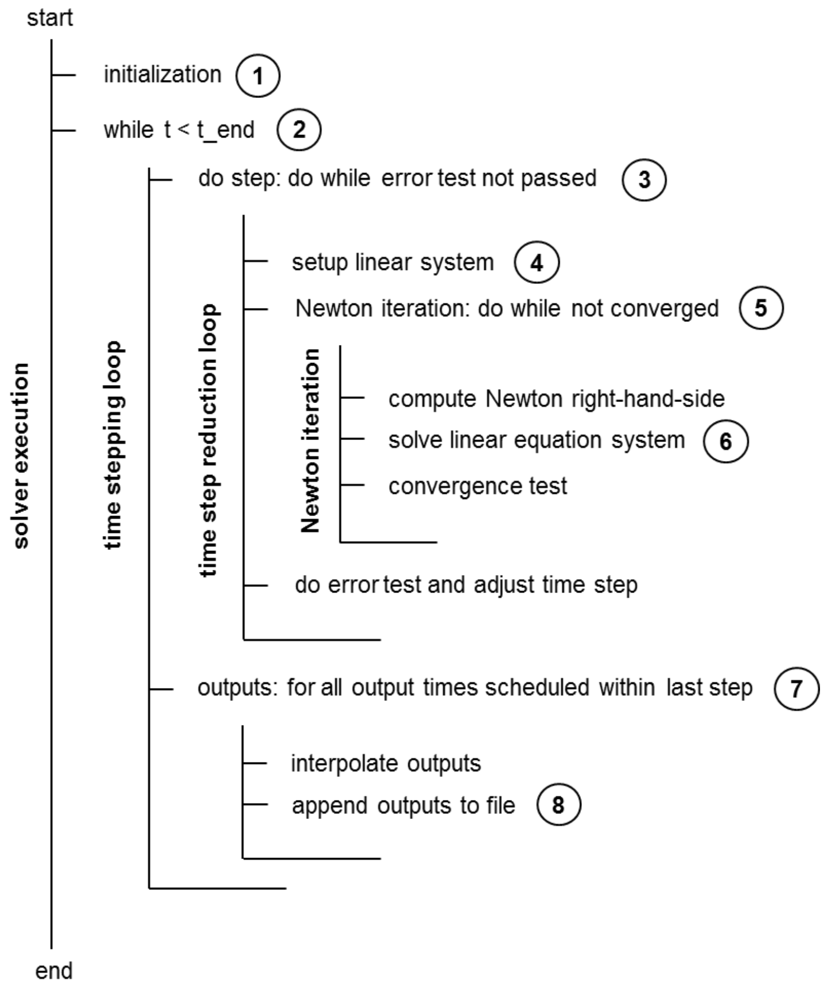


Single-zone hygrothermal simulation model

Using the IBK Integration Framework (also used by DELPHIN 6):

Variable time-step, error controlled time integration with preconditioned Newton-Krylov-iteration for solution of sparse linear equation systems

