# Introduction to SystemC-AMS Library Prototype

Thomas Uhle, Karsten Einwich

Fraunhofer Institute for Integrated Circuits IIS
Design Automation Division Dresden, Germany

Fraunhofer Institut Integrierte Schaltungen

# Outline

- Focus of SystemC-AMS
- Why is having different Models of Computation cute?
- SystemC and its extension SystemC-AMS
  - Common Use Flow
  - Short overview to SystemC's capabilities
  - Concepts and implementation of SystemC-AMS
- Models of Computation again
  - Synchronous / Static Dataflow
  - Linear Networks
- What's left?
  - Non-linear Networks, etc.

# Focus of SystemC-AMS

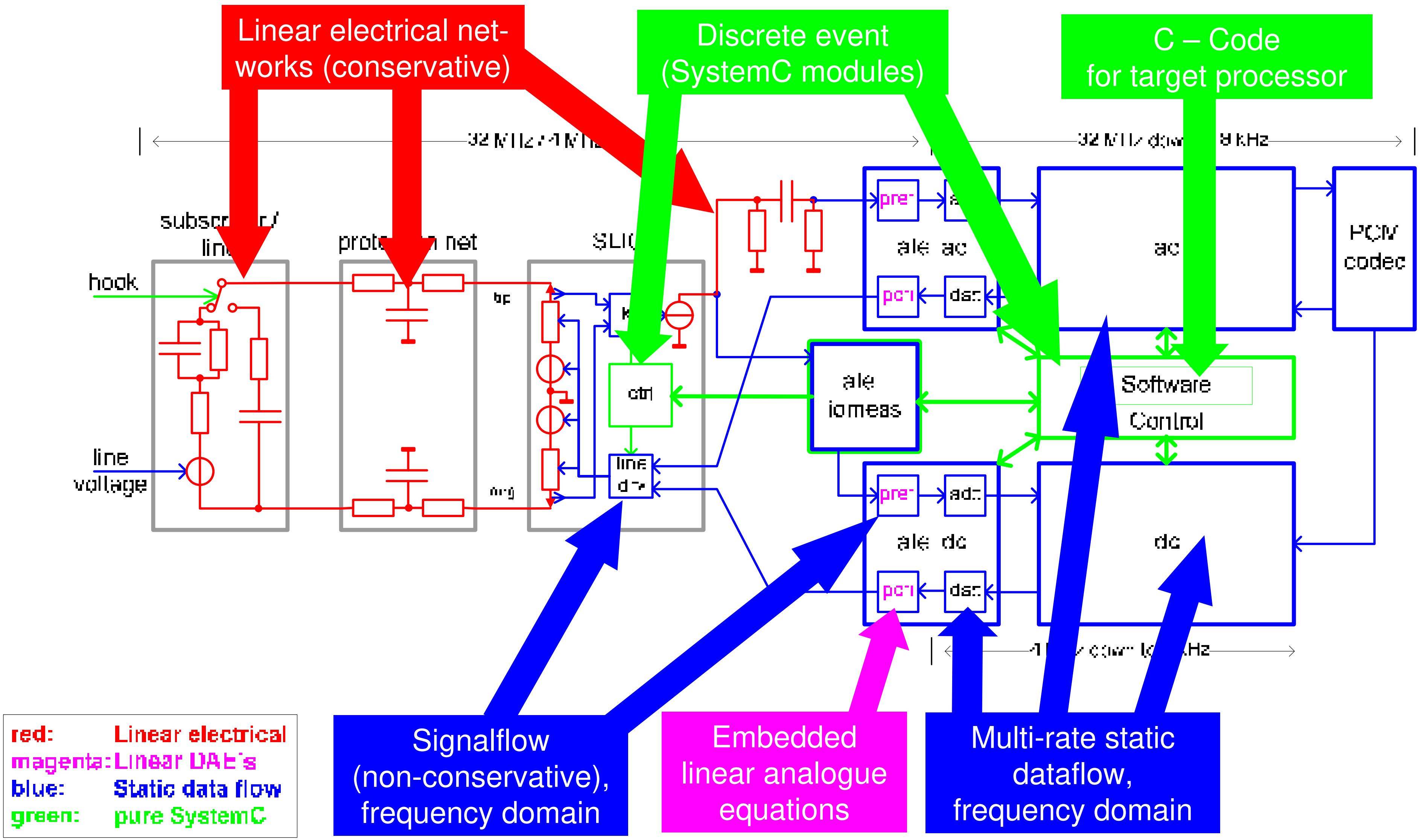
**Description, Simulation and Verification for:**

- Functional **complex** integrated systems
- <u>A</u>nalogue <u>M</u>ixed-<u>S</u>ignal systems / **Heterogeneous** systems
- **Specification / Concept and System Engineering**
- **System design**, development of a ("golden") **reference model**
- Embedded **Software** development
- Next Layer (Driver) Software development
- **Customer model, IP protection**

