# SystemC / AMS

Franco Fummi


UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Version 1.0

# Use cases

- Executable specification
  - Verify correctness of system requirement using simulation
- Virtual prototyping
  - High-level (untimed/timed) model of HW architecture
- Architecture exploration
  - Evaluate mapping between behavior and system architecture
- Integration validation
  - Verify the correctness of integrated components

# Discrete-time vs. Continuous-time descriptions

- Discrete-time descriptions:
  - Signals and physical quantities defined at discrete time points
    - Assumed constant between time points
  - Behavior as procedural assignments involving sampled signals
  - Well suited for describing signal-processing dominated behaviors
    - Signals are naturally (over)sampled
  - Used also for describing (approximation of) continuous-time behaviors
- Continuous-time descriptions:

AMS = Analog Mixed Signals ⇒ We can model both discrete and continuous signals

  - Signal and physical quantities described as real-valued functions of time
    - Time considered as a continuous value
  - Behavior as Differential algebraic equations (DAE) or Ordinary differential equations (ODE)
    - Solved by linear or non-linear solver (complex algorithms)
  - Well suited for describing physical behaviors of dynamic systems

# Non-conservative vs. conservative descriptions

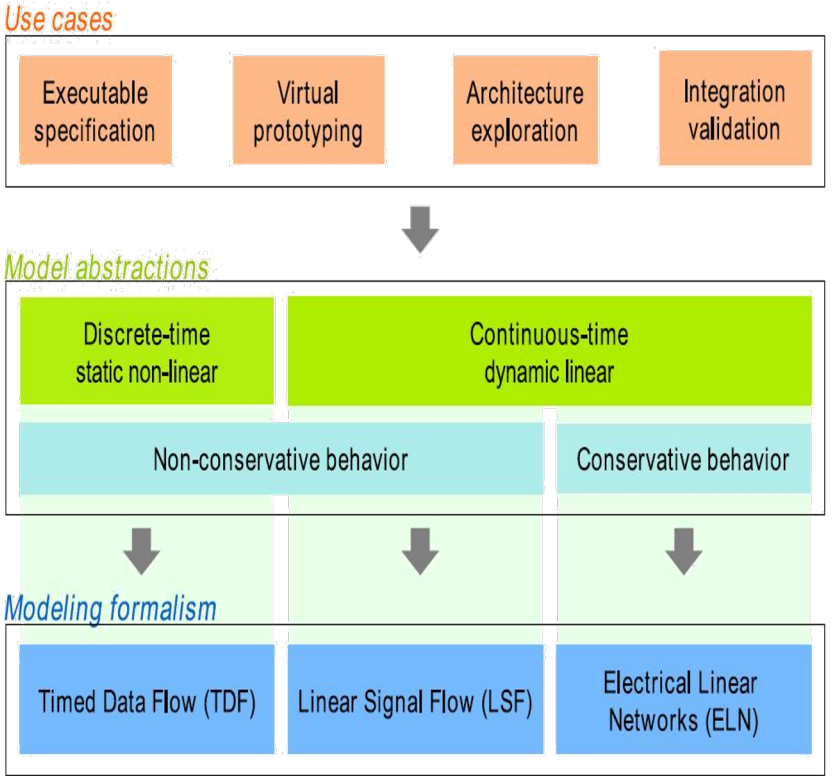
- Non-conservative descriptions:
  - Behavior expressed as directed flows of continuous time signals or variables
    - Processing functions (e.g., filtering or integration) are applied
  - Non-linear dynamic can be described
  - Mutual effects and interaction between AMS components not supported
    - E.g., impedances or loads
- Conservative descriptions: describe only a subset of the system's equation because some can be derived from standard laws
  - Based on satisfaction of Kirchhoff's laws
  - Huge sets of equations to solve are inferred
    - Computational hard

# Modeling formalisms

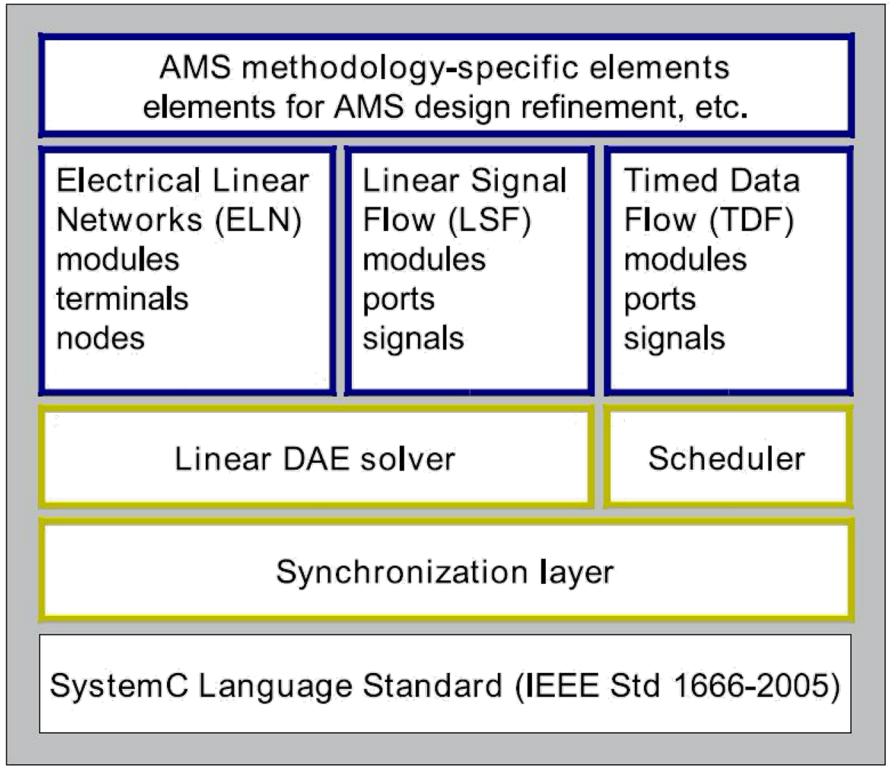
- **Timed Data Flow** (TDF)
  - Discrete time, non-conservative modeling
  - Static scheduler (based on dataflow)
- **Linear Signal Flow** (LSF)
  - Continuous, non-conservative modeling
  - DAE and ODE based
    - Represented by connection of primitives for real-valued time-domain signals
  - Symbolic and numeric solvers
- **Electrical Linear Network** (ELN)
  - Continuous, conservative modeling
  - Modeling of electrical networks
    - Linear network primitives (e.g., resistors, capacitor etc…)
    - Continuous relations between voltage and currents

