INTRODUCTION TO MODELICA

External Code

Modelon

OVERVIEW

- External Functions
- External Objects

EXTERNAL CODE

EXTERNAL FUNCTIONS

EXTERNAL FUNCTIONS

- An external function is a function that uses non Modelica code that is defined in a external file, e.g. a C-file.
- The Modelica external function call interface provides:
  - Support for external functions written in C or FORTRAN
  - Mapping of argument types from Modelica to the target language and back.

- External functions are used in the Modelica Standard Library
  - Example: *Modelica.Math.Matrices.LU*

Very useful interface if you already have a large code base

EXAMPLE – EXTERNAL FUNCTIONS

- Example: Using a polynomial multiply function implemented in C
  - Modelica wrapper to a C-file:

```
function polynomialMultiply
  input  Real a[:];
  input  Real b[:];
  output Real c[:] = zeros(size(a,1)+size(b, 1) - 1);
  external "C" polmult(b, a, c, size(a,1), size(b,1));
end polynomialMultiply;
```

- Assumes following C-function:

```
void (polmult)(double  const *,
           double  const *,double  *, int, int);
```

EXTERNAL CODE

- External functions are included as C functions compiled with the model code or binary library which is linked to the model.
- Annotations are used to specify code includes or header and library names:
  - *annotation(Include="#include <add2.c>");*
    - Code can be located in current directory, relative location or in $DYMOLA\Source.
  - *annotation(Include="#include <add2.h>", Library="ext");*
    - Library prefix is added by the linker – depends on the used compiler.

EXTERNAL C-CODE

- Annotation appended External ”C” definition

```
function powerFunction
  input  Real value;
  input  Integer p;
  output Real y;
  external "C" y =power(value,p)
annotation (
IncludeDirectory="modelica://ExternalCode/Resources/",
Include="#include <power.c>");
end powerFunction;
```

```
double power(double val, int pow)
{       double ret_val = 1.0;
        int i;
        for(i = 0; i < pow; i++)
                ret_val *= val;
        return(ret_val);
}
```

EFFICIENT CODE

- Numerical solvers are more robust and faster, when symbolic derivatives are available.
- For external code, if gradients can be computed, Modelica derivative annotations for the wrapper functions can be supplied to point to the external gradient functions.
- Our experience shows that it is worth the additional effort, high-quality implementations of linking to external code should have derivatives.
- If external functions do not behave like pure mathematical functions, i.e. a set of inputs always generates the same outputs (no state, no memory), the solver will hang or give unpredictable results.