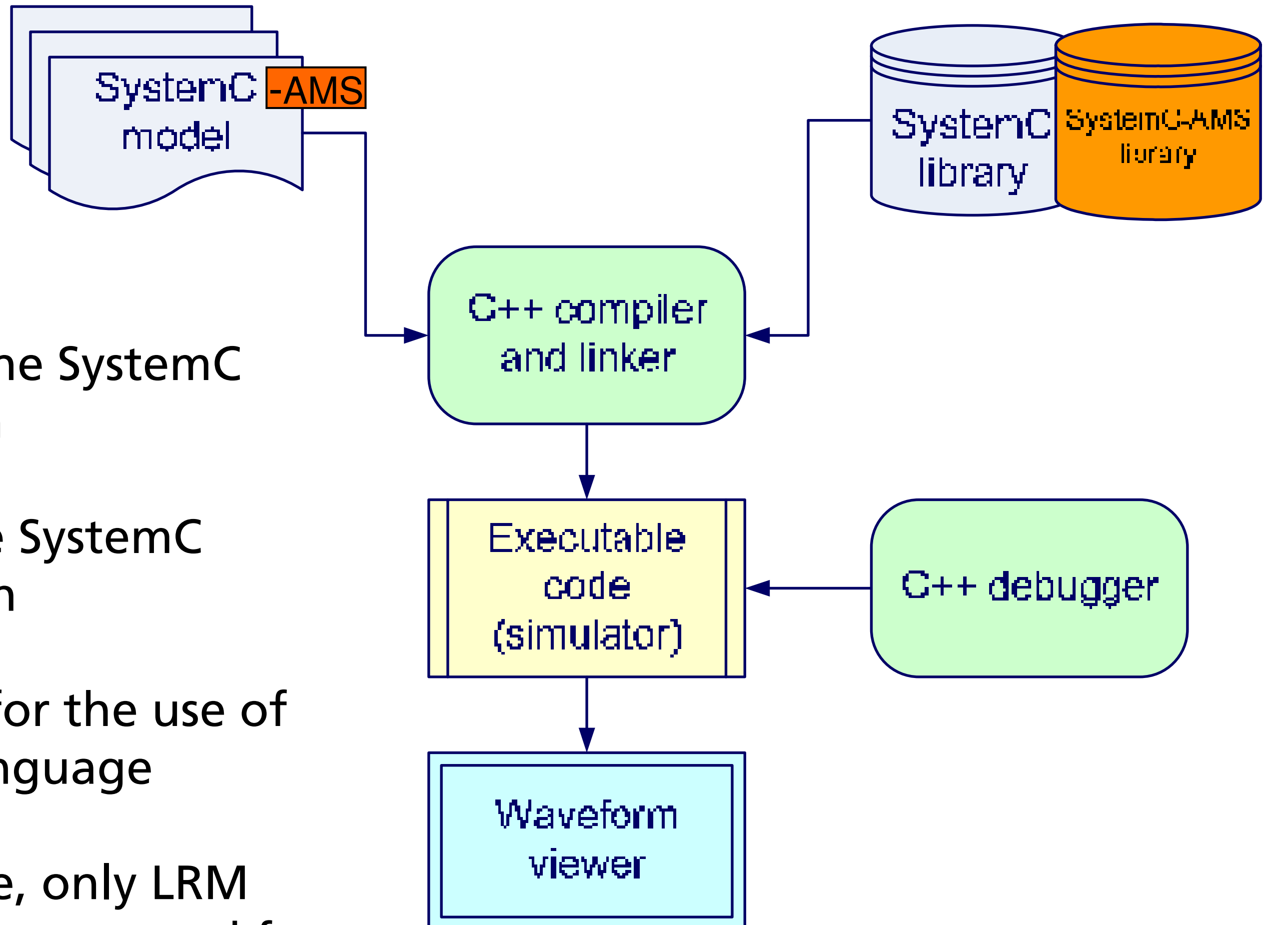
# Why having different analogue Models of Computation?

- Modelling on different abstraction / accuracy levels yields the possibility to apply specialised algorithms, which are **orders of magnitude faster** than a general approach.
- It is possible to **reduce the solvability problem** significantly.
- Due to the encapsulation of analogue MoC / solvers SystemC-AMS models are **very well scalable** – very large models can be handled.
- Examples for specialised analogue Models of Computations (MoC):
  - Linear Networks / Differential-Algebraic Equation (DAE) systems
  - Non-linear Networks / DAE systems
  - Switched Capacitor Networks (leads to simple algebraic equation)
  - Dataflow solver for Signalflow Descriptions and Bond Graphs
  - …