Efficiency reasons: Using SystemC timing part would force jumping from SystemC-AMS kernel to SystemC kernel

E.g.: resistors, capacitors, …



# Language Architecture




**Semantics** defined in the AMS language standard

**User features**
Classes and interfaces defined in the AMS language standard

**Enabling technology**
Classes and interfaces *not* defined in the AMS language standard
*( implementation defined )*

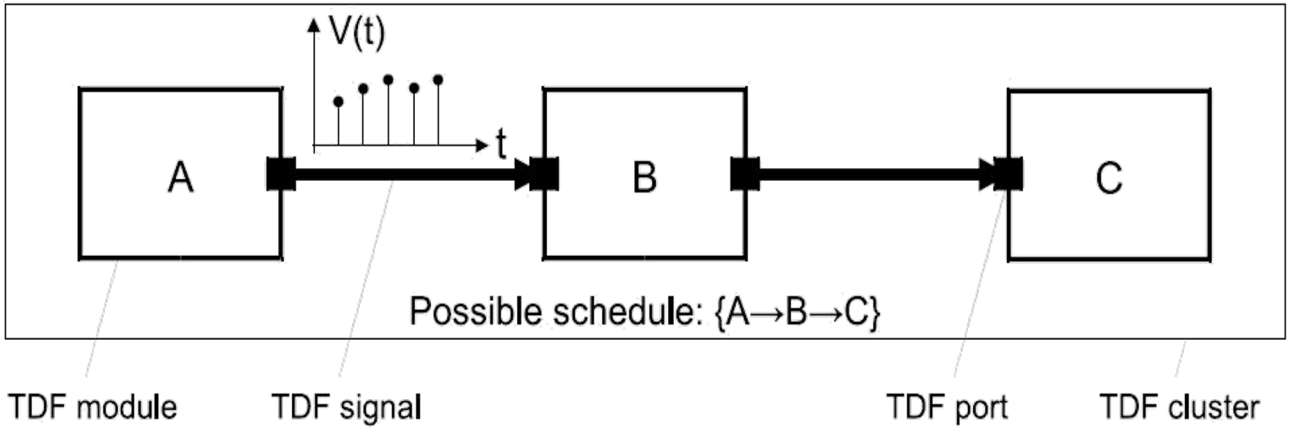
# TIME DATA FLOW

# Fundamentals

- Based on Synchronous Data Flow (SDF)
  - SDF untimed, TDF discrete-time
- Each TDF module contains a C++ method
  - Computes a mathematical function
    - Depending on its direct inputs
    - Can depend also on its internal states
  - Composition of modules in appropriate order

# Composition: Example

- Overall behavior given by: $f_C(f_B(f_A))$



# Function execution

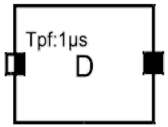
- A given function is executed iff there are enough samples available at the input port
  - Fixed number of consumed and produced samples

- Every sample has a time stamp
  - Fixed interval called time step

# TDF module and port attributes

- Time step (module)
- Time step (port)
- Rate (port)
- Delay (port)
- Time offset (port)
  - Specialized port



# Scheduling

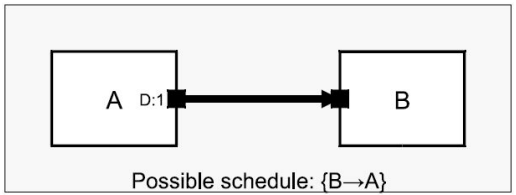
- TDF Cluster: set of connected TDF modules which belong the same static schedule
  - Parameters must be compatible
  - Defines a sequence in wich TDF modules are executed

- Overcome SystemC event-based scheduler
  - Increase in efficiency
  - Interaction with pure SystemC through specific converter ports

