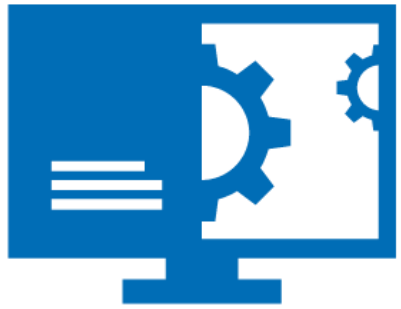
KÄRNTEN
University of
Applied Sciences
Integrated Systems and Circuit Design
WWW.CUAS.AT

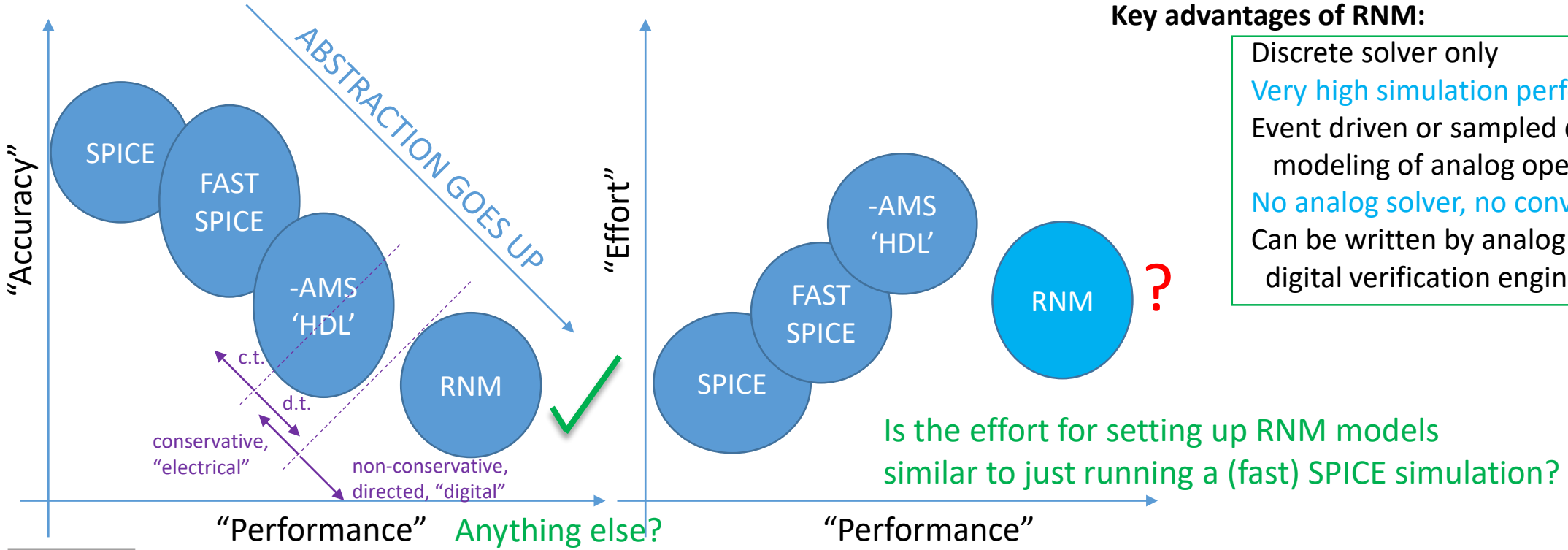


Beyond real number modeling: Comparison of analog modeling approaches.

Wolfgang Scherr, FDL 2020, 17th Sep, Kiel

A **qualitative** view on state of the art:

- Behavioral modeling of M/S systems



Are these really (and always) advantages?

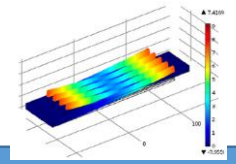
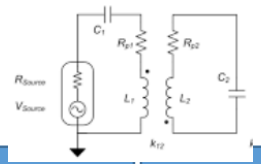
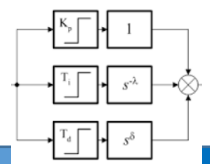
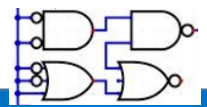
Key advantages of RNM:

Discrete solver only
Very high simulation performance ?
Event driven or sampled data modeling of analog operation
No analog solver, no convergence problems!
Can be written by analog designers and/or digital verification engineers

Is the effort for setting up RNM models similar to just running a (fast) SPICE simulation?

Source (modified/simplified):
"Real number modeling capabilities", A. Osman, DVCON 2015 https://dvcon-europe.org/sites/dvcon-europe.org/files/archive/2015/proceedings/DVCon_Europe_2015_T13_Presentation.pdf

Models, models, everywhere…


Finite Element	Nonlinear Conservative	Linear Conservative	Linear Non-Conservative	Dataflow	Discrete Event	Transactions, Threads, …	Algorithmic/ Meta / SW
HFSS, ADS, Momentum,… *(Keysight, Anysys, …)*		Simulink/Plecs			*(Mathworks or Simetrix Suite)*		Matlab
	(Open)Modelica (incl. multiphysics)						Ptolemy/…
	(H)Spice		speed-up		VHDL	*Digital Simulators*	Excel/VBA, etc.
	VHDL-AMS				Verilog		ISO C/C++
	Verilog-A				SystemVerilog		SysML/UML /XML/…
	Analog Simulators				SystemC incl. TLM		Python (NumPy)
							Octave

*) * different tool/vendor couplings (Cadence, Mentor, Synopsys, Simplis, Ansys,…)*

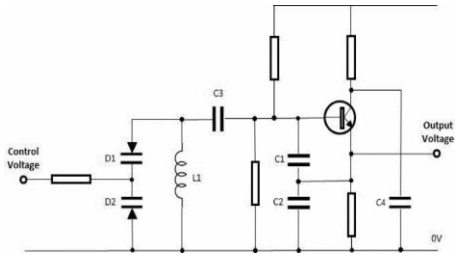
real number models

time continuous | time discrete | (often) timeless (by order, etc…)

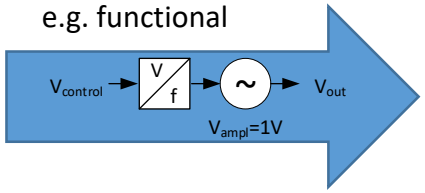
Please note: qualitative, not exhaustive and simplified overview. There are many, many more tools and languages out there, these are just representing some most common ones!

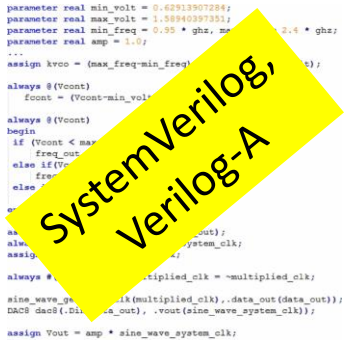
What about the effort?