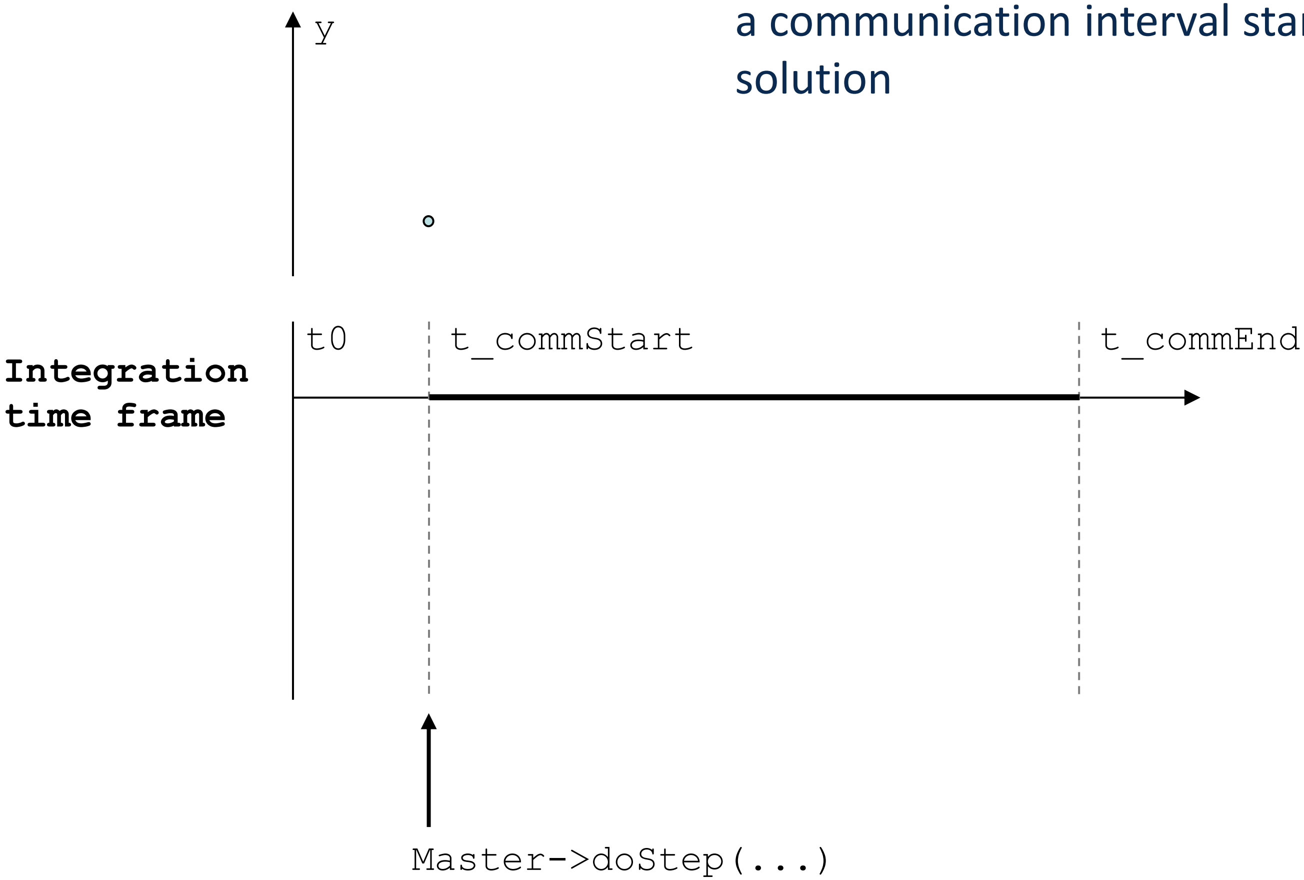
Basic functionality of a variable time step solver

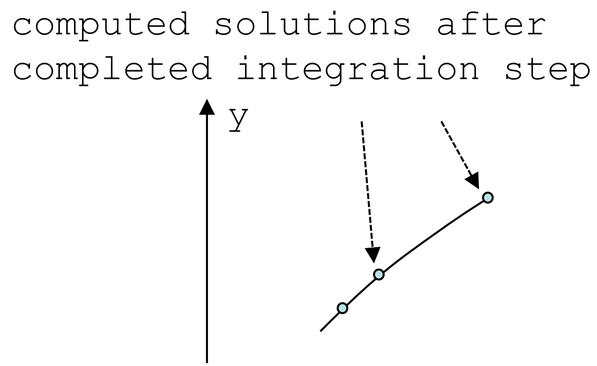


Functionality within Co-Simulation Master

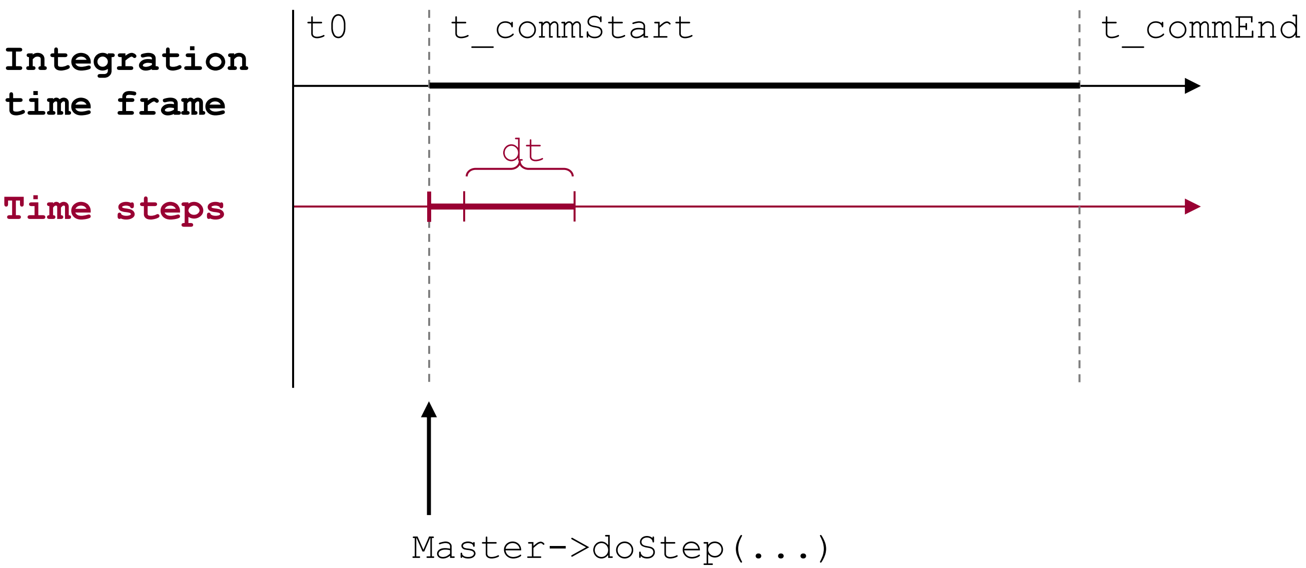
1. Model acts as a Co-Simulation slave
2. Interacts with Master through get/set functions
3. Time integration is done within a communication step when Master calls doStep()