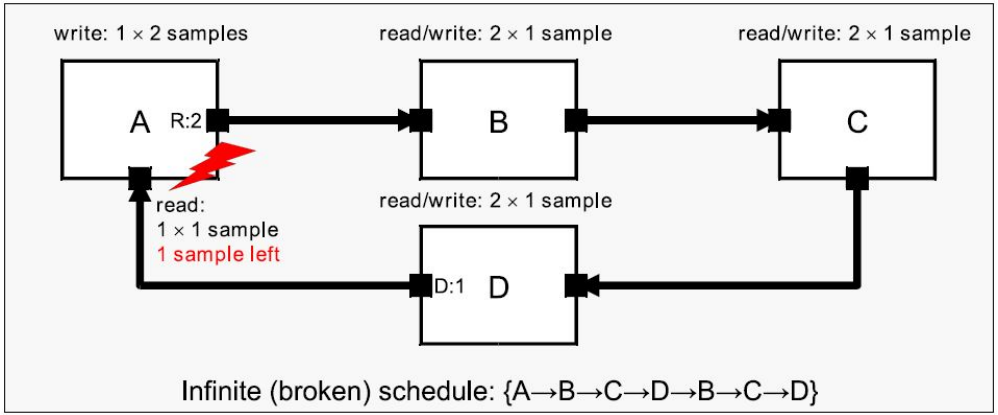
# Scheduling feasibility

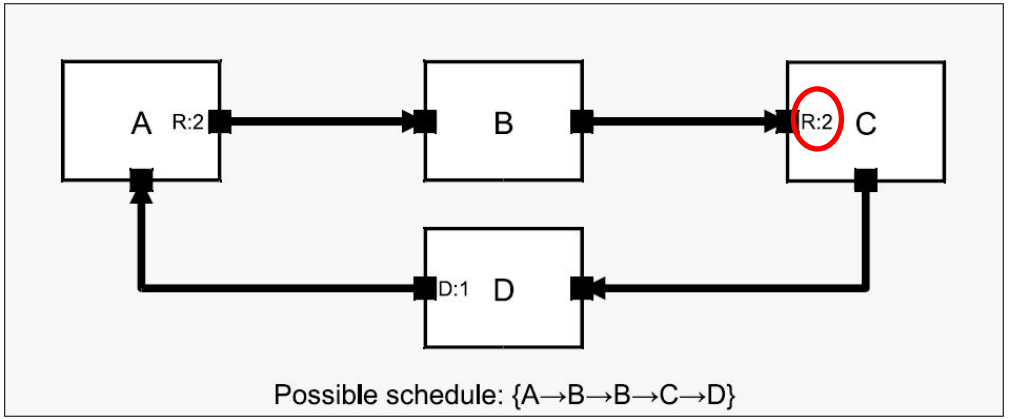
- Loops are sources for problems:
  - Every loop must present at least one delay port
  - No leftover samples allowed

- Delay ports can lead to inconsistency:
  - Initial value should be specified



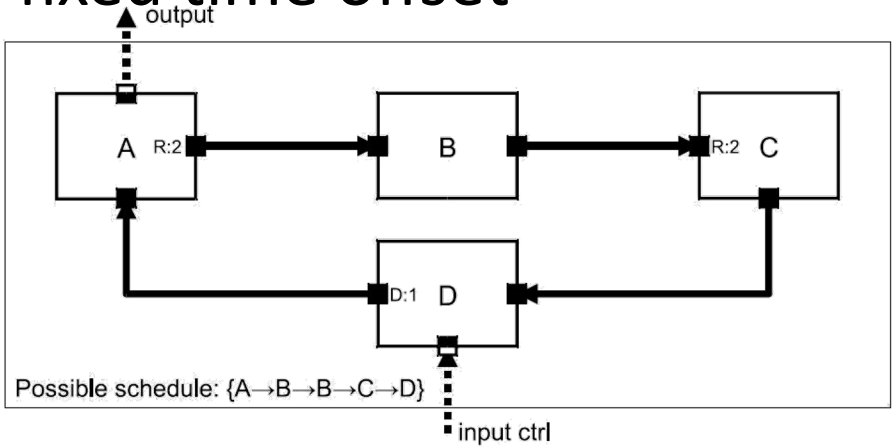
# Scheduling loops



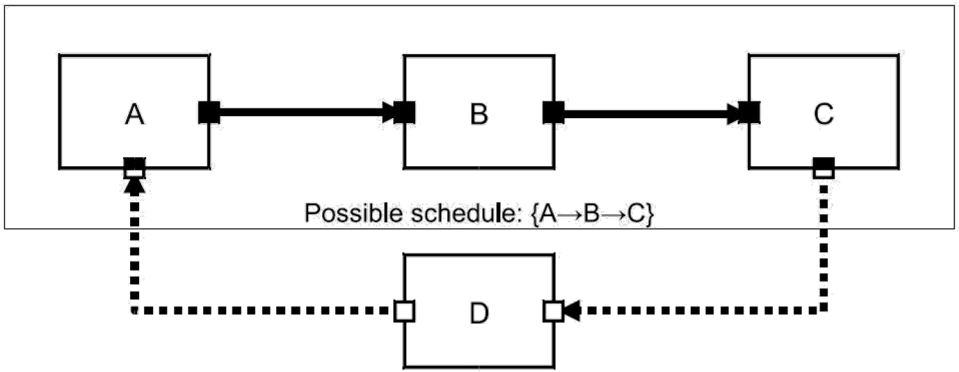
# Connection to discrete-event models

- Converter input/output port are given
  - With a fixed time offset



# Closed loop and discrete domain

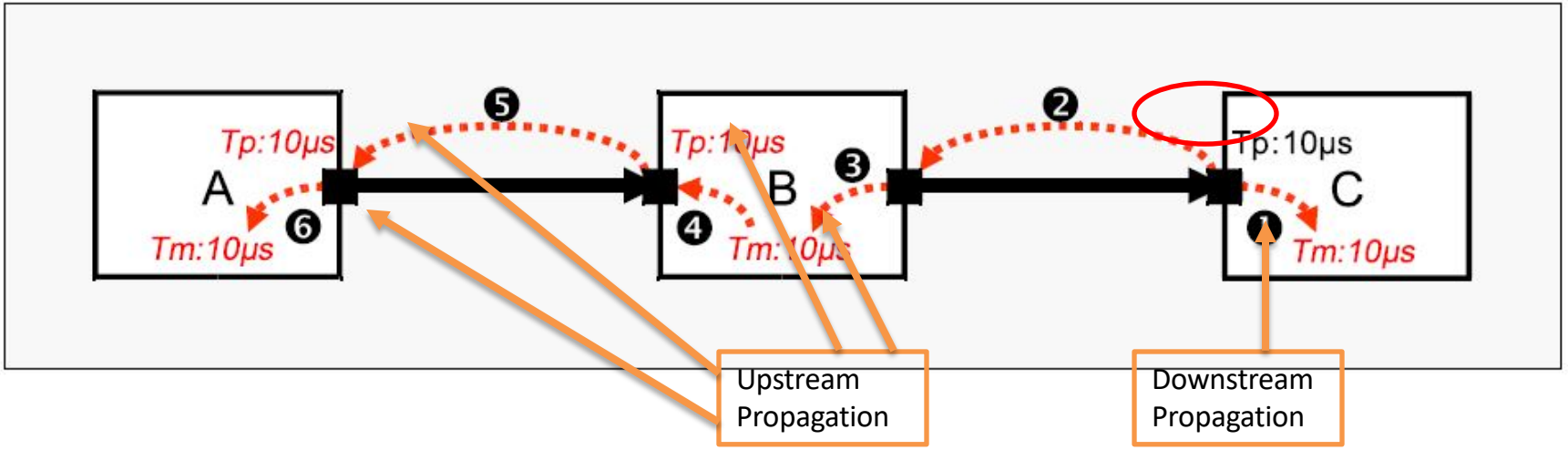
- Special converter ports introduce implicit delay
  - Data readed by A, reach C in the same Delta cycle
  - Data produced by C are read by A after 1 cycle