# Application of SystemC-AMS to a Voice Codec System



# SystemC-AMS is an extension of SystemC



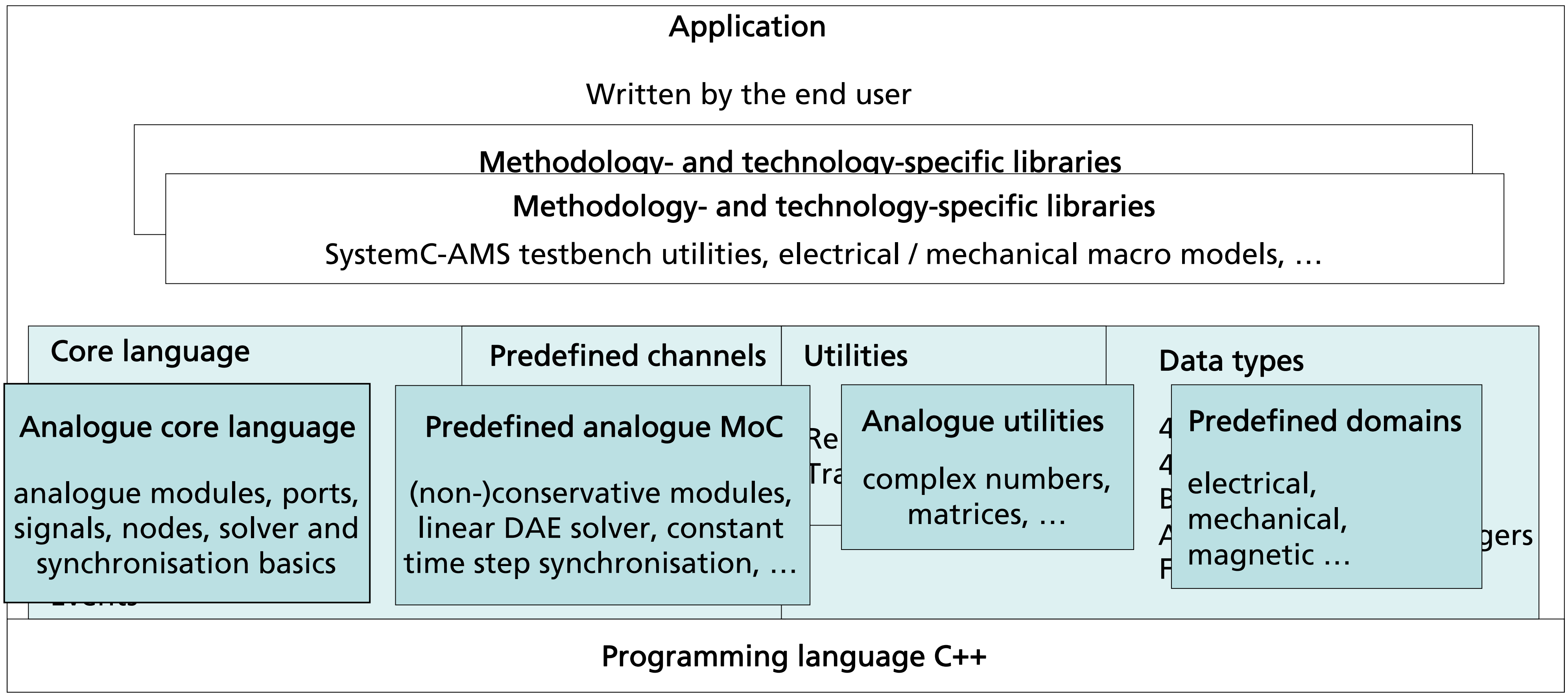

- no changes to the SystemC implementation

➡ use of the same SystemC implementation

➡ no restrictions for the use of the SystemC language

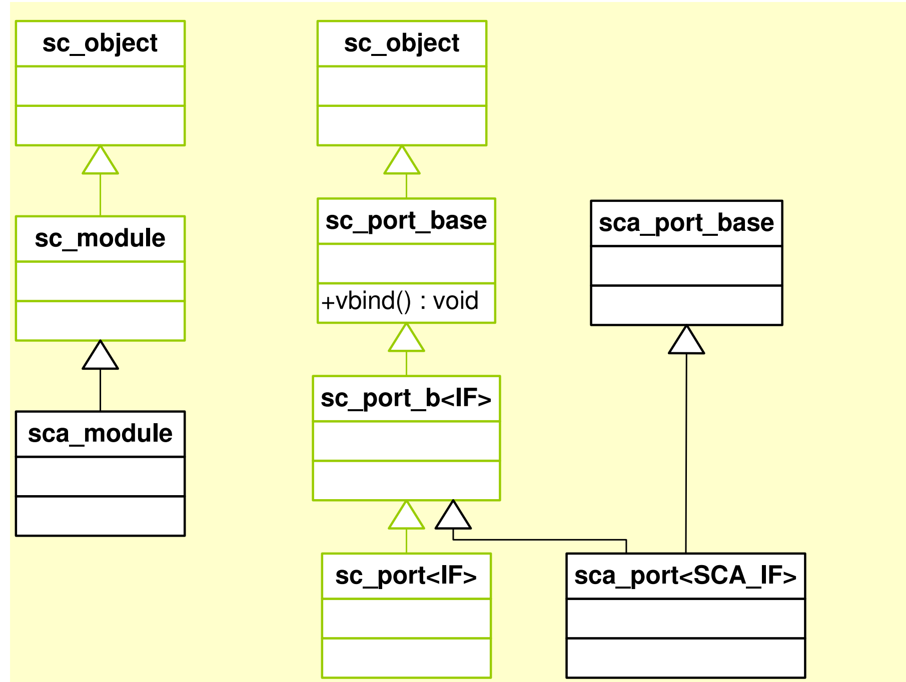
- as far as possible, only LRM documented features used for the library implementation

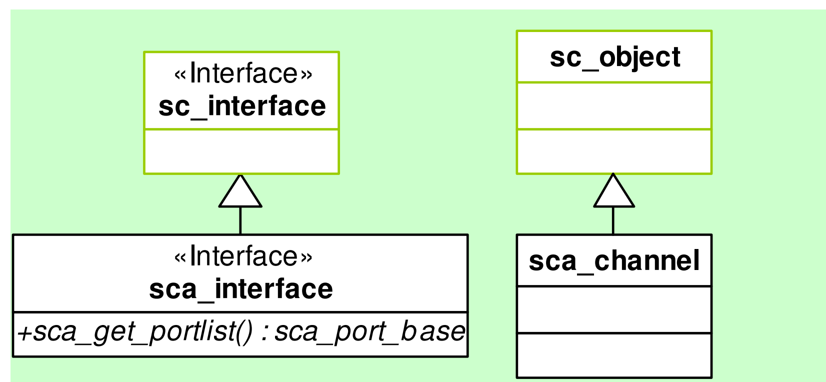
# SystemC / SystemC-AMS language architecture



# SystemC-AMS Implementation

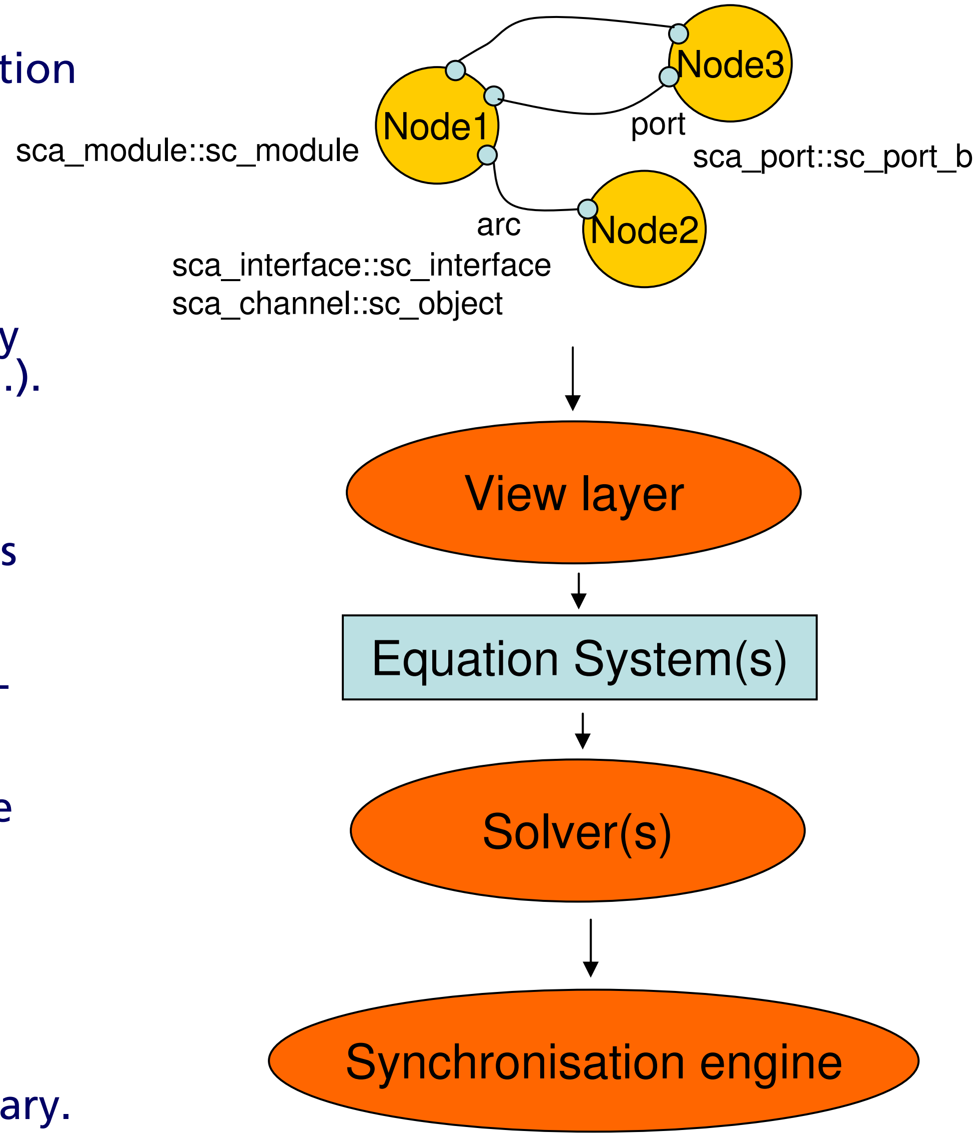
- Analogue Module
  - container class for analogue ports and **primitive** behaviour
- Analogue Port
  - provides access to a connected interface, channel
- Analogue Interface
  - provides access routines
- Analogue Channel
  - implements access routines