- VCO example*)



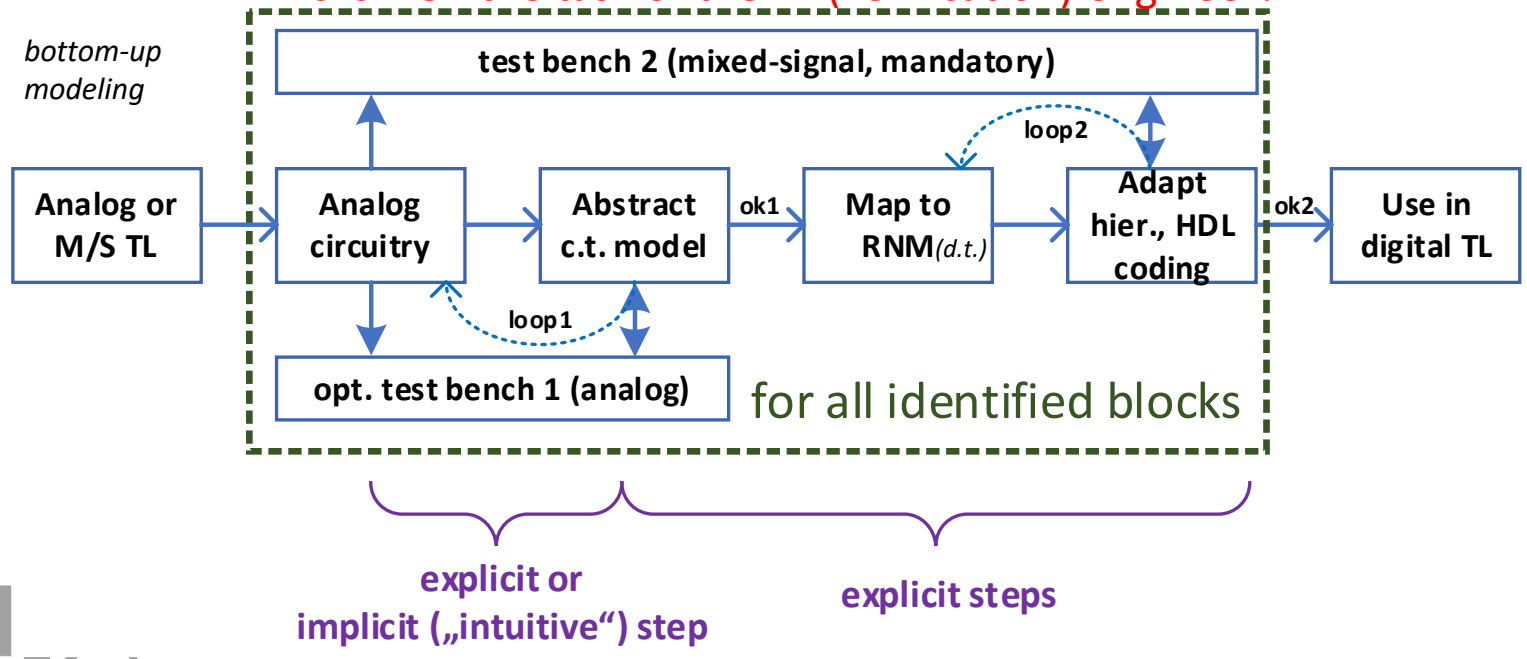
implicit assumption
of abstract behaviour
e.g. functional





How to match model with design?

This is NOT the task of the TL (verification) engineer!



Important facts:

Additional tasks/effort/skills on top of analog design!

.) the engineer is **trained and skilled** for writing **efficient, stable** HDL models

.) writing **maintainable models** which can be re-used or updated on design changes

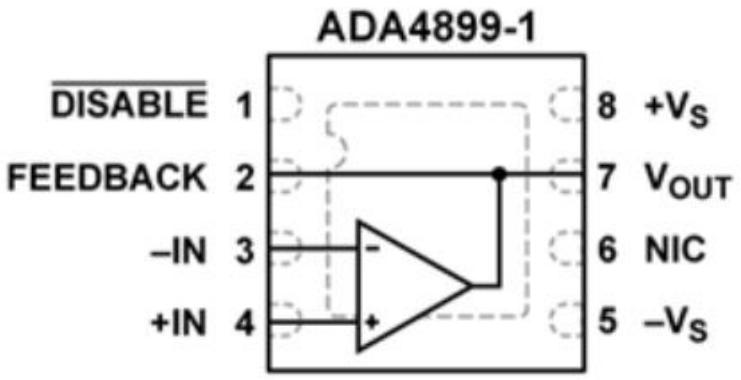
.) implicit or (well documented) explicit analog abstraction of (SPICE) circuits

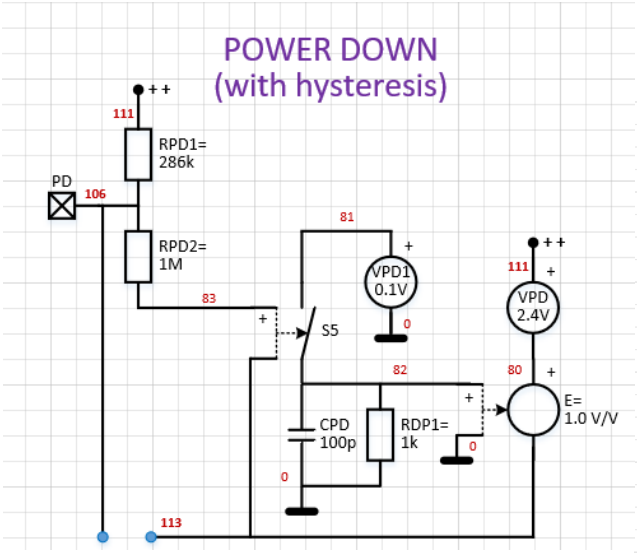
.) the model itself has a **proper verification coverage** to safely replace the circuit