# Time step assignment and propagation

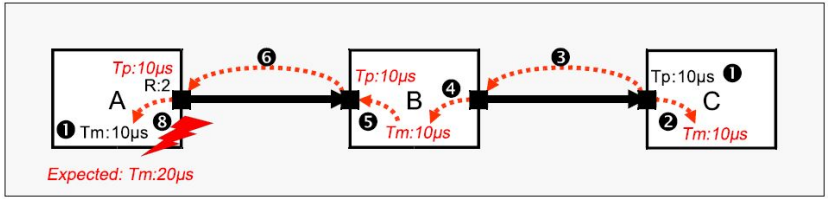
- Port rates and delays are very useful
  - Handle different frequency domains
  - Express models involving nested loops
- Compatibility mandatory for cluster consistency
  - Consistency is indipendent from sampling period
- Compatibility assure inference of the time step
  - Upstream and Downstream

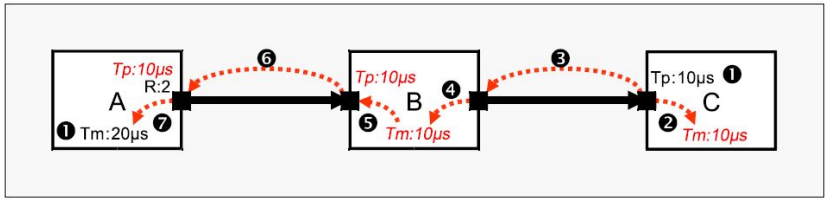
# Time step assignment and propagation (Example)



# Consistency

- $T_m = T_{p_{input}} * R_{input} = T_{p_{output}} * R_{output}$



# Multiple clusters

- A TDF model can present more clusters
  - Every cluster is scheduled independetly
  - Every cluster can be connected to the others

- Communication between TDF cluster evolves accordingly to the SystemC discrete-event scheduler!

# Language Constructs

- TDF Modules
- TDF Ports
- TDF Signals

# TDF Modules

```
SCA_TDF_MODULE(my_tdf_module) ❶
{
  // port declarations
  sca_tdf::sca_in<double>  in; ❷
  sca_tdf::sca_out<double> out;

  SCA_CTOR(my_tdf_module) {} ❸

  void set_attributes() ❹
  {
    // module and port attributes
  }

  void initialize() ❺
  {
    // initial values of ports with a delay
  }

  void processing() ❻
  {
    // time-domain signal processing behavior or algorithm
  }

  void ac_processing() ❼
  {
    // small-signal frequency-domain behavior
  }
};
```

- Primitive module declaration
  - Macro or extending sca_tdf::sca_module
- Input and output ports declarations
- Constructor
  - Macro or by creating a new class
    - Derived from sca_tdf::sca_module
    - Always with paramenter sc_core::sc_module_name
  - Only function that can be called by the user
- Function to define module and ports attributes
- Function to initialize data members
  - Representing module state and samples
- Implementation of functionality
  - Processing function in time-domain
- Implementation of functionality and noise
  - Small-frequency domain can be used to model noise

# TDF Modules: constraints on usage

- Hierarchy is not supported
  - Composition of TDF modules is possible using regular SystemC classes
- Function to describe discrete-event behavior are not allowed
  - SystemC forbidden constructs:
    - SC_HAS_PROCESS, SC_METHOD, SC_THREAD
    - wait, next_trigger, sensitive
  - Incompatibility in execution semantics
- Local time of a TDF module is calculate independently
  - get_time instead of sc_core::sc_time_stamp
- Specialized converter port to use SystemC signals

# TDF ports

- in and out port but no inout
  - Incopatible with the adopted model of computation
- «intra-cluster» ports:
  - sca_tdf::sca_in<T>
  - sca_tdf::sca_out<T>
- «extra-cluster» ports:
  - sca_tdf::sca_de::sca_in<T>
  - sca_tdf::sca_de::sca_out<T>

# TDF ports (cont.d)

- Getter and Setter methods are given for ports attributes
  - Available attributes: simestep, rate, delay, timeoffset
  - set_[attribute]( value ) and get_[attribute]() functions defined
  - Setters called by set_attribute(), Getters by initialize() and process()

- A port with non-zero delay must be always initialized
  - Member function initialize() of the module

- Multirate port permit to read/write on a specific sample
  - E.g., input_port.read(0) or output_port.write(val, 1)

# TDF Signals

- Used to connect TDF ports of different modules
  - They carry the sample, ports determine direction
  - sca_tdf::sca_signal<T>