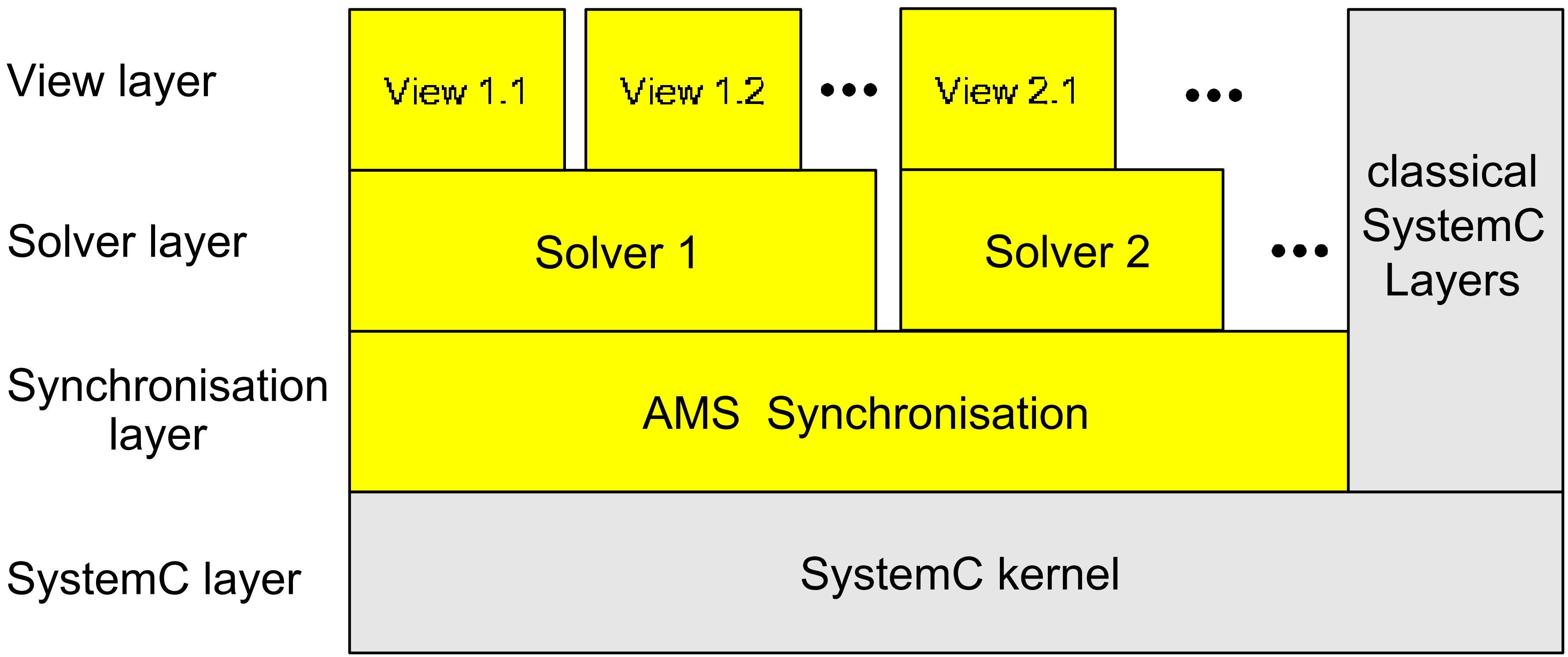



# SystemC-AMS Principal Implementation Concepts

- abstract syntax for structural description
- A couple of commonly required operations can be performed on the abstract graph (description, hierarchy flatting, clustering, graph analysis, …).
- All nodes connected together are assigned to a concrete view which setups the equations and instantiates one or more solvers.
- A solver contains the concrete implementation of the solver's algorithm which implements an abstract solver interface which in turn is used by the view layer to setup the equations.
  A solver is as well an abstract object (with a defined interface) for the synchronisation layer.
- This way only one synchronisation engine to the DE – SystemC is necessary.