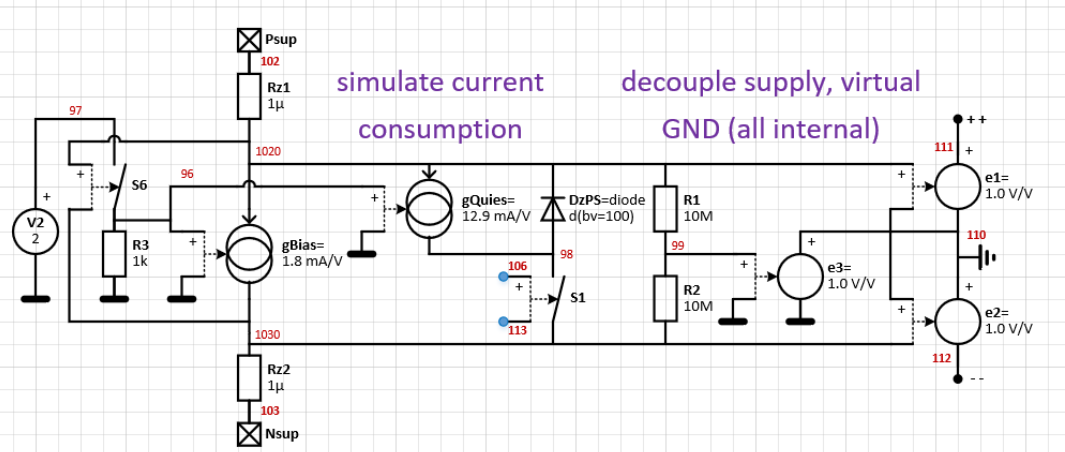
*) Source:
“Design of a SystemVerilog-Based VCO Real Number Model”, N. Georgoulopoulos at al, MOCAST 2019 https://ieeexplore.ieee.org/document/8741782

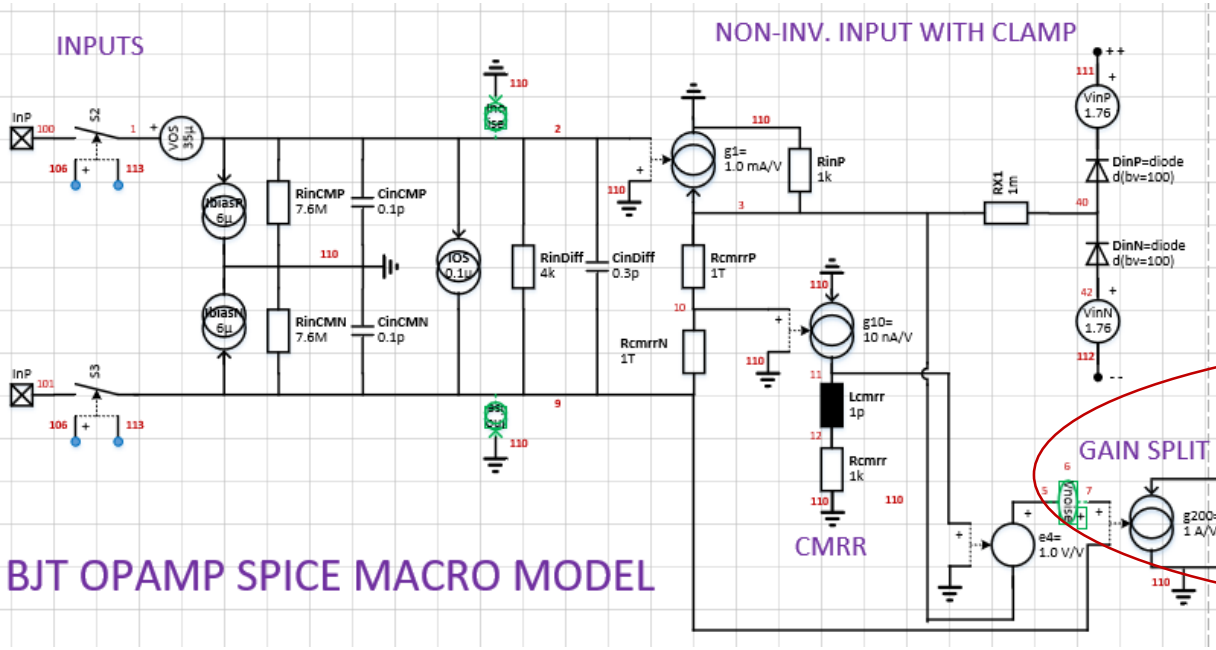
Many functional aspects in real-live (complex) circuits…

Source: Analog Devices Datasheet and LT-Spice model

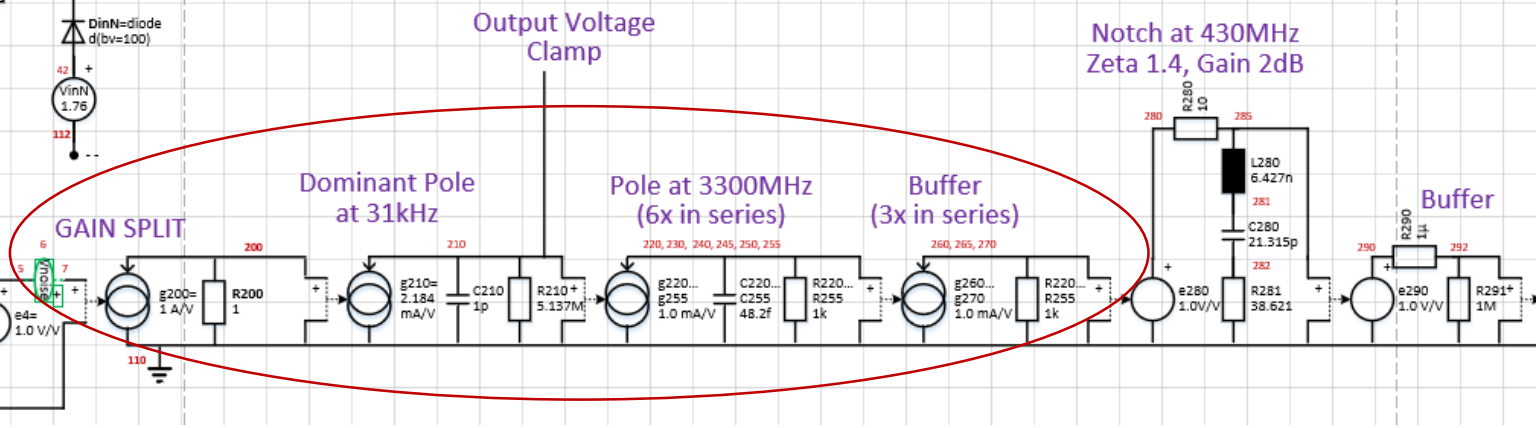




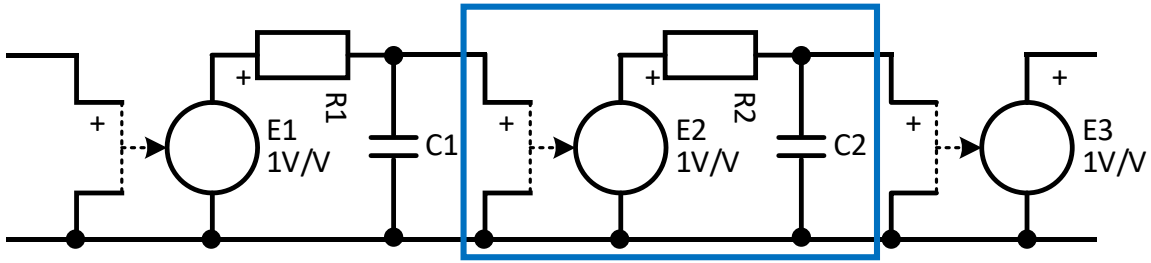




Assuming most important behavior:
gain and lowpass characteristics.