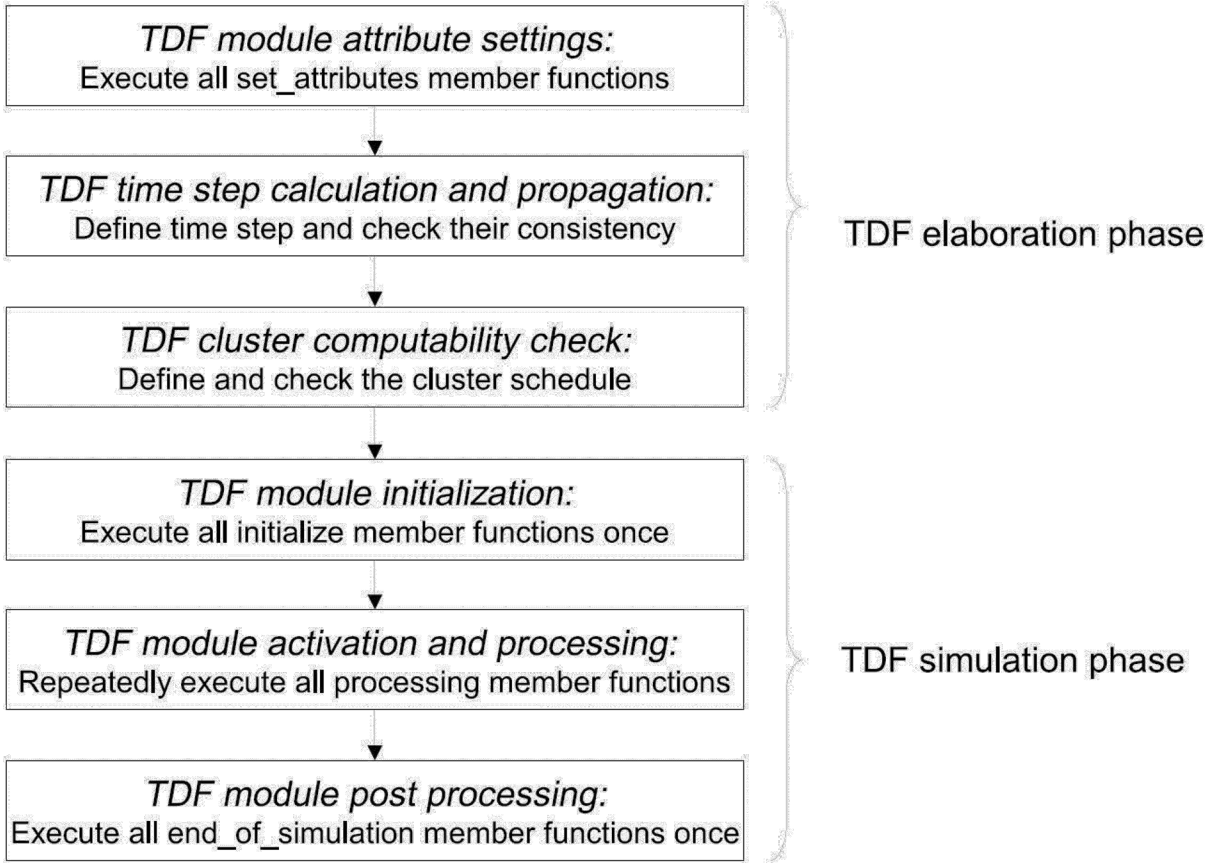
- Little differences with SystemC signals
  - read() and write() functions are not provided
  - Naming for signal as in SystemC on costructor

# TDF and continuous-time modeling

- TDF is based on production of (discrete) samples

- It is possible to embed linear dynamic equations
  - Linear transfer function (Laplace)
    - Numerator-denominator form (sca_tdf::sca_ltf_nd)
    - Zero-pole form (sca_tdf::sca_ltf_zp)
    - Coefficients as objects of sca_util::sca_vector
    - Complex domain through sca_util::sca_complex
  - State-space equations
    - Matrix based representation (sca_tdf::sca_ss)

# TDF execution semantics



# LINEAR SIGNAL FLOW

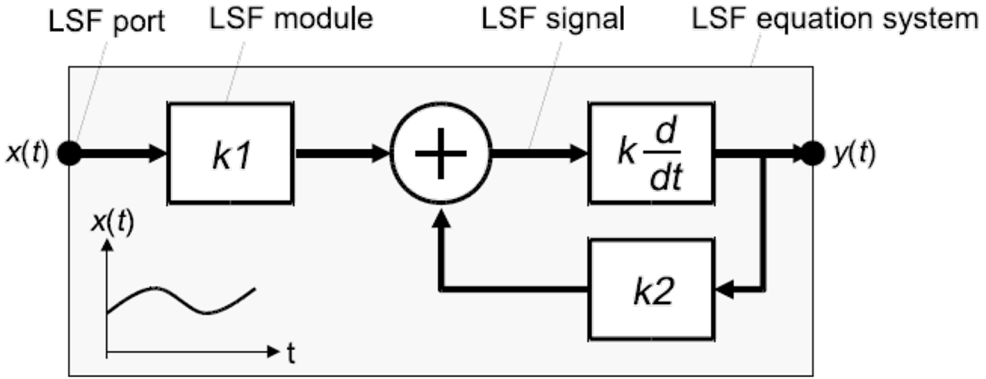
# Fundamentals

- Behavior defined as relations between variables of a set of Linear DAE
  - Continuous-time modeling using directed real-value signals
    - Non conservative description
    - Only one real-valued is used to represent each signal
- Signal flow models represented by block diagram notation
  - Elementary parts represented by blocks
  - Signal used to interconnect these blocks
- An LSF model is composed by a set of connected LSF modules (LSF equation system or cluster)

# Block diagrams

- Elementary parts or functions
  - LSF Blocks
    - Every block introduces a well-defined function
    - Define within a Standard API
    - Extension of API: “forbidden”!
- Interconnection between basic blocks
  - LSF Signals
    - Specialization of SystemC signals
    - Fixed (Real) data-type
- Interfaces for discrete models
  - Timed Data Flow
  - SystemC-RTL

# Setup of LSF equations




- Equations system composed by the constructor
  - Compose mathematical equations
    - Blocks
    - Interconnections
  - Set to default values the non-specified parameters
- E.g., $y(t) = k_1 \frac{dx(t)}{dt} + k_2 \frac{dy(t)}{dt}$

