technische universität dortmund

fakultät für informatik
informatik 12

Comparison of models

Peter Marwedel
Informatik 12,
TU Dortmund,
Germany

2010/11/07

Models of computation considered in this course

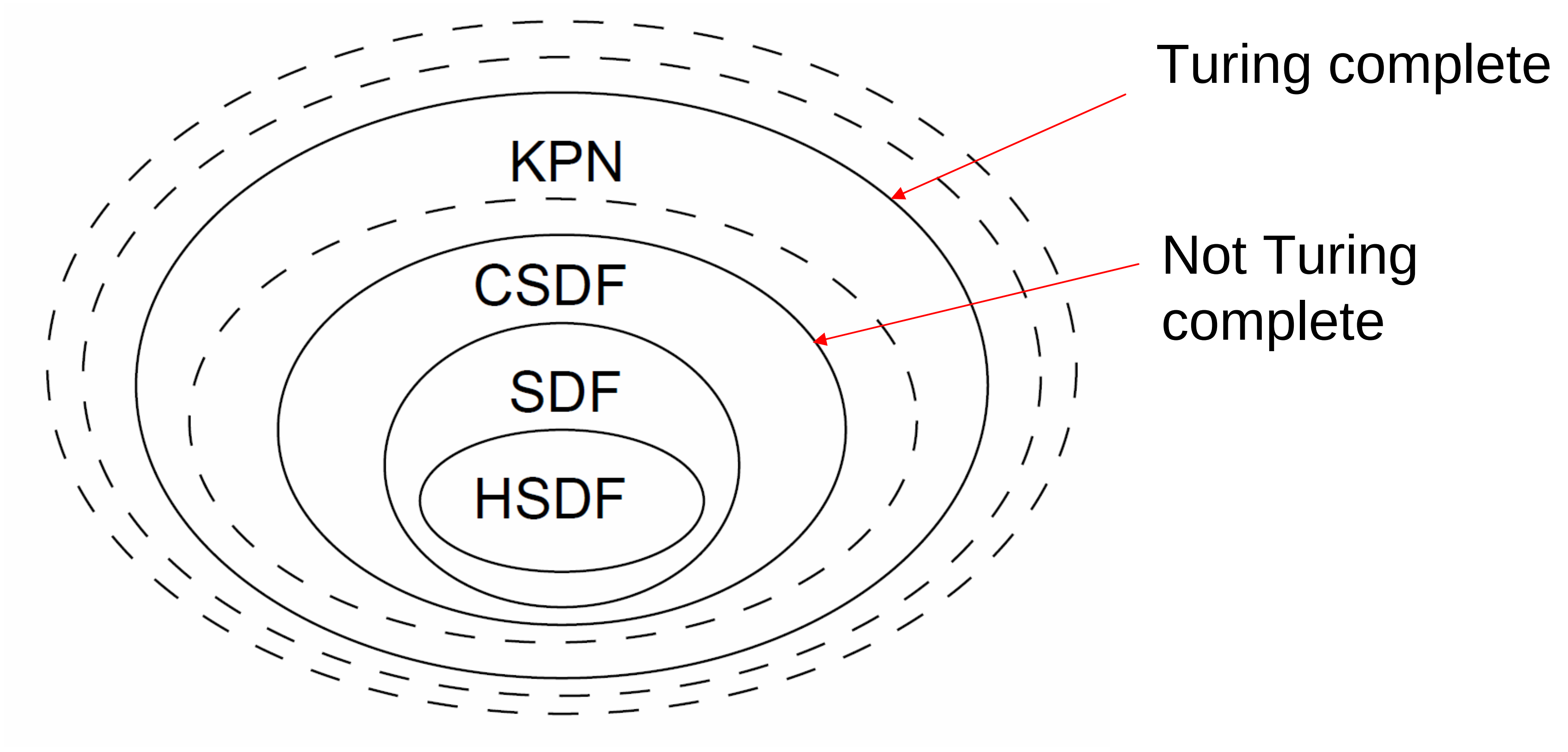
Communication/ local computations	**Shared memory**	**Message passing**	
		Synchronous	**Asynchronous**
Undefined components	Plain text, use cases	(Message) sequence charts	
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL*, Verilog*, SystemC*, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Imperative (Von-Neumann) model	C, C++, Java	C, C++, Java with libraries CSP, ADA	