RNM using SystemVerilog and SystemC




```
`timescale 1ns/1ps
// RC lowpass model using an IIR representation
// (based on classic ct -> dt transformation)
// (c) 2020 Carinthia University of Applied Science
module RC_LP(input vreal inp, output vreal outp);
  parameter real R=1e3;     // series resistance [Ohm]
  parameter real C=1e-9;    // capacitance [F]
  parameter real fs=1e6;    // sample frequency [Hz]
  // initial calculations
  real Ts = 1.0 / fs;
  import "DPI" pure function real exp (input real x);
  real a = exp(-Ts/(R*C));
  // internal storage node
  real z_int = 0.0;
  // filter function in z domain
  always #(Ts*1s) begin
    z_int = a*z_int + (1.0 - a)*inp;
  end
  assign outp = z_int;
endmodule
```

SV

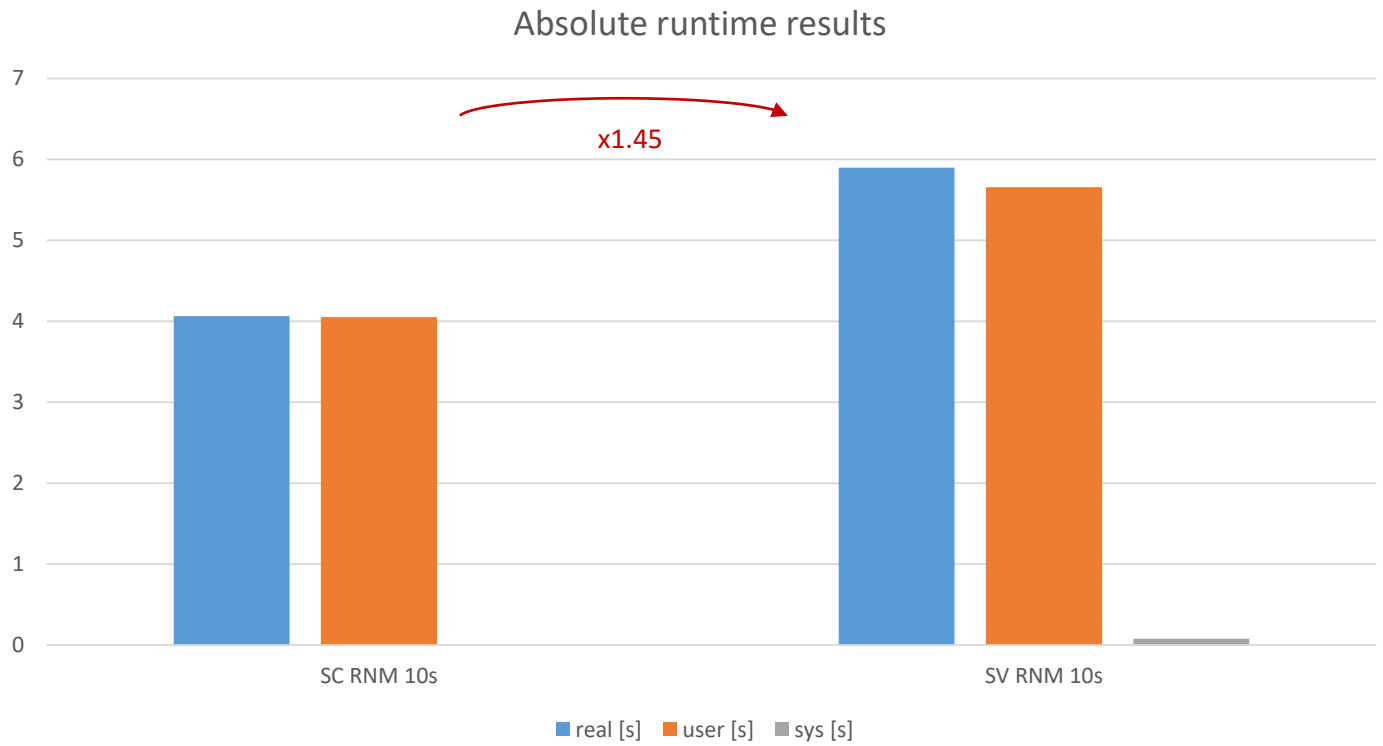
```
/// sc ports
sc_core::sc_in<double> inp;
sc_core::sc_out<double> outp;
// parameters
const struct params
{
    double R; /** resistor value [Ohm] */
    double C; /** capacitor value [F] */
    double fs; /** sample rate [s] */
    params() { R = 1e3; C = 1e-9; fs = 1e-6; }
} p;
// states
struct lp_de_coded::states
{
    float z_int;
  states(const params& p) { z_int = 0.0; }
};

  // SC_THREAD filter_process
  void lp_de_coded::filter_process()
  { static double Ts = 1.0/p.fs;
      static double a = exp(-Ts/(p.R*p.C));
      while(true) {
          s.z_int = a*s.z_int + (1.0-a)*inp.read();
          outp.write(s.z_int);
          wait(Ts, SC_SEC);
      }
  }
```

SC

RNM implement „mini-simulators“ running their MoC in each instance – hard to have global control e.g. of speed versus precision…