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

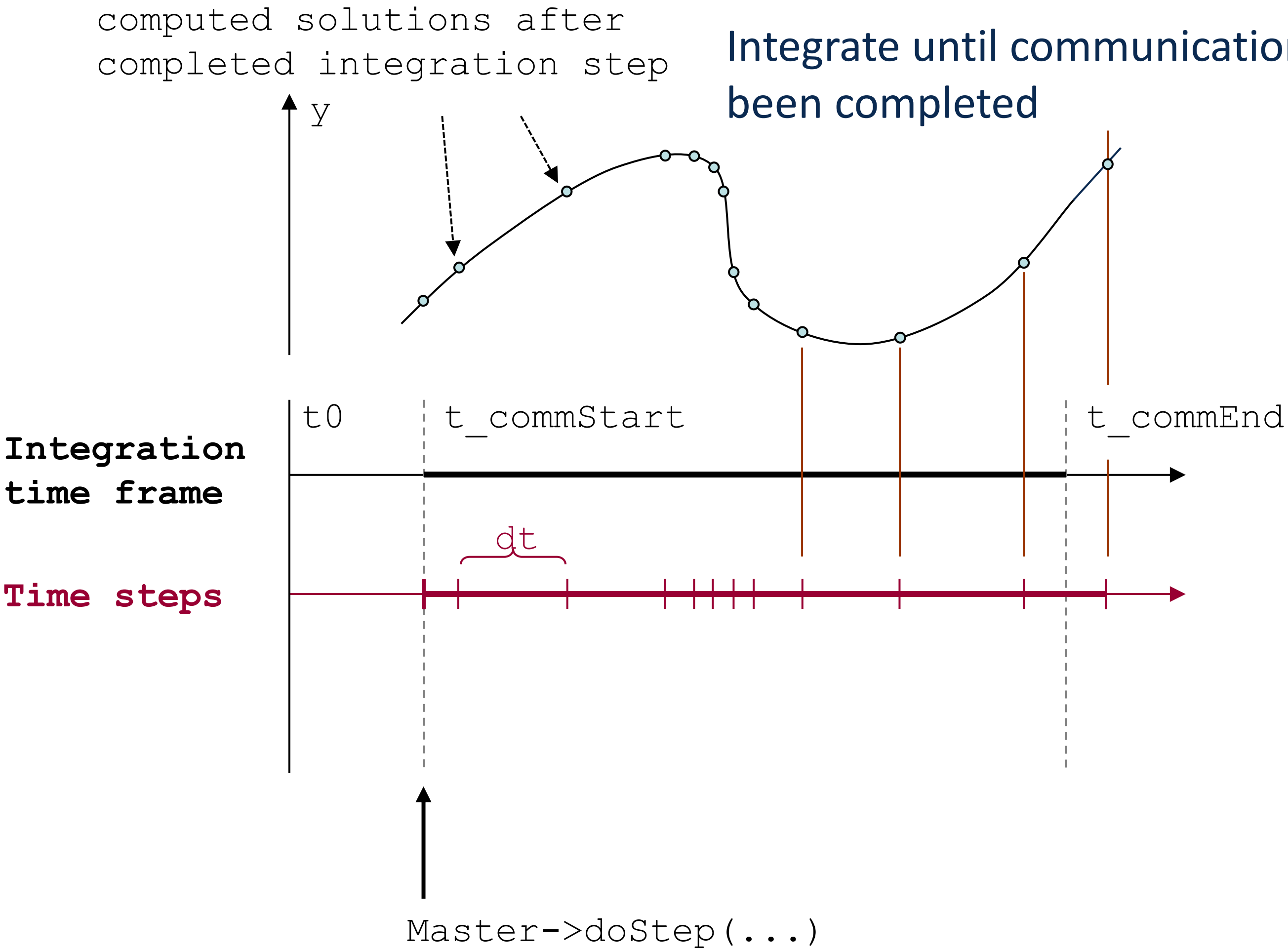




Integrate step-by-step and adapt time step size based on local error control