# LSF modules

| LSF module name | Description |
|---|---|
| **sca_lsf::sca_add** | Weighted addition of two LSF signals. |
| **sca_lsf::sca_sub** | Weighted subtraction of two LSF signals. |
| **sca_lsf::sca_gain** | Multiplication of an LSF signal by a constant gain. |
| **sca_lsf::sca_dot** | Scaled first-order time derivative of an LSF signal. |
| **sca_lsf::sca_integ** | Scaled time-domain integration of an LSF signal. |
| **sca_lsf::sca_delay** | Scaled time-delayed version of an LSF signal. |
| **sca_lsf::sca_source** | LSF source. |
| **sca_lsf::sca_ltf_nd** | Scaled Laplace transfer function in the time-domain in the numerator-denominator form. |
| **sca_lsf::sca_ltf_zp** | Scaled Laplace transfer function in the time-domain in the zero-pole form. |
| **sca_lsf::sca_ss** | Single-input single-output state-space equation. |
| **sca_lsf::sca_tdf::sca_gain**, **sca_lsf::sca_tdf_gain** | Scaled multiplication of a TDF input signal with an LSF input signal. |
| **sca_lsf::sca_tdf::sca_source**, **sca_lsf::sca_tdf_source** | Scaled conversion of a TDF input signal to an LSF output signal. |
| **sca_lsf::sca_tdf::sca_sink**, **sca_lsf::sca_tdf_sink** | Scaled conversion from an LSF input signal to a TDF output signal. |
| **sca_lsf::sca_tdf::sca_mux**, **sca_lsf::sca_tdf_mux** | Selection of one of two LSF input signals by a TDF control signal (multiplexer). |

# LSF modules (Cont.d)

| LSF module name | Description |
|---|---|
| **sca_lsf::sca_tdf::sca_demux, sca_lsf::sca_tdf_demux** | Routing of an LSF input signal to either one of two LSF output signals controlled by a TDF signal (demultiplexer). |
| **sca_lsf::sca_de::sca_gain, sca_lsf::sca_de_gain** | Scaled multiplication of a discrete-event input signal by an LSF input signal. |
| **sca_lsf::sca_de::sca_source, sca_lsf::sca_de_source** | Scaled conversion of a discrete-event input signal to an LSF output signal. |
| **sca_lsf::sca_de::sca_sink, sca_lsf::sca_de_sink** | Scaled conversion from an LSF input signal to a discrete-event output signal. |
| **sca_lsf::sca_de::sca_mux, sca_lsf::sca_de_mux** | Selection of one of two LSF input signals by a discrete-event control signal (multiplexer). |
| **sca_lsf::sca_de::sca_demux, sca_lsf::sca_de_demux** | Routing of an LSF input signal to either one of two LSF output signals controlled by a discrete-event signal (demultiplexer). |

# Time step assignment and propagation

- The time step
  - Simulation step for a LSF cluster
  - Disconnected by the numerical integration algorithm
    - Integration step can be smaller (never larger)
    - In PoC implementation (1.0): time step = integration step
- Can be explicitly assigned for every block or every cluster
  - At least one block: time step explicitly set
- Can be inferred by a propagation mechanism
  - Inferred by time step assignments within blocks
  - Connection of LSF and TDF
    - TDF Dynamic Time Step introduced in 2.0 standard
- Must preserves consistency

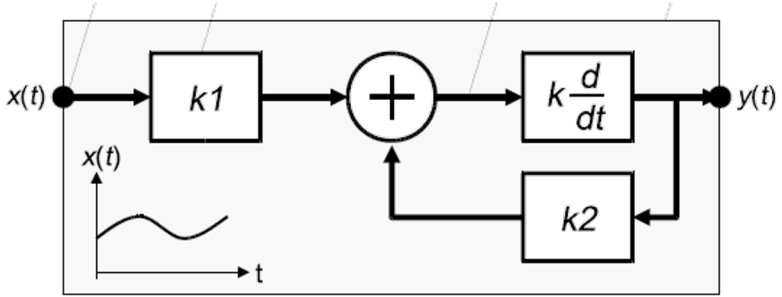
# LSF Port and Signals

- Two classes of Ports
  - sca_lsf::sca_in
  - sca_lsf::sca_out
  - Fixed data-type (called *signal flow nature)*
    - Implicitly a double
    - Convertions are forbidden!
  - Special ports to connect the discrete world!