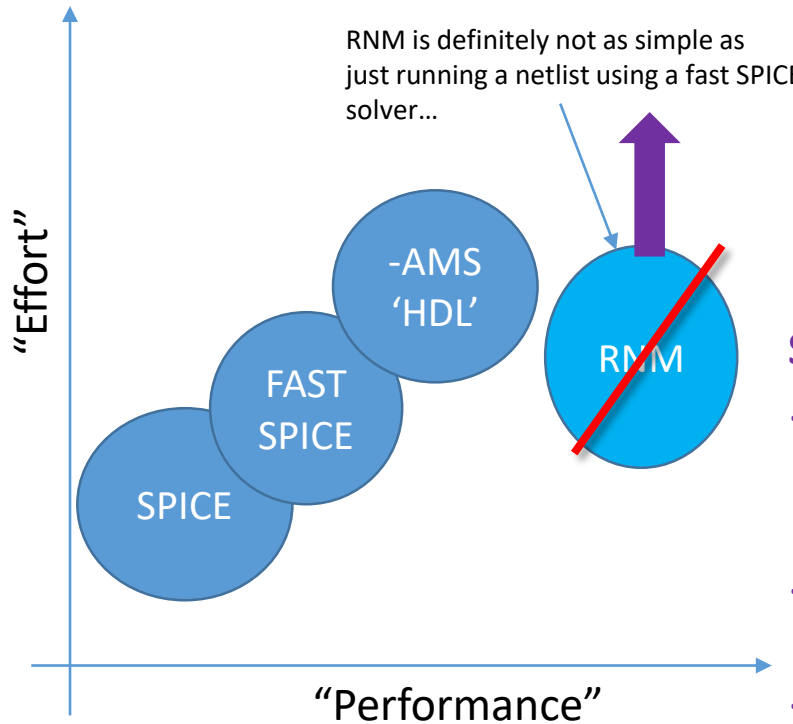
But these “mini-simulators” can be fast: RNM performance of SysC and SV

- RNM models execute in about half the real-time speed…



A qualitative view on “state of the art”

- Revising the effort diagram on RNM modeling

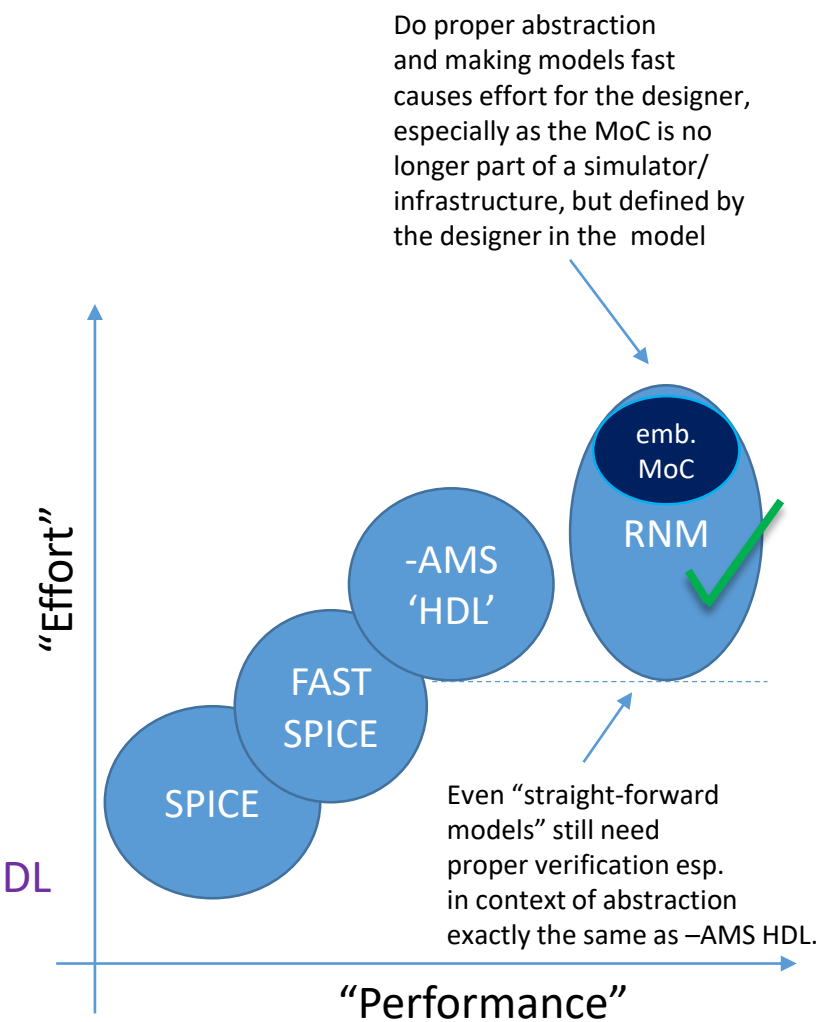


Simulation performance is not “for free”:

.) effort to set up (proper abstraction, explicit/implicit model definition) *if not done top-down from system/concept level*

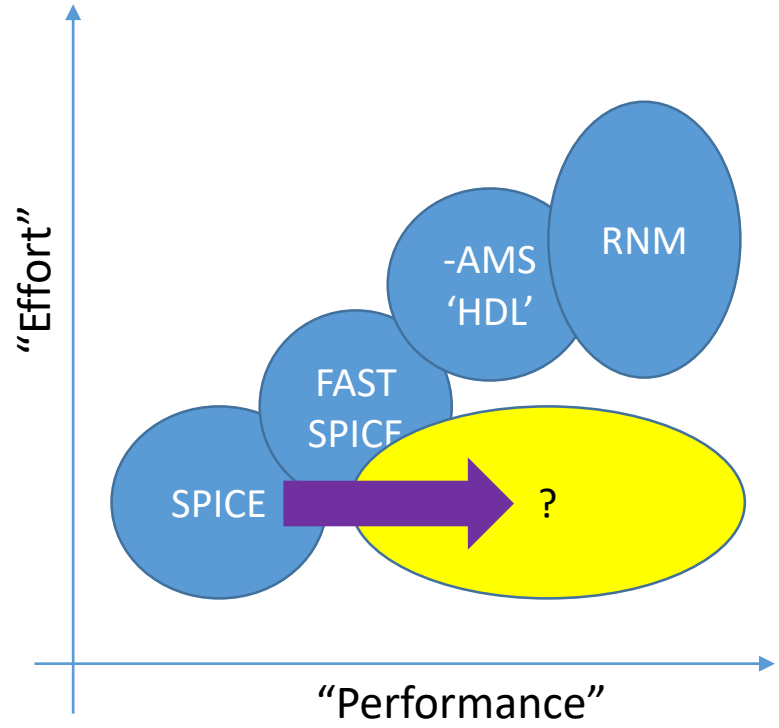
.) experience to write efficient models in a HDL *additional skill for an analog designer*

.) good verification and knowledge of model limitations within its context is important *additional skill for an analog verification engineer*

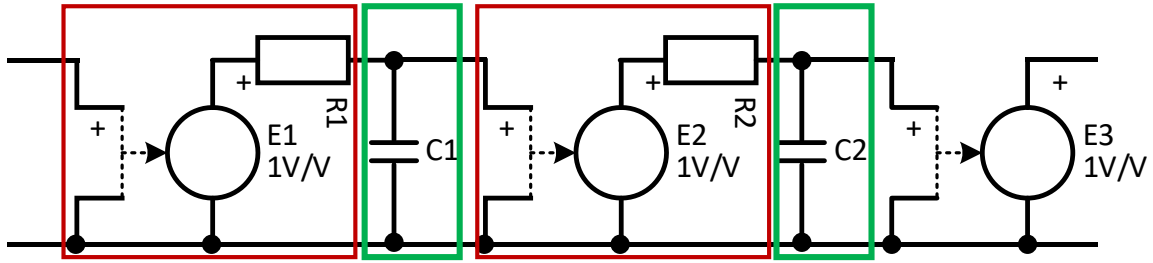


Beyond “state of the art” of RNM…

- Can we stick to a “more SPICE like level” **keeping modelling efforts low**, but still improve performance similar (or close to) to RNM?