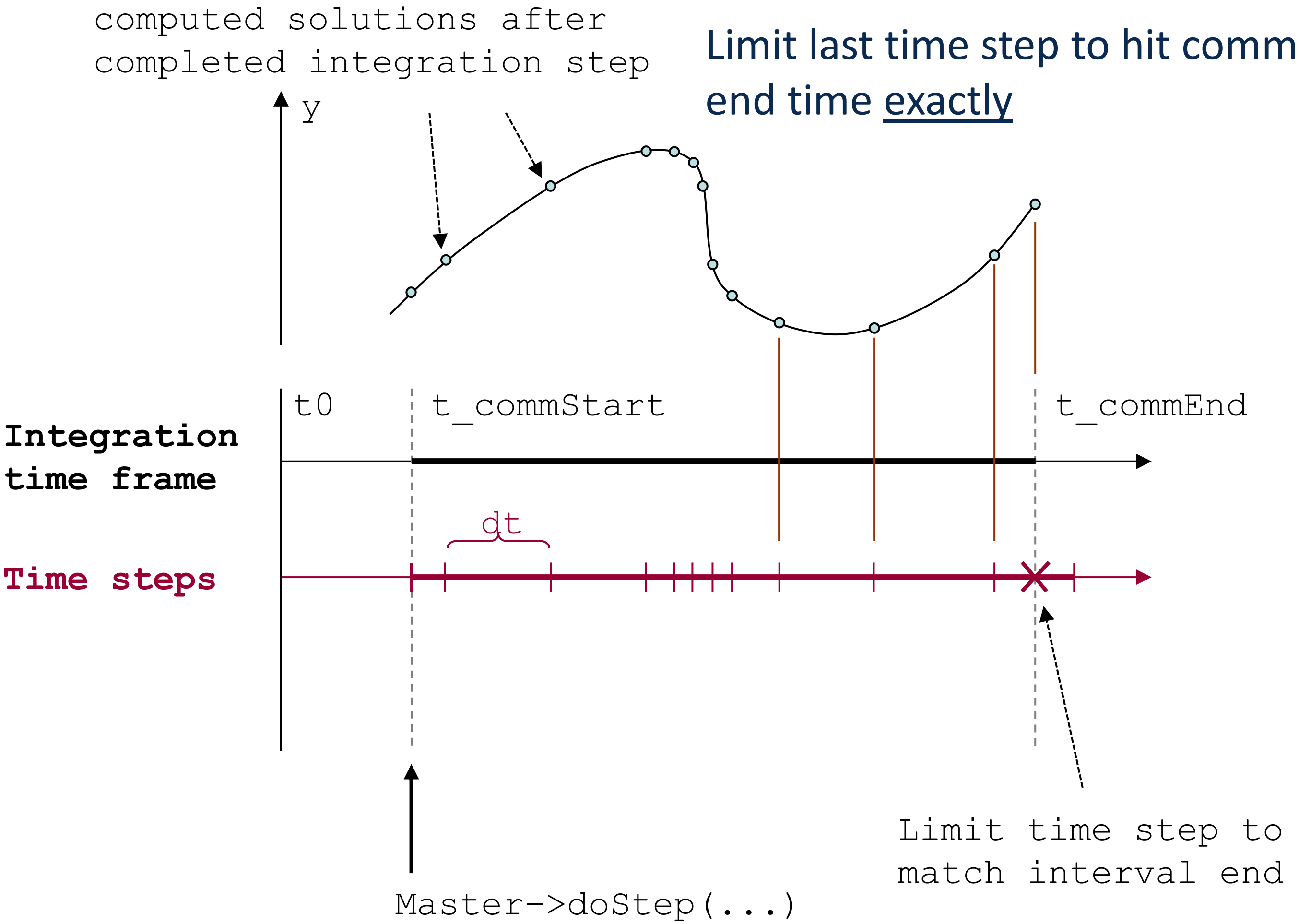


Integrate until communication interval has been completed

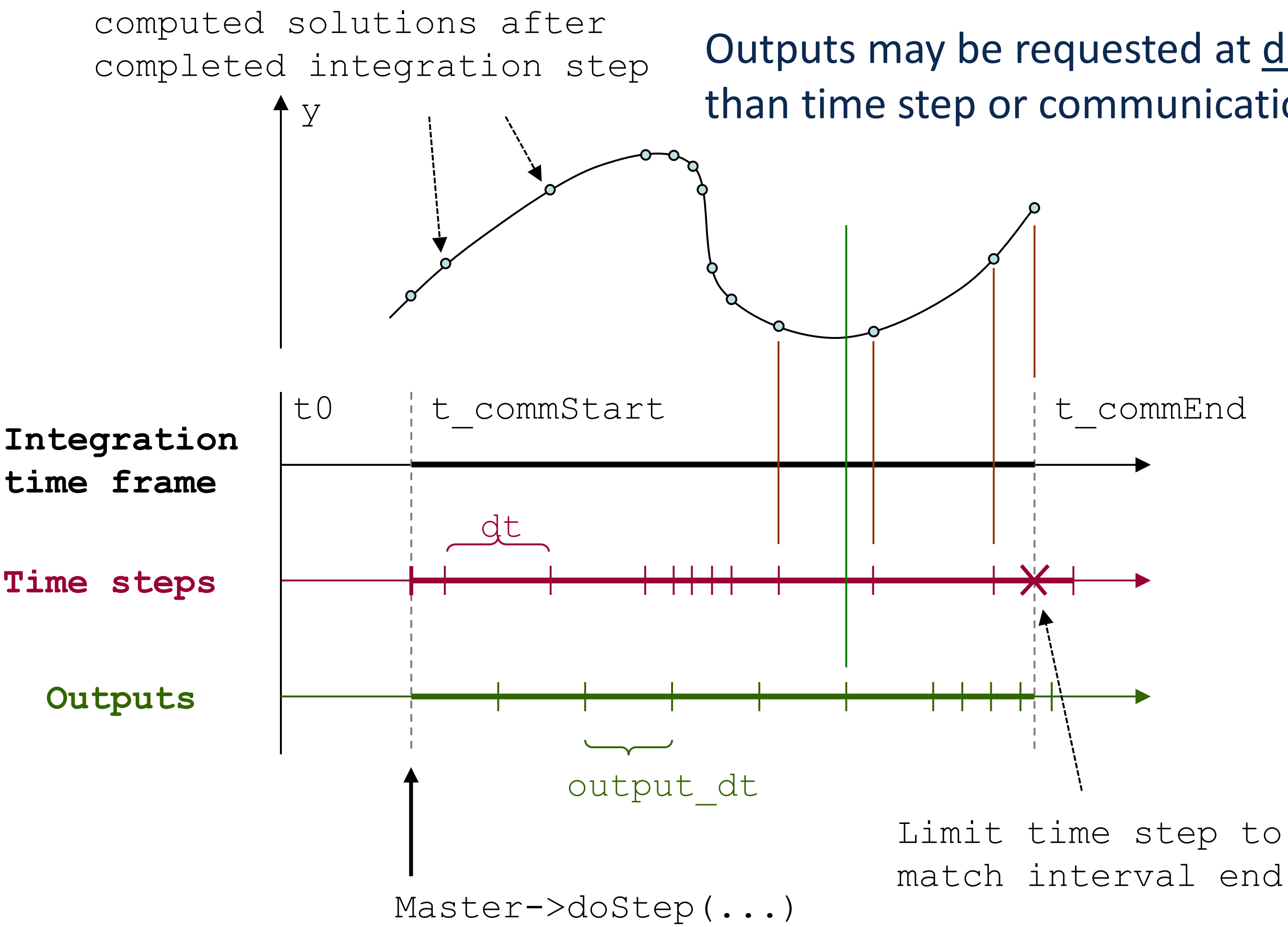


TECHNISCHE UNIVERSITÄT DRESDEN

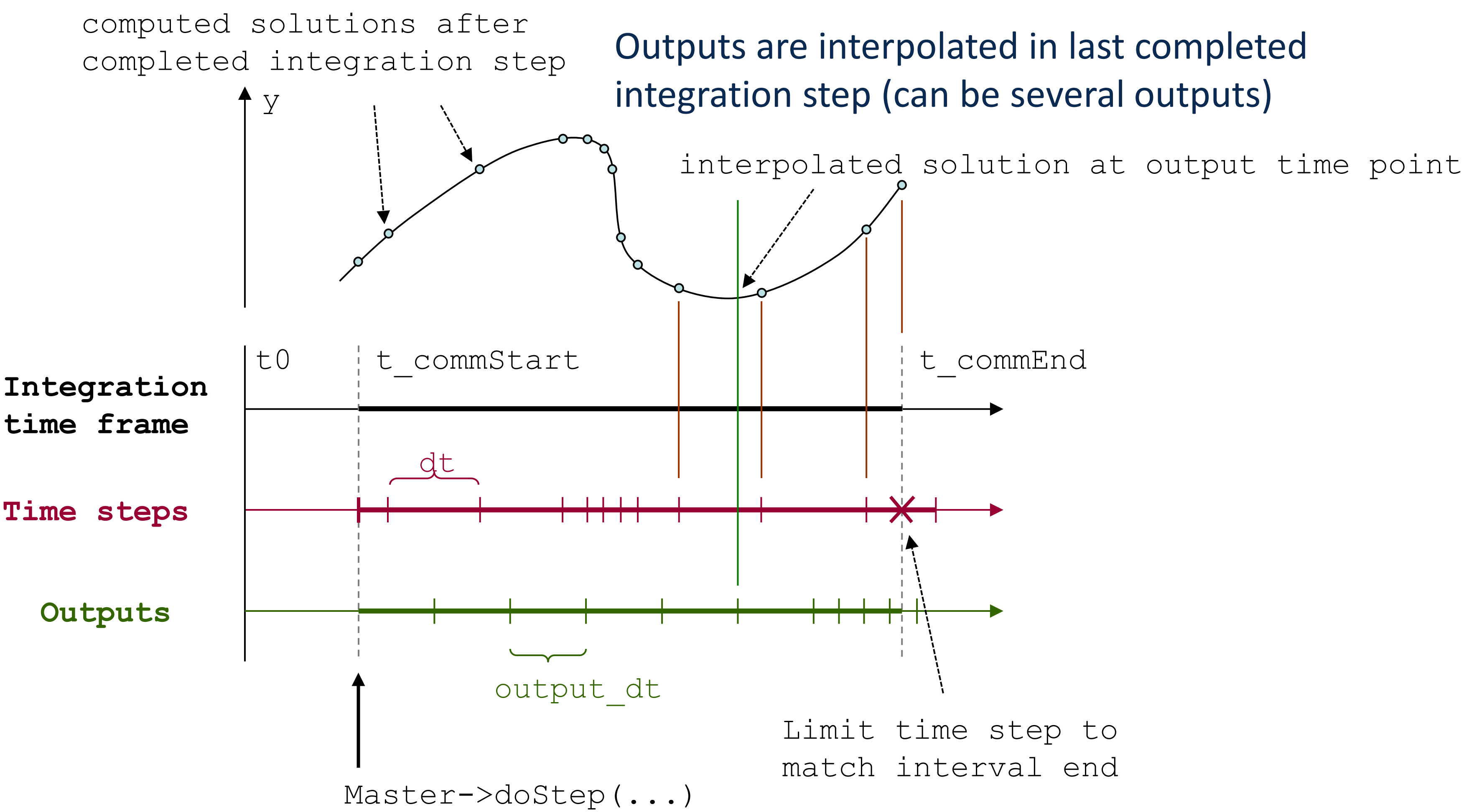