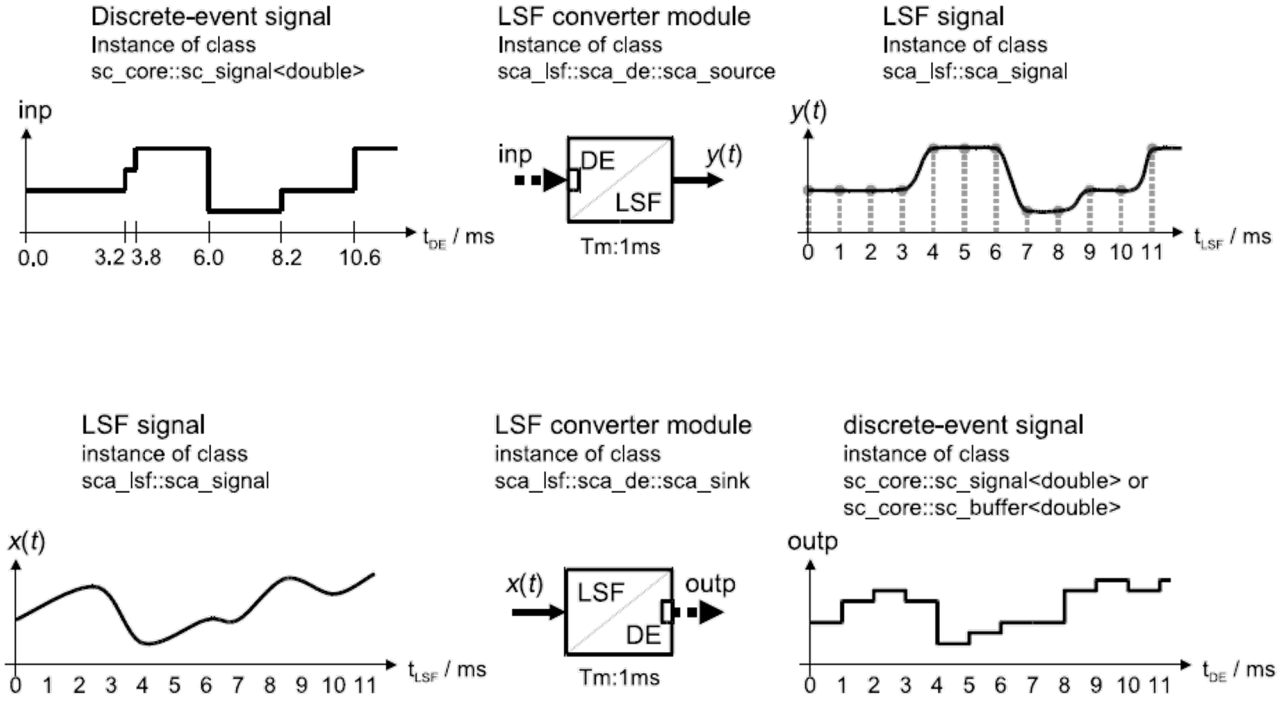
- Signals
  - Used to bind LSF ports and components
  - Determine the direction of the signals
  - Not a templatic class
    - They can’t be customized!

# Modeling continuous-time behavior

```
SC_MODULE(my_structural_lsf_model)
{
  sca_lsf::sca_in  x; ❶
  sca_lsf::sca_out y;

  sca_lsf::sca_gain gain1, gain2; ❷
  sca_lsf::sca_dot dot1;
  sca_lsf::sca_add add1;

  my_structural_lsf_model( sc_core::sc_module_name, double k1, double k2 )
  : x("x"), y("y"), gain1("gain1", k1), gain2("gain2", k2), dot1("dot1"), add1("add1"), ❸
    sig1("sig1"), sig2("sig2"), sig3("sig3")
  {
    gain1.x(x); ❹
    gain1.y(sig1);
    gain1.set_timestep(1,sc_core::SC_MS); ❺

    add1.x1(sig1);
    add1.x2(sig3);
    add1.y(sig2);

    dot1.x(sig2);
    dot1.y(y);

    gain2.x(y);
    gain2.y(sig3);
  }

 private:
  sca_lsf::sca_signal sig1, sig2, sig3; ❻
};
```

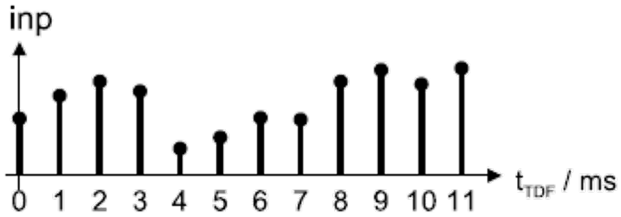


# Interaction with discrete world

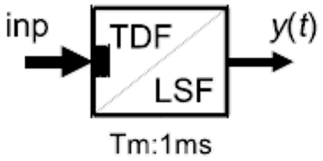


# Interaction with TDF

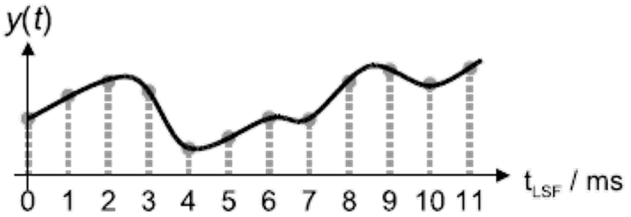
TDF signal
Instance of class
sca_tdf::sca_signal<double>