The EEnet approach in SystemVerilog




```
`timescale 1ns/1ps
import EE_pkg::*;

// EEnet voltage controlled voltage source model ("to ground")
// Carinthia University of Applied Science

module EE_E(Pi,Po);
inout EEnet Pi;          // positve control input
inout EEnet Po;          // positive source output

parameter real gain=1.0; // voltage gain [V/V]
parameter real rout=1.0; // output resistance [Ohm]
parameter real fs=1e6;   // sampling frequency [Hertz]

// internal variables
real Ts=1/fs;            // integration (=sample) time
real Vnew=0.0;           // new output voltage

// update for each sample timepoint
always #(Ts*1s)  begin
  Vnew = gain*Pi.V;
end

// update EEnet node
assign Po = '{Vnew,0,rout};

endmodule
```

```
`timescale 1ns/1ps
import EE_pkg::*;

// EEnet capacitor model ("to ground")
// (c) 2020 Carinthia University of Applied Science

module EE_C(P);
inout EEnet P;
parameter real C=1e-9;   // capacitance [F]
parameter real fs=1e6;   // sampling frequency [Hz]

// internal variables
real Ts=1/fs;            // integration (=sample) time
real Ic=0;               // charge current
real Vnet=0,Rnet=Ts/C;   // set values to EEnet
real Vlast=0;            // store last node voltage

// update for each sample timepoint
always #(Ts*1s)  begin
  // new integration step
  Ic = (Vlast-Vnet)/Rnet;
  Vnet = P.V + Ic*Rnet;
  // remember node voltage
  Vlast = P.V;
end

// update EEnet node
assign P = '{Vnet,0,Rnet};

endmodule
```

Be aware:

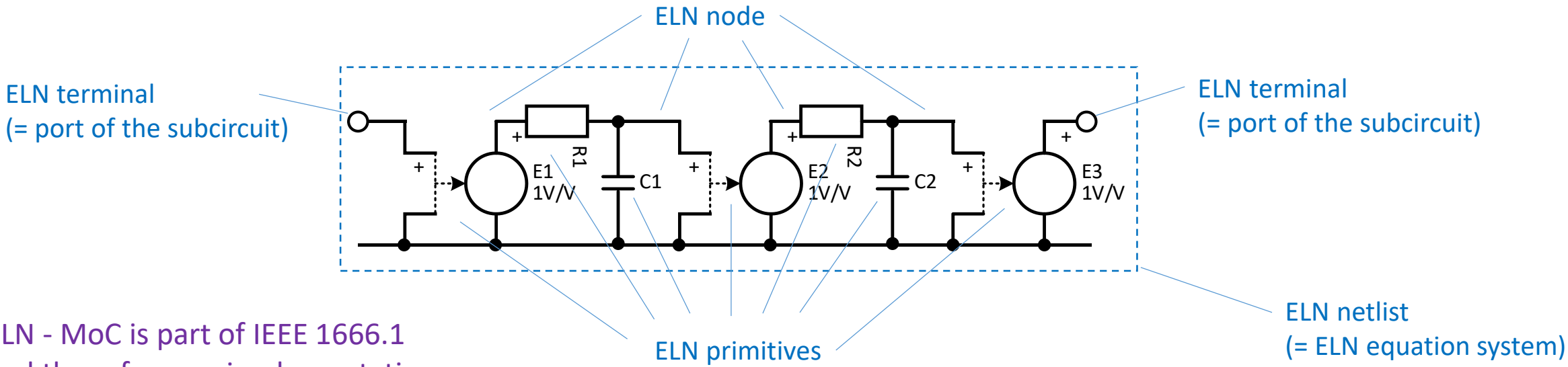
.) still an abstraction (or kind of translation) step to re-partition the SPICE model is needed as well as a c.t. -> d.t. transform (similar RNM).

.) MoC is still (partially) in the hand of the designer.

.) separating the series-R requires to implement an iterative solver in the model → **inefficient**!

Based on a Cadence EEnet tutorial and ideas from:
"Enabling Digital Mixed-Signal Verification of Loading Effects in Power Regulation using SystemVerilog User-Defined Nettype", A. Caicedo et.al., DVCON 2019
https://dvcon-europe.org/sites/dvcon-europe.org/files/08_2_Caicedo_finalpaper_09_16_2019_15_52.pdf

SystemC-AMS enables again simple schematics entry + netlisting…



ELN - MoC is part of IEEE 1666.1
and the reference implementation.
→ designer focuses on the model again!

Table 4.1—ELN primitive modules *)

ELN module name	Description
sca_eln::sca_r	Resistor
sca_eln::sca_c	Capacitor
sca_eln::sca_l	Inductor
sca_eln::sca_vcvs	Voltage controlled voltage source