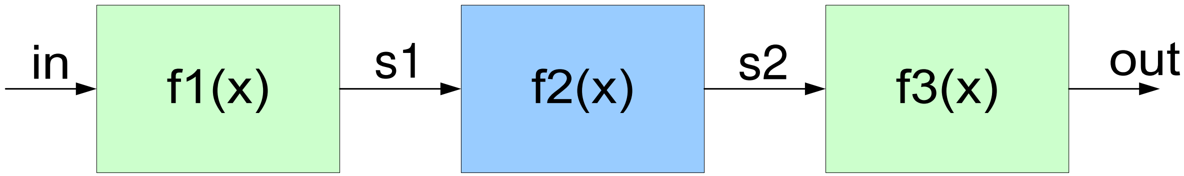
# Concept of SystemC-AMS



# Outline

- Focus of SystemC-AMS
- Why is having different Models of Computation cute?
- SystemC and its extension SystemC-AMS
  - Common Use Flow
  - Short overview to SystemC's capabilities
  - Concepts and implementation of SystemC-AMS
- **Models of Computation again**
  - Synchronous / Static Dataflow
  - Linear Networks
- What's left?
  - Non-linear Networks, etc.

# Dataflow MoC: Modelling non-conservative behaviour



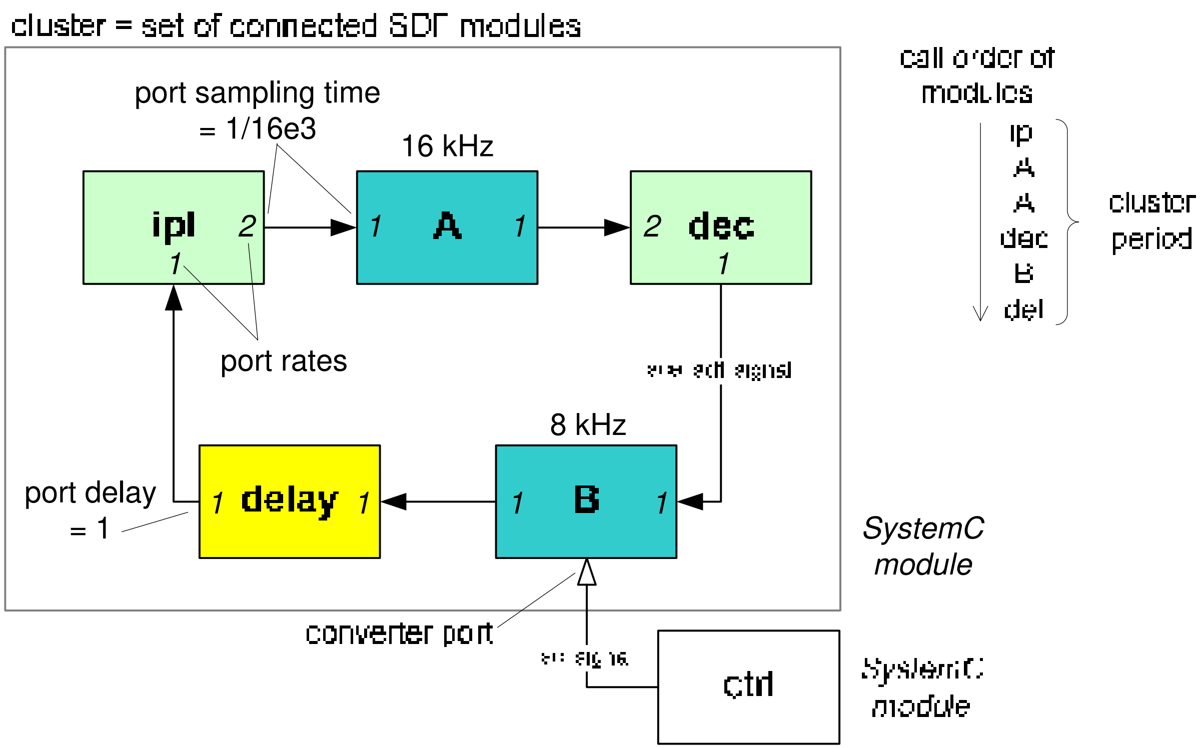

out = f3( f2( f1(in) ) )

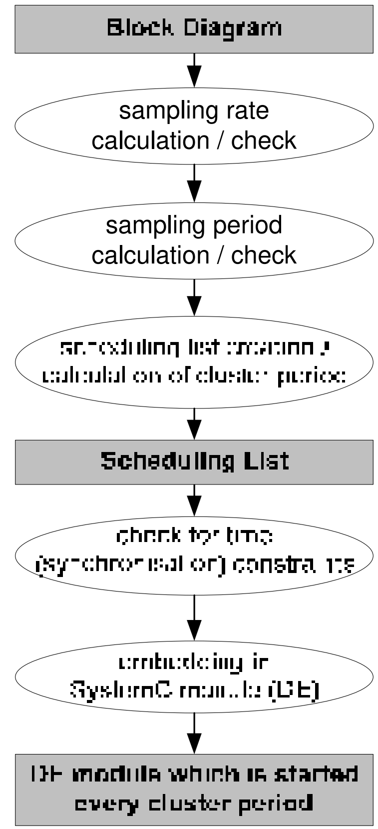
<u>equation system:</u>

s1 = f1(in)
s2 = f2(s1)
out = f3(s2)

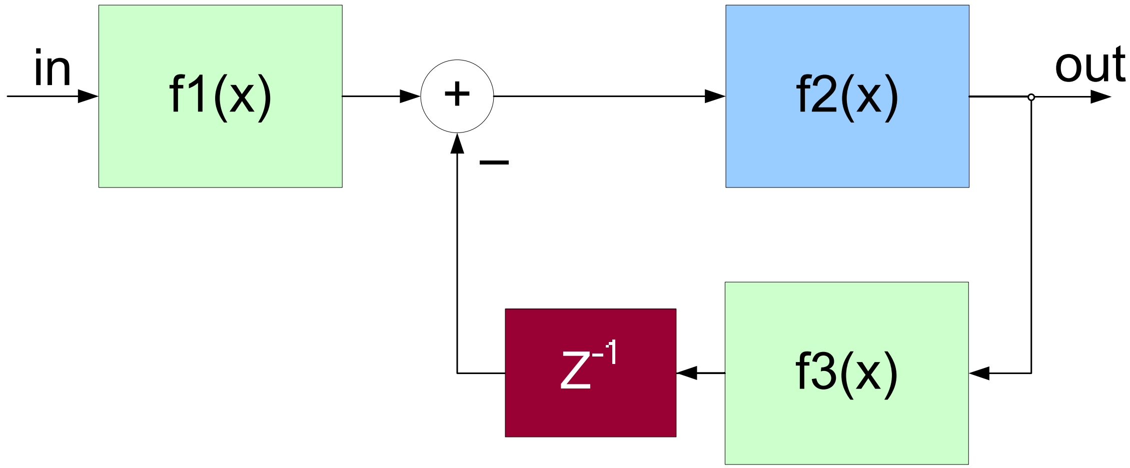
- Simple firing rule: A block is called if enough samples are available at its input ports.
- The function of a block is performed by
    1. reading from the input ports (thus consuming samples),
    2. processing the calculations and
    3. writing the results to the output ports.
- For **synchronous dataflow** (SDF) the numbers of read / written samples are **constant** for each block call.
- The scheduling order follows the signalflow direction.
- One drawback is the need of having the equations in an **explicit formulation**. Thus, only explicit DAE systems can be described by means of the SDF.