Limit last time step to hit communication interval end time <u>exactly</u>



Outputs may be requested at <u>different</u> 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 …

TECHNISCHE UNIVERSITÄT DRESDEN

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

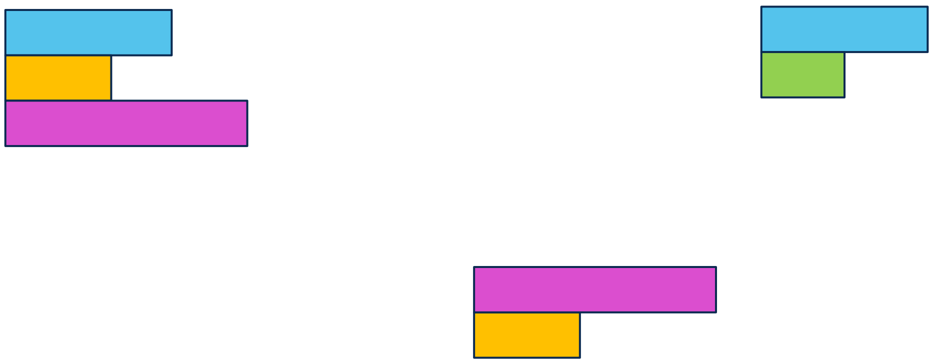
Getting FMU State via Serialization

Typical implementation of fmi2GetFMUState():