* Classification based on the **implementation** of HDLs

Classification by Stuijk

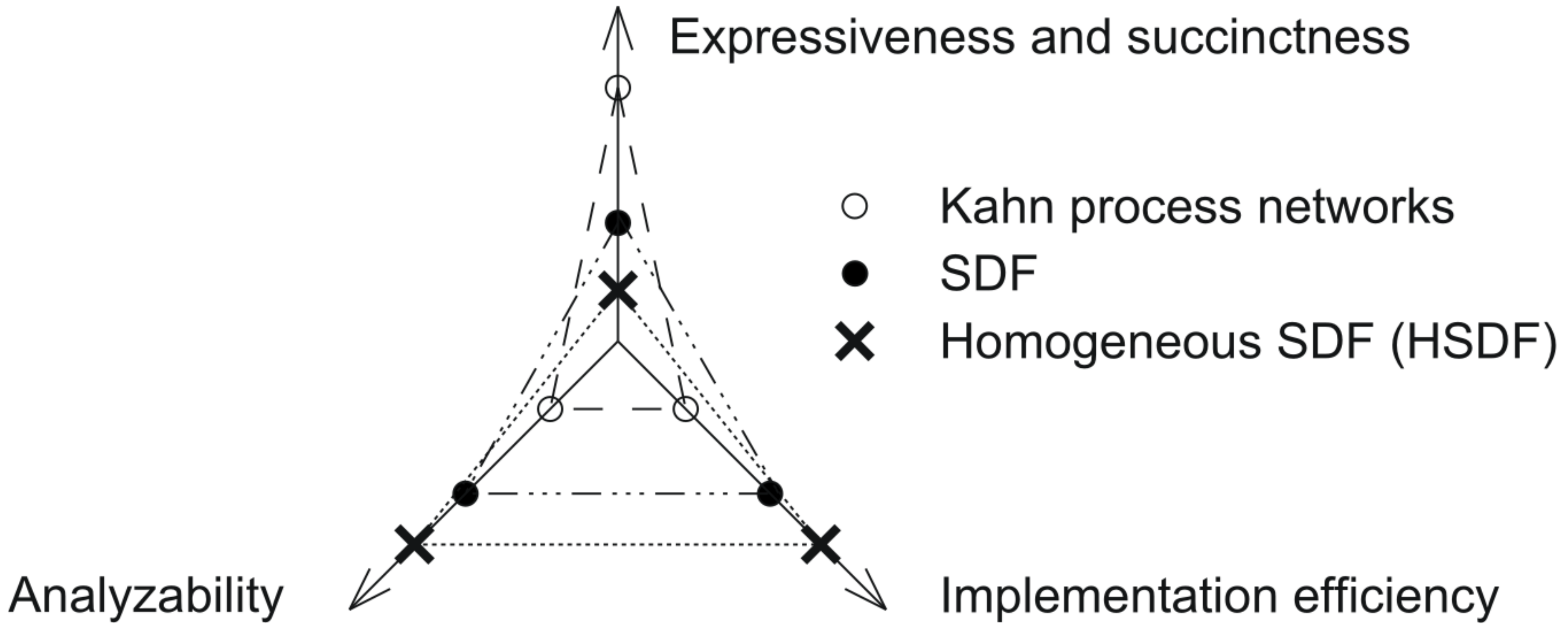
- **Expressiveness** and **succinctness** indicate, which systems can be modeled and how compact the are.
- **Analyzability** relates to the availability of scheduling algorithms and the need for run-time support.
- **Implementation** efficiency is influenced by the required scheduling policy and the code size.

Expressiveness of data flow models



[S. Stuijk, 2007]

Classification by Stuijk (2)

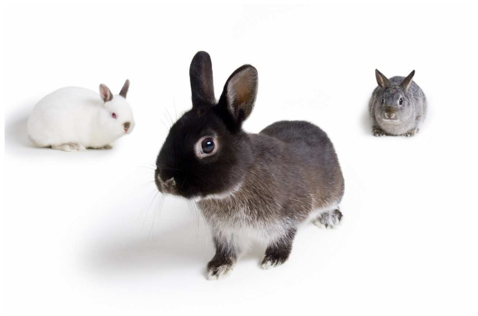


KPN very expressive, but difficult to analyze

[S. Stuijk, 2007]

Properties of processes (1)

- **Number of processes**
  static;
  dynamic (dynamically changed hardware architecture?)



- **Nesting:**
  - Nested declaration of processes

```
process {
  process {
    process {
}}}
```

  - or all declared at the same level

```
process { … }
process { … }
process { … }
```