# Implementation of Multi-Rate SDF in SystemC-AMS



# Loops in Synchronous / Static Dataflow Clusters

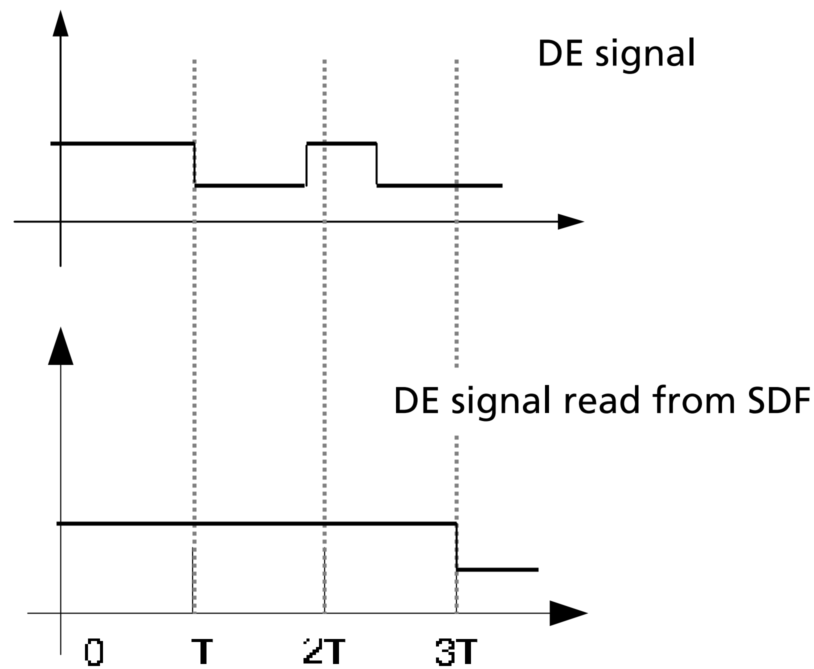



- Simulating signalflow behaviour by synchronous dataflow MoC with algebraic loops is **not possible.**
- Thus, **at least one delay** in the loop is crucial!
- For analogue modelling the delay is a "hopefully" acceptable approximation:

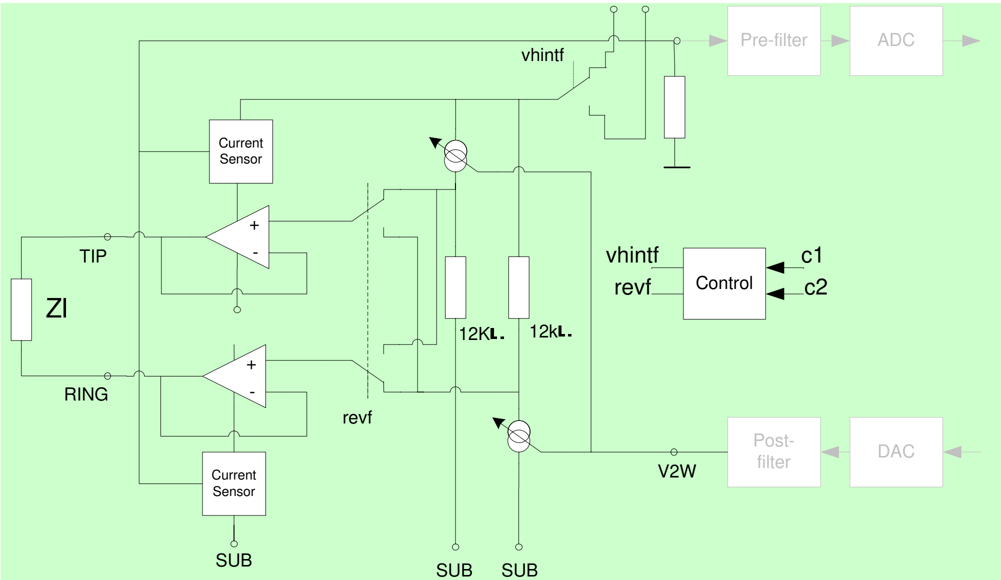
$$out = f2(\ f1(in) - f3(out)\ ) \longrightarrow out = f2(\ f1(in) - f3(out)\ z^{-1}\ )$$

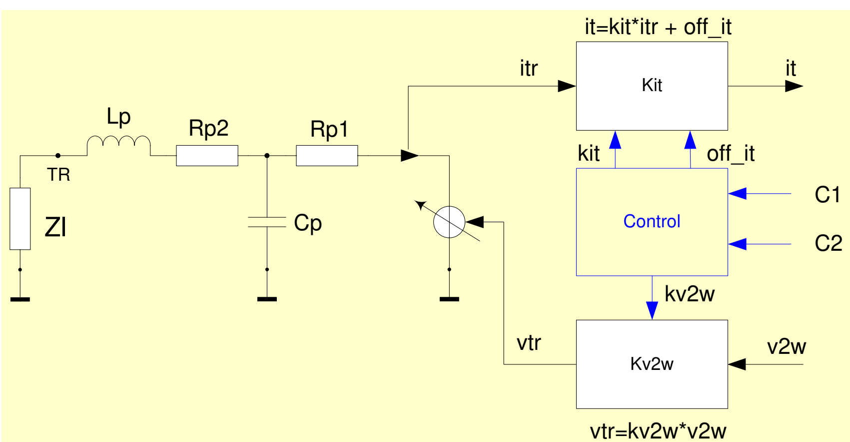
# Synchronisation between SDF and DE Domain

- SDF samples are mapped to sc_time.
- SystemC (DE) signals are sampled at $\Delta$=0 of the specified sampling period. SDF samples are scheduled at $\Delta$=0 as well (and thus valid at least at $\Delta$=1).
- The sampling period T is specified as port attribute and propagated along the SDF signals of the cluster.
- That is why the sampling period must be specified at least for one port of a module in every SDF cluster – are ≥ 2 sampling periods given, the simulator performs a consistency check.