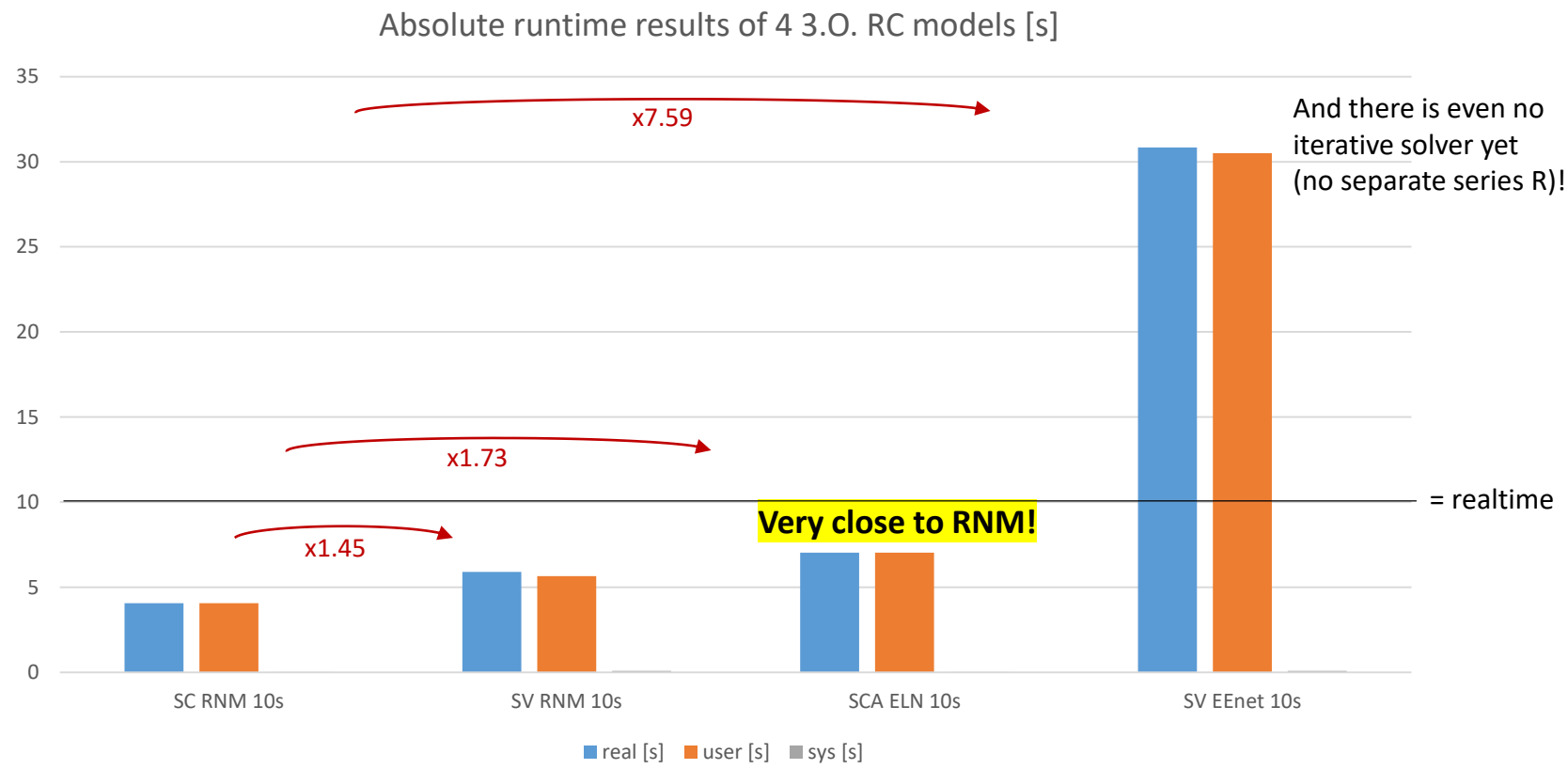
…and many more

**But SPICE is slow!
And now you tell
us to use
something like
SPICE ?!?!?!?!**

*) Source:
"**SystemC Analog/Mixed-Signal User's Guide User Perspective on IEEE Std. 1666.1-2016"**, Jan. 2020
https://www.accellera.org/images/downloads/standards/systemc/Accellera_SystemC_AMS_Users_Guide_January_2020.pdf

What about performance ?

- SystemC-AMS easily outperforms SystemVerilog EEnet

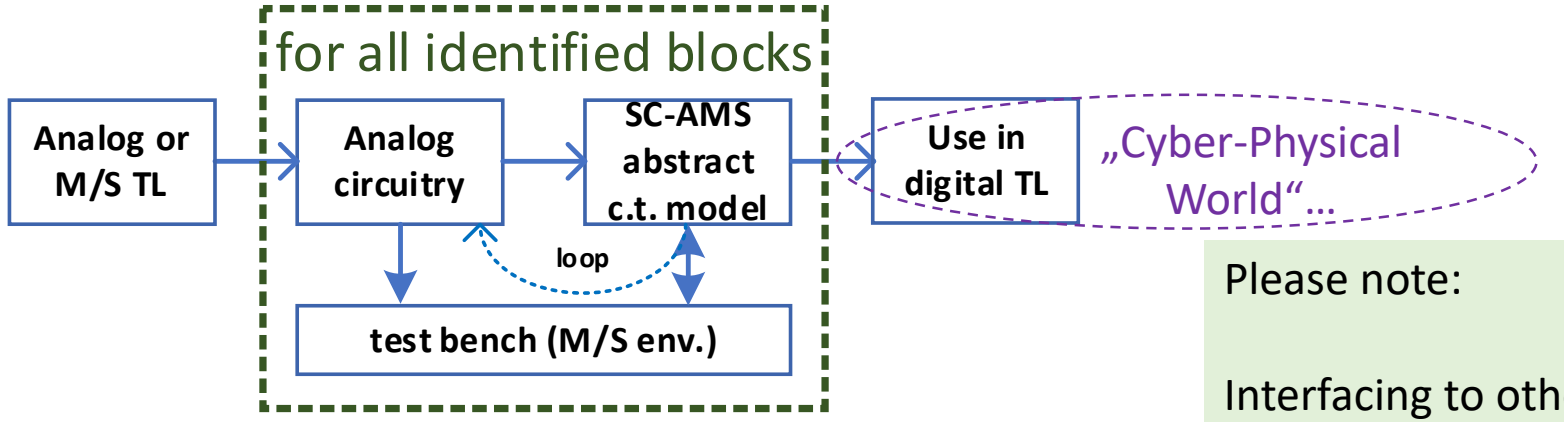


And there is even no iterative solver yet (no separate series R)!

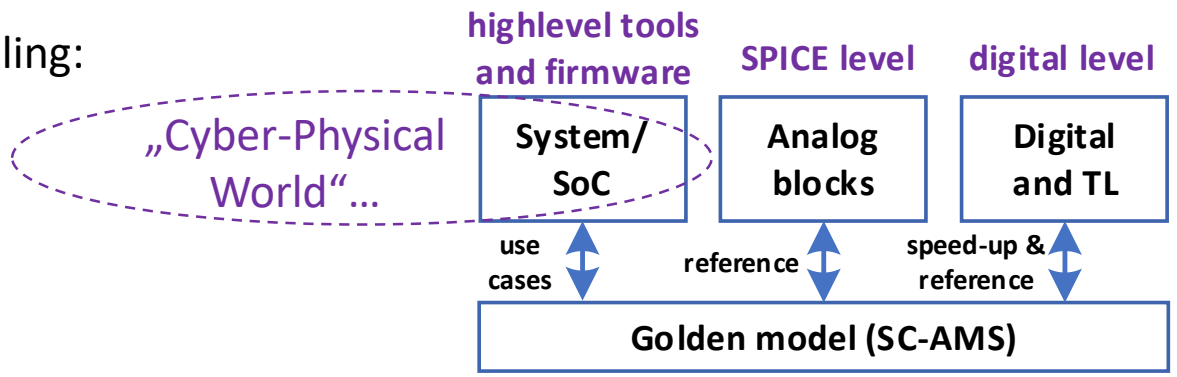
SystemC-AMS allows analog designers to do what they can do best…