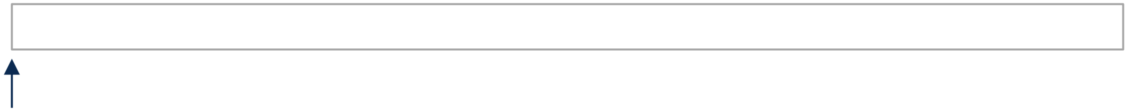
```
// check if new alloc is needed
if (*FMUstate == NULL) {
    // alloc new memory
    fmi2FMUstate fmuMem = malloc(modelInstance->m_fmuStateSize);
    // remember this memory array
    modelInstance->m_fmuStates.insert(fmuMem);
    // store size of memory in first 8 bytes of fmu memory
    *(size_t*)(fmuMem) = 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



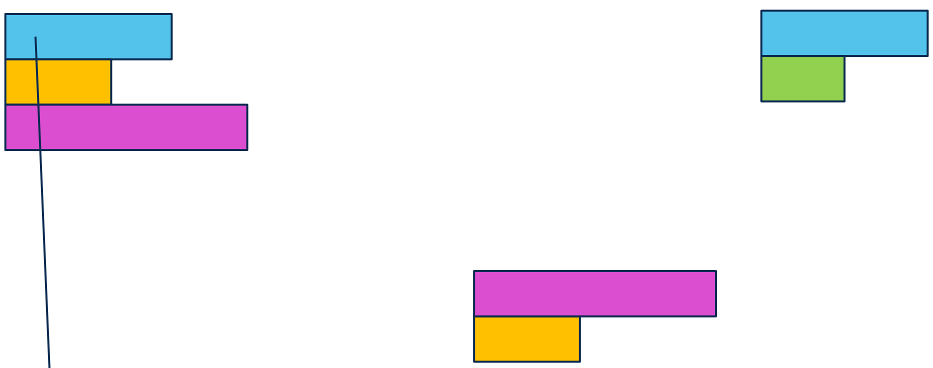
Sequential memory



"Append"-pointer

Serialization into Sequential Memory

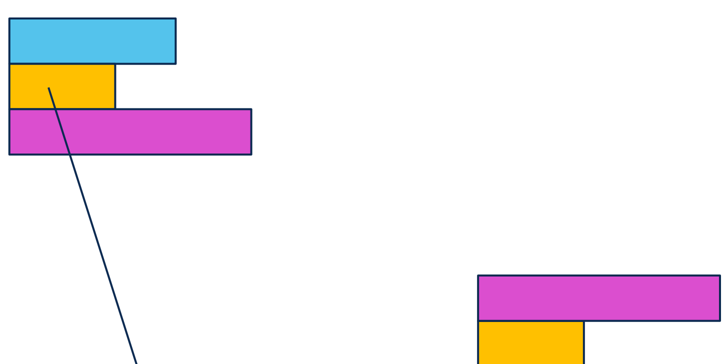
Fragmented memory structures



Sequential memory

Serialization into Sequential Memory

Fragmented memory structures

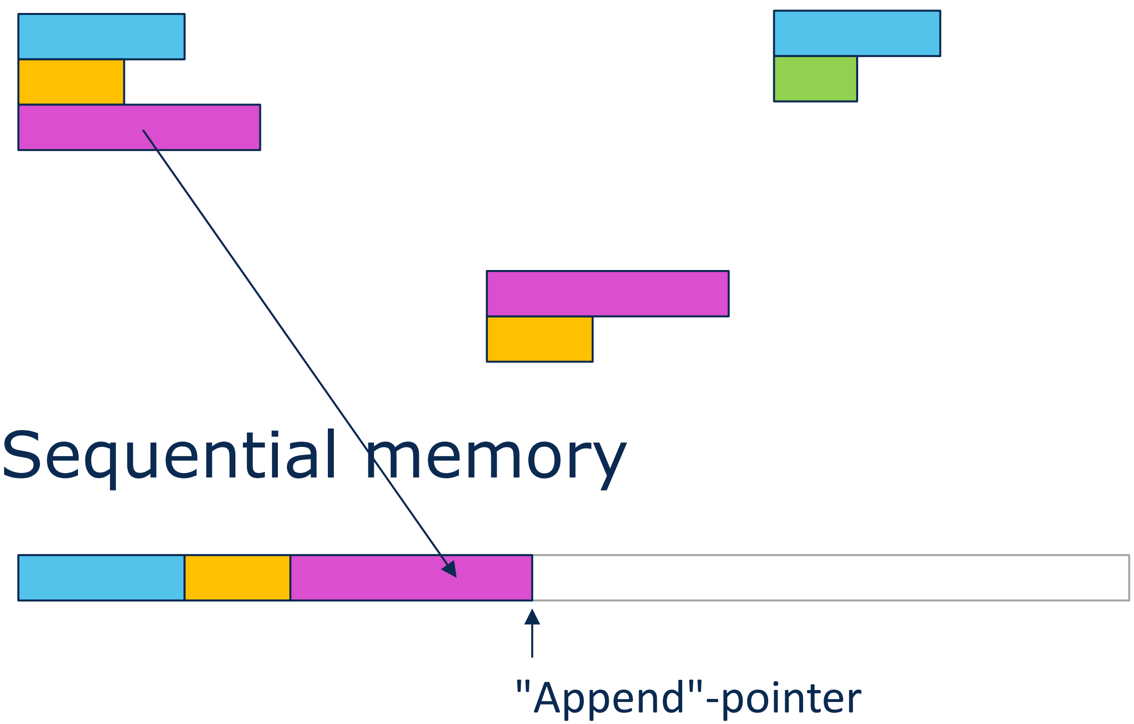


Sequential memory

Serialization into Sequential Memory

Fragmented memory structures

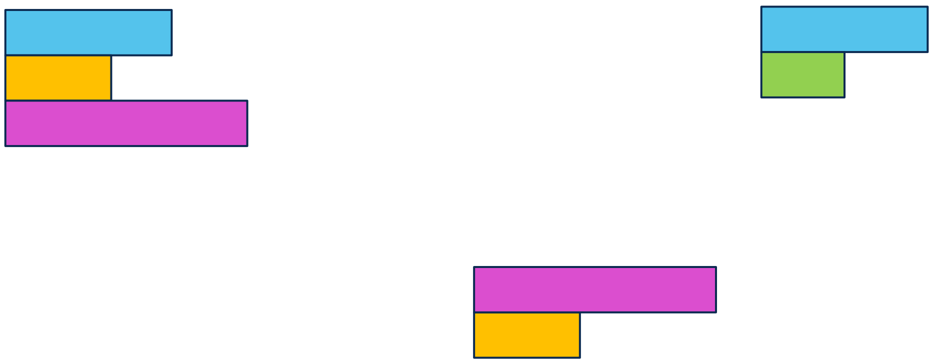
Sequential memory



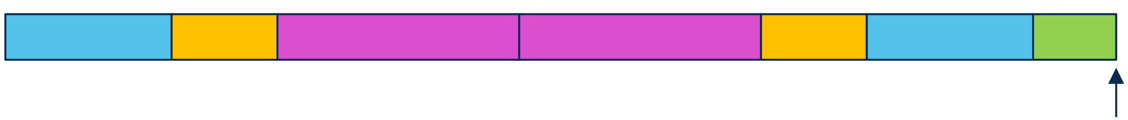
TECHNISCHE UNIVERSITÄT DRESDEN

Serialization into Sequential Memory

Fragmented memory structures



Sequential memory



"Append"-pointer

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