# Static Dataflow Modules (non-conservative Modules)



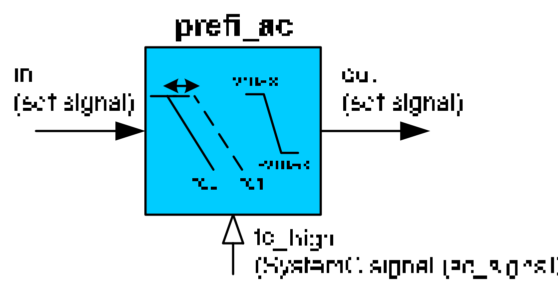

```
SCA_SDF_MODULE(kv2w)
{
    sca_sdf_in<double>    v2w;
    sca_sdf_out<double> vtr;

    // control / DE inport
    sca_scsdf_in<double> k_v2w;

    void sig_proc();

    SCA_CTOR(kv2w)
    {
    }
};

void kv2w::sig_proc()
{
    double v2w_tmp = v2w.read();
    double vtr_tmp;

    vtr_tmp = k_v2w.read() * v2w_tmp;

    vtr.write(vtr_tmp);
}
```

# Static Dataflow Modules – Example with LTF

```
SCA_SDF_MODULE(prefi_ac)
{
   sca_sdf_in<double>    in;    // signal inport
   sca_sdf_out<double> out;  // signal outport

   // control / DE signal from SystemC
   // (connected to sc_signal<bool>)
   sca_scsdf_in<bool>   fc_high;

   double fc0, fc1;    // cut-off frequency
   double v_max;       // max. out value

   sca_ltf_nd  ltf_0, ltf_1;       // filter equation inst.
   sca_vector<double>  a0, a1, b;
   sca_vector<double>  s;       // state vector

   void init() // filter coeffs for transfer function
   {
      const double r2pi = M_1_PI * 0.5;
      b(0)   = 1.0;            a1(0) = a0(0) = 1.0;
      a0(1) = r2pi/fc0;      a1(1) = r2pi/fc1;
   }
```

```
   void sig_proc() {
      double tmp;     // high or low cut-off freq.
      if(fc_high.read())  tmp = ltf_1(b, a1, s, in.read());
      else                       tmp = ltf_0(b, a0, s, in.read());

      if        (tmp >  v_max)    tmp =   v_max;  // output voltage
      else if (tmp < –v_max)    tmp = –v_max;  // limitation

       out.write(tmp);  // assign output voltage to port
   }

   SCA_CTOR(prefi_ac)
   {    // default parameter values
       fc0 = 1.0e3;  fc1=1.0e5;  v_max  = 1.0;
   }
};
```

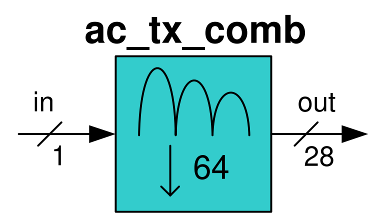
$$H(s) = \frac{1}{1 + \frac{1}{2\pi f_c} s}$$



# Frequency Domain Specification – Example

```
SCA_SDF_MODULE(ac_tx_comb)
{
   sca_sdf_in<bool>            in;
   sca_sdf_out<sc_int<28> > out;

   void attributes()
   {
      in.set_rate(64);     // 16 MHz
      out.set_rate(1);     // 256 kHz
   }

   void ac_sig_proc()
   {
      double        k  = 64.0;   // decimation factor
      double        n  =   3.0;   // order of comb filter
      sca_complex z1 = sca_ac_z(in.get_T().to_seconds(), -1);

      // complex transfer function:
      sca_complex h = pow((1.0 – pow(z,k)) / (1.0 – z), n);

      sca_ac(out) = h * sca_ac(in) ;
   }
```

```
   void sig_proc()
   {
      int x, y, i;
      for (i=0; i<64; ++i) {
         x = in.read(i);
      ...
      out.write(y);
   }

   SCA_CTOR(ac_tx_comb)
   {
      ...
   }
};
```

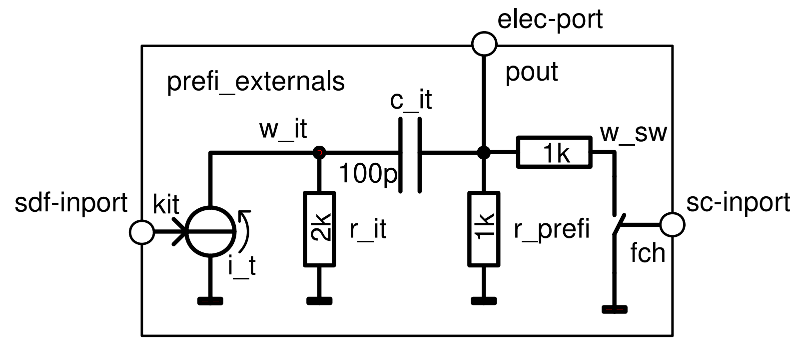



$$H(z) = \left( \frac{1 - z^{-k}}{1 - z^{-1}} \right)^n \qquad z = e^{j2\pi \, f/f_s}$$