- Simplified analog modeling flow:

„Bottom up“ modeling designs:



„Top-down“ modeling:

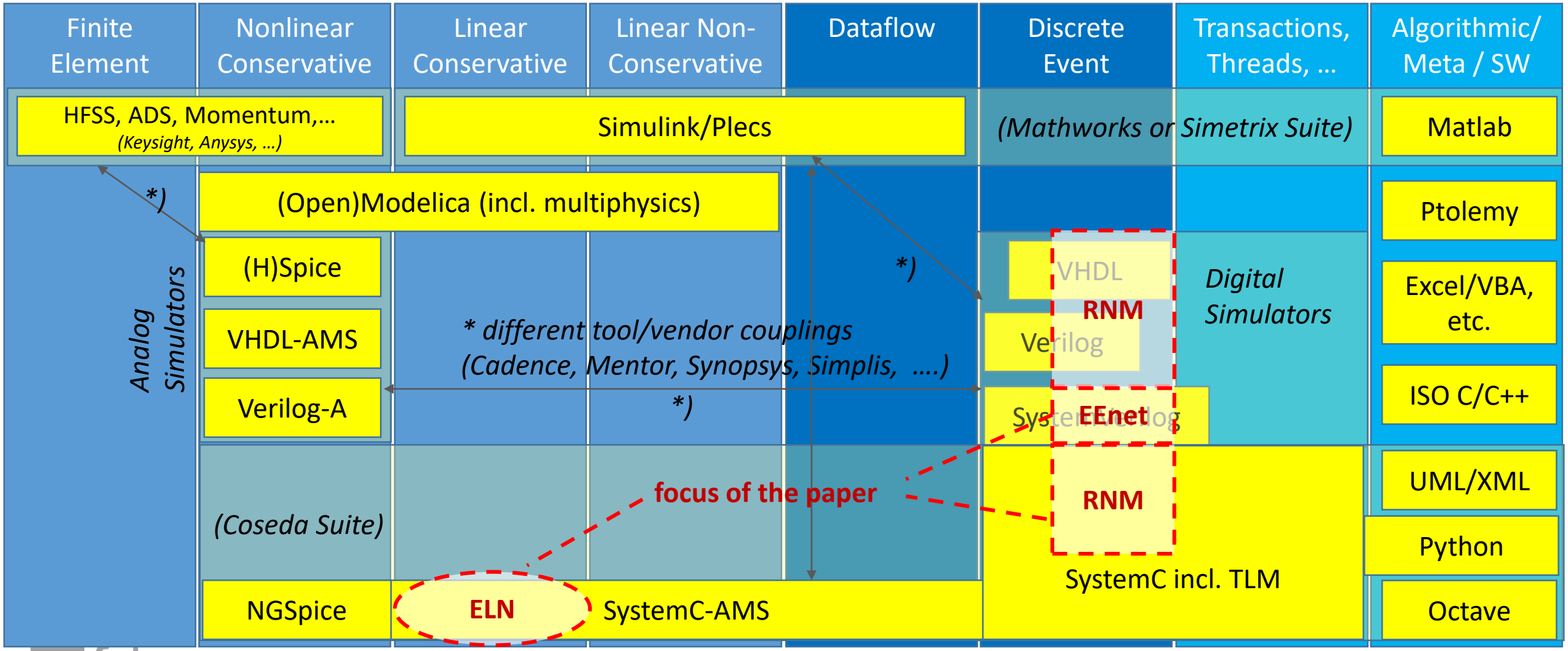


Please note:

Interfacing to other tools/languages exist (e.g. SPICE or dig. simulator) but is not well standardized yet.

Might be nice to have FMI etc. for standardized interfaces…?

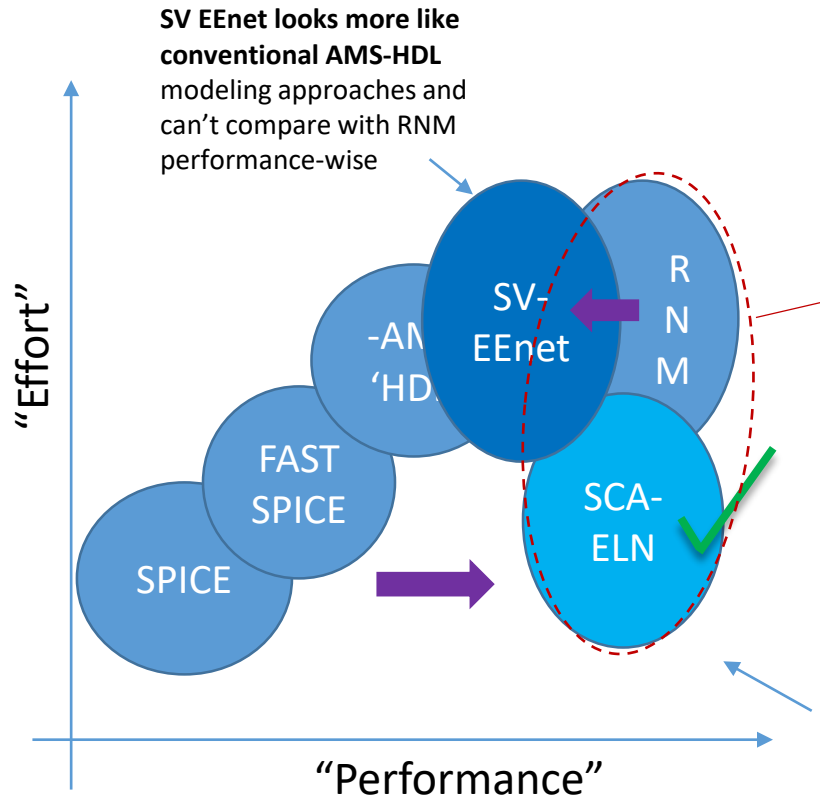
Conclusion I: SystemC-AMS bridges gaps



Please note: qualitative, not exhaustive and simplified overview. There are many, many more tools and languages out there, these are just representing some most common ones!

Conclusion II: effort and performance

- SystemC-AMS can improve analog modeling significantly
  → in terms of effort and performance !



Please note:

IEEE 1666 / 1666.1
allows to use (SC-)RNM
together with SCA-ELN
and other standard MoCs
to find a **good balance between performance, abstraction, modelling effort**
(e.g. also nonlinear behavior).

SCA-ELN provides the possibility to directly
re-use (linear) SPICE models in a fast, digital environment
close to RNM performance, given a stable reference implementation
these models created from SPICE could be even "**correct by construction**"…
(But I always advice to verify the models!)