```c
#define SERIALIZE(type, storageDataPtr, value)\
{\
  *(type *)(storageDataPtr) = (value);\
  (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
}

#define DESERIALIZE(type, storageDataPtr, value)\
{\
  (value) = *(type *)(storageDataPtr);\
  (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
}
```

Serialization into Sequential Memory

```
#define SERIALIZE(type, storageDataPtr, value)\
{\
  *(type *)(storageDataPtr) = (value);\
  (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
}

#define DESERIALIZE(type, storageDataPtr, value)\
{\
  (value) = *(type *)(storageDataPtr);\
  (storageDataPtr) = (char *)(storageDataPtr) + sizeof(type);\
}
```

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

  switch (op) {\
  case SUNDIALS_SERIALIZATION_OPERATION_SERIALIZE:\
    SERIALIZE(type, storageDataPtr, value)\
    break;\
  case SUNDIALS_SERIALIZATION_OPERATION_DESERIALIZE:\
    DESERIALIZE(type, storageDataPtr, value)\
    break;\
  case SUNDIALS_SERIALIZATION_OPERATION_SIZE:\
    mem_size += sizeof(type);\
    break;\
  }
```

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)
```

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)
```

Open-Source Patch for SUNDIALS release

Output file handling and iterating over FMUs

- FMI for CoSimulation v2 allows iterations
- Complex models may write many output files (<u>impossible</u> to pass all data as output)

Need to decide when to write outputs!

Best practice: whenever master signals in doStep() that *no resetting is done* prior to communication start:

noSetFMUStatePriorToCurrentPoint == fmi2True

Output file handling and iterating over FMUs

- FMI for CoSimulation v2 allows iterations
- Complex models may write many output files (impossible to pass all data as output)

Need to decide when to write outputs!

Alternative method: write binary output files and overwrite records in repeated communication intervals (may be less coding effort)

Issues when restoring FMU states

- FMI for CoSimulation v2 allows deserialization
- Simulation may be continued at stored state

What about output files written so far?

Good case/error handling needed:

- **Re-open files and continue writing (nominal case)**
- What if simulation is continued sometime in between a completed simulation → truncate output files?
- What if output files are missing? What if gaps are in output files? …

Summary – or is there an easier path to an FMU?

- No iterating master algorithms → no get/set FMU state needed
- No simulation restarting → no serialization functionality needed
- Communication intervals (~ 1 min) always shorter than model output steps (~ 30 min.. 1h) → no output backsetting needed
- Output steps even multiple of constant communication steps → no output interpolation needed

⇒ All of that: → EnergyPlus-FMU and similar

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Architektur, Institut für Bauklimatik, Professur für Bauphysik

Transformation of a building energy simulation tool with variable time step solver into an FMU for CoSimulation v.2 with serialization capabilities

Andreas Nicolai and Anne Paepcke

andreas.nicolai@tu-dresden.de

Dresden, 16.09.2016