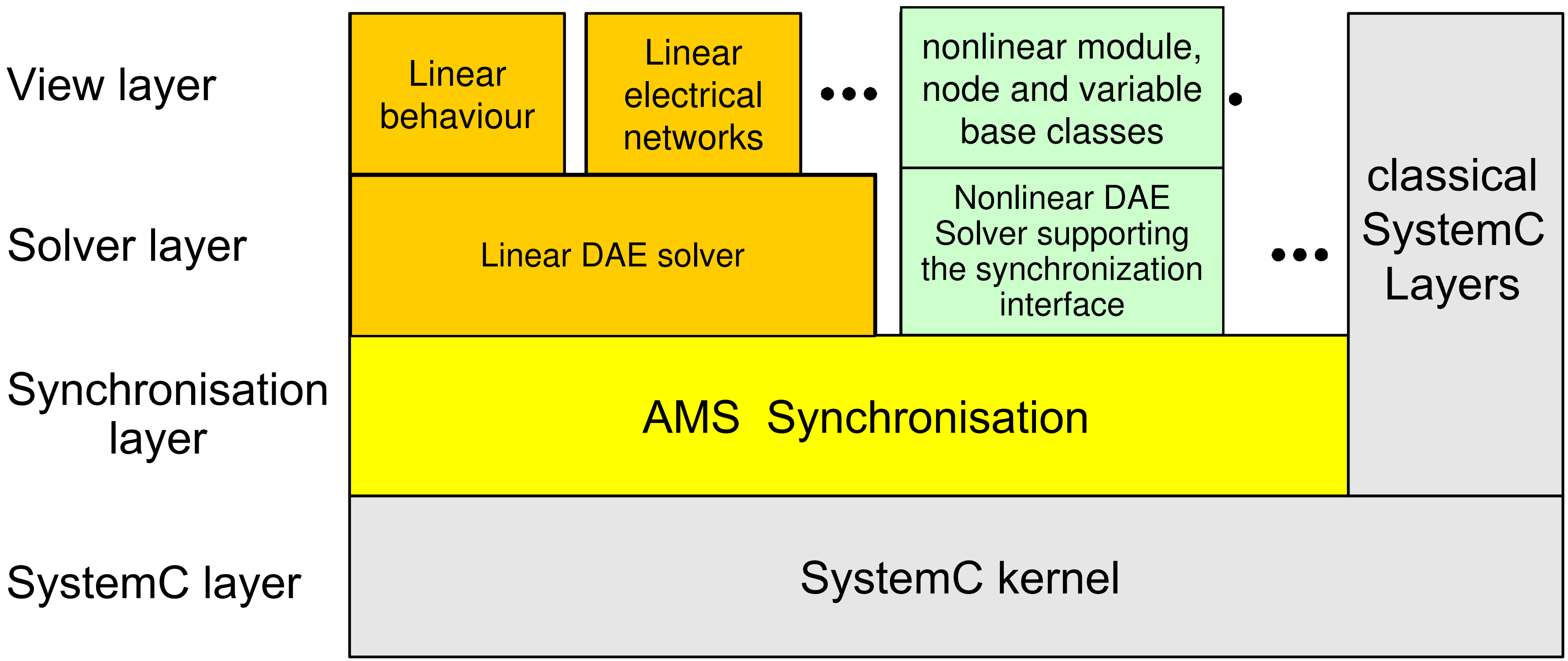
# Hierarchical Modules – Linear Network Example

```
SC_MODULE(prefi_externals)
{
   // synchronous dataflow inport
   sca_sdf_in<double>   kit

   // converter inport (connect with sc_signal<bool>)
   sca_scsdf_in<bool> fch;

   // electrical port
   sca_elec_port   pout;

   // internal nodes declaration
   sca_elec_node w_it, w_sw;
   sca_elec_ref     gnd;

   // component declarations
   sca_r              *r_it, *r_prefi, *r_prefi2;
   sca_c              *c_it;
   sca_sdf2i          *i_t;
   sca_sc_rswitch  *sw_prefi;
```

```
 SC_CTOR(prefi_externals)
 {
    i_t = new sca_sdf2i("i_t");
       i_t->p(gnd);
       i_t->n(w_it);
       i_t->ctrl(kit);

    r_it = new sca_r("r_it");
       r_it->p(gnd);
       r_it->n(w_it);
       r_it->value=2.0e3;

    …
 }
};
```



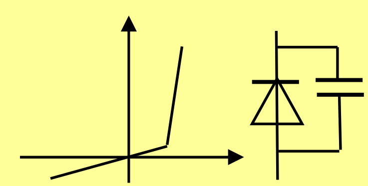
# SystemC-AMS Layers



# Example Nonlinear Primitive Description

```
SCA_NL_MODULE(sca_nl_rdiode)
{
   sca_nl_elec_port a;
   sca_nl_elec_port b;

   double v_thres;
   double r_on;
   double r_off;
   double cj;

   sca_nonlinnet_var  v_diode;

   void equations();

   SCA_CTOR(sca_nl_rdiode)
   {
      v_thres = 0.7;
      r_on      = 1e-2;
      r_off     = 1e7;
      cj          = 1e-12;
   }

};
```

```
#include "sca_nl_rdiode.h"

void sca_nl_rdiode::equations()
{
   null(v_diode) = v_diode – ( a.v() – b.v() );

   double i_diode;

   if ( v_diode.above(v_thres) )
   {
      i_diode = ( v_diode – v_thres ) / r_on
                    + v_thres / r_off;
   }
   else
   {
      i_diode = v_diode / r_off;
   }

   i_diode += cj * v_diode.dt();

   a += i_diode;
   b –= i_diode;
}
```



# Summary

- SystemC-AMS is an extension library for SystemC.
- A prototype implementation is publicly available from Fraunhofer.
- It supports: modelling of non-conservative systems, multi-rate static dataflow modelling, linear electrical conservative networks, linear behavioural functions, frequency domain (ac) simulation, powerful trace functionality.
- Download at http://www.systemc-ams.org/.
- The prototype will be further developed by the OSCI Analogue Mixed-Signal Working Group.
- Extensions available at FhG: Switched Capacitor solver, Non-linear DAE solver with SystemC / DE synchronisation
- Our SystemC-AMS related activities are presented at http://systemc-ams.eas.iis.fraunhofer.de/.

Thanks for your attention!

Are there still questions?