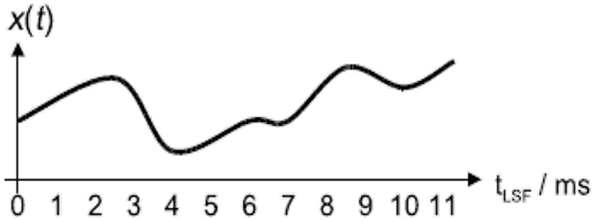

LSF converter module
Instance of class
sca_lsf::sca_tdf::sca_source



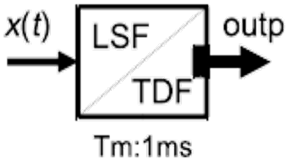

LSF signal
Instance of class
sca_lsf::sca_signal



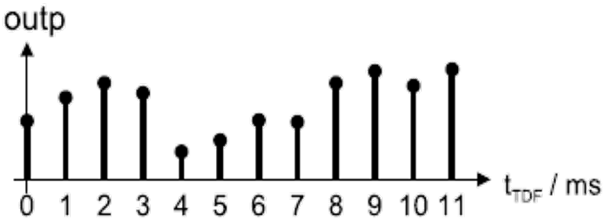

LSF signal
Instance of class
sca_lsf::sca_signal



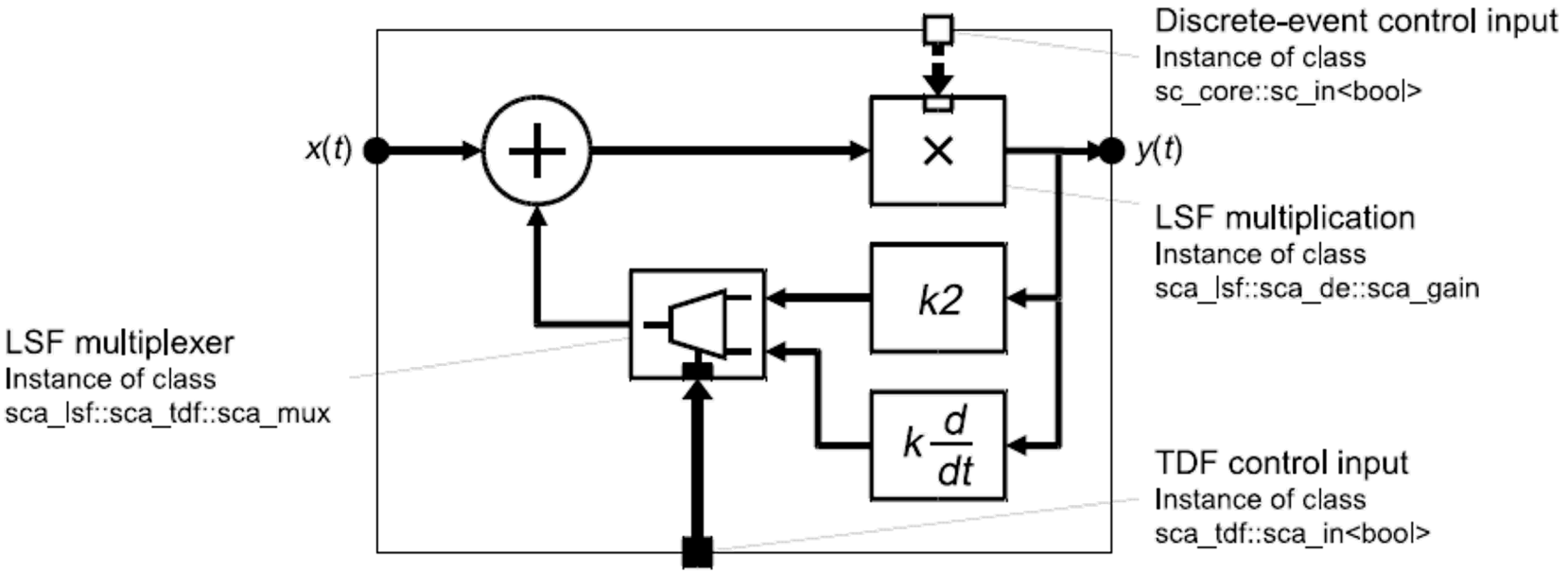

LSF converter module
Instance of class
sca_lsf::sca_tdf::sca_sink



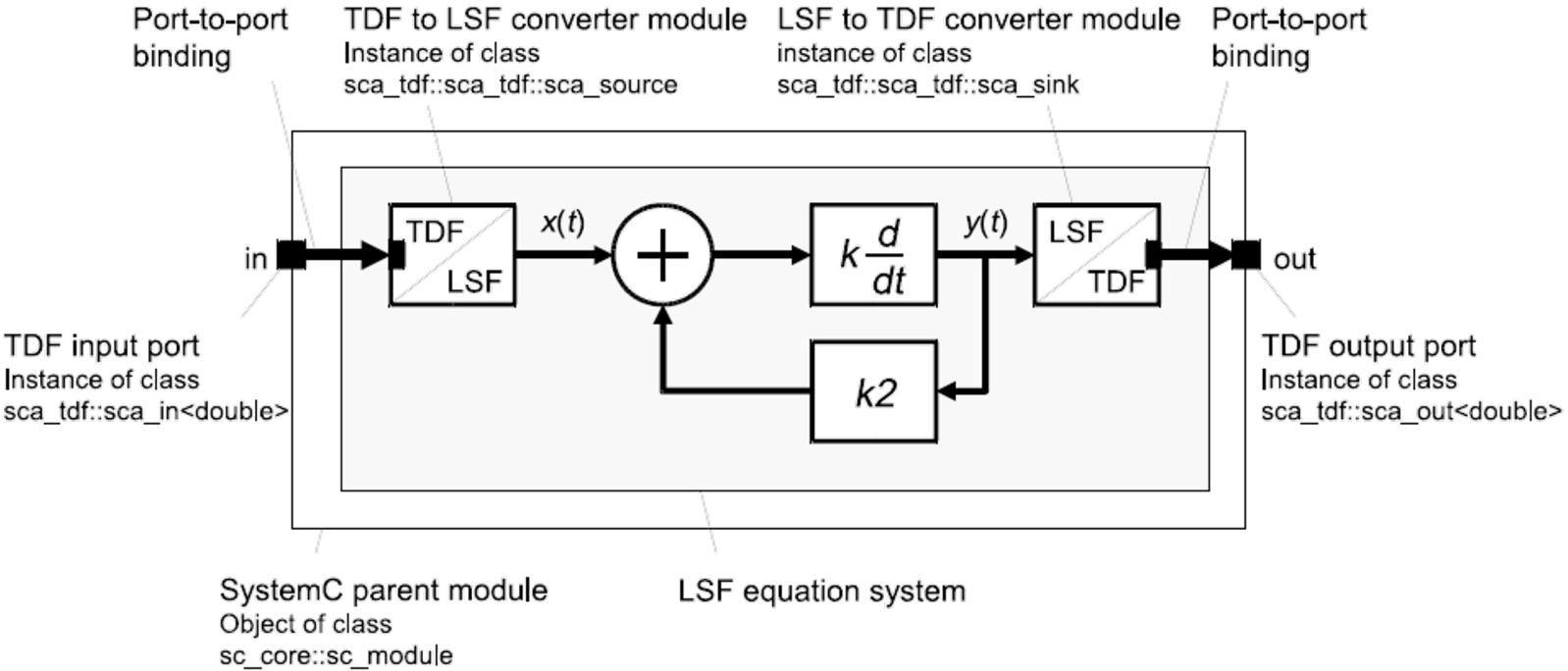

TDF signal
Instance of class
sca_tdf::sca_signal<double>



# Using discrete-events



# LSF Model encapsulation



# Execution Semantics