EXTERNAL CODE

EXTERNAL OBJECTS

EXTERNAL OBJECTS

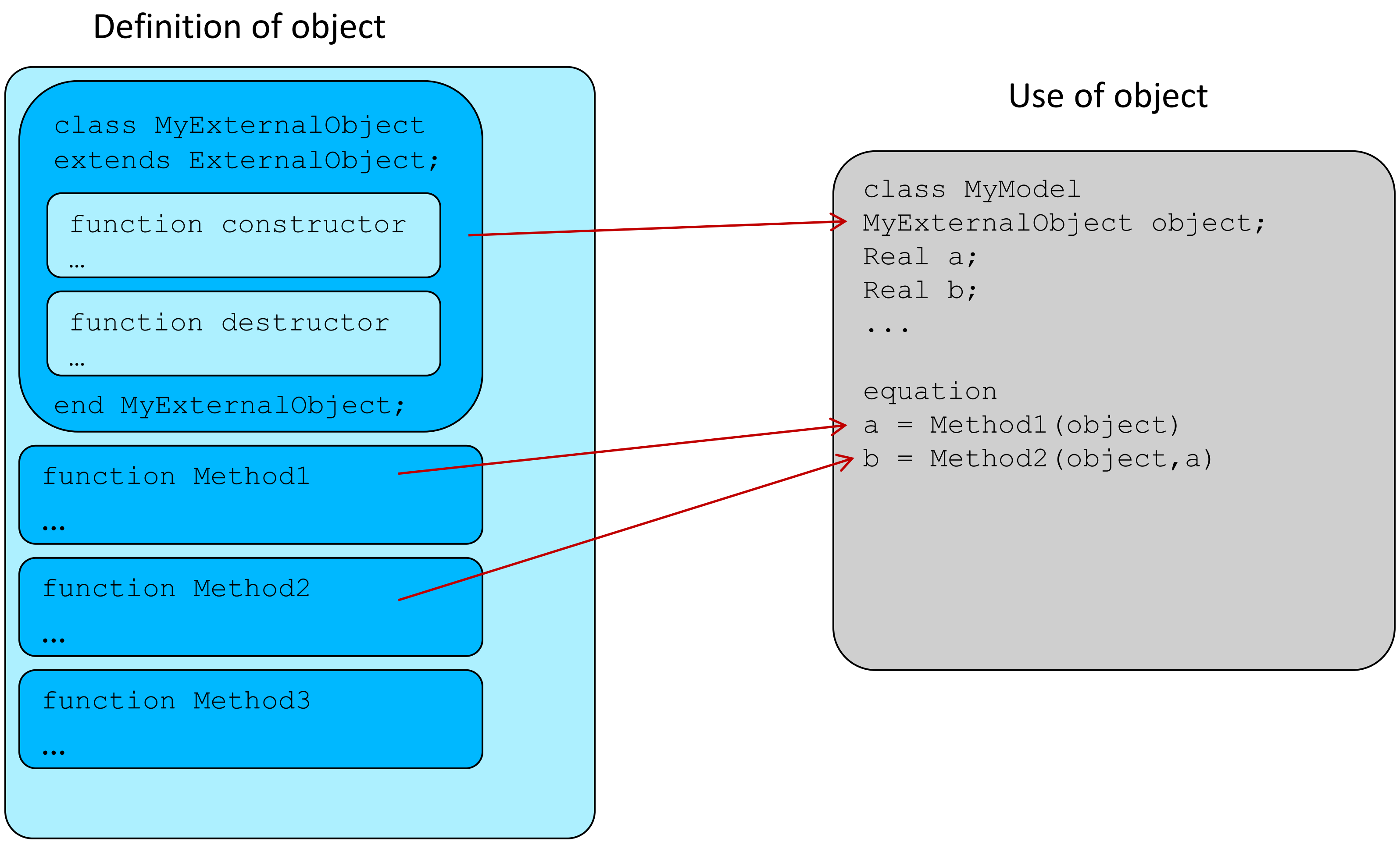
- External Functions may not have memory or internal states
- Often more efficient if external code has internal state (e.g. large table interpolations), even though function acts as if it had no internal states (side effects)
- Many couplings to external code require state/memory in external code (e.g. real controller code)
- Simple forms of co-simulation possible

EXTERNAL OBJECTS

- There is a predefined partial class "*ExternalObject*"
    - An external object class must be extended from "*ExternalObject*" and contain two function definitions, called "constructor" and "*destructor*", and shall not contain other elements.

- Modelon Impact automatically handles the construction and deconstruction of the objects, although the user needs to define the functions accordingly:
    - The constructor shall have one output argument in which the constructed ExternalObject is returned.
    - The destructor shall have only one input argument, ExternalObject.
    - It is not legal to call explicitly the constructor and destructor functions.

DEFINING AN EXTERNAL OBJECT



DEFINING AN EXTERNAL OBJECT

```
class MyExternalObject
extends ExternalObject;

  function constructor
  output MyExternalObject obj;
  external "C" obj = MyObject_init();

  function destructor
  …

end MyExternalObject;
```

```
function Method1
output Real a;
external "C" a = MyObject_Method1();
```

```
function Method2
output Real b;
external "C" b = MyObject_Method2();
```

```
MyExternalCode.c

void MyObject_init()
...

void MyObject_Method1();
...

void MyObject_Method2();
...
```

WORKSHOP 4.4

In this workshop you will:

- Implement a small C function
- Create a modelica wrapper function
- Execute a model using the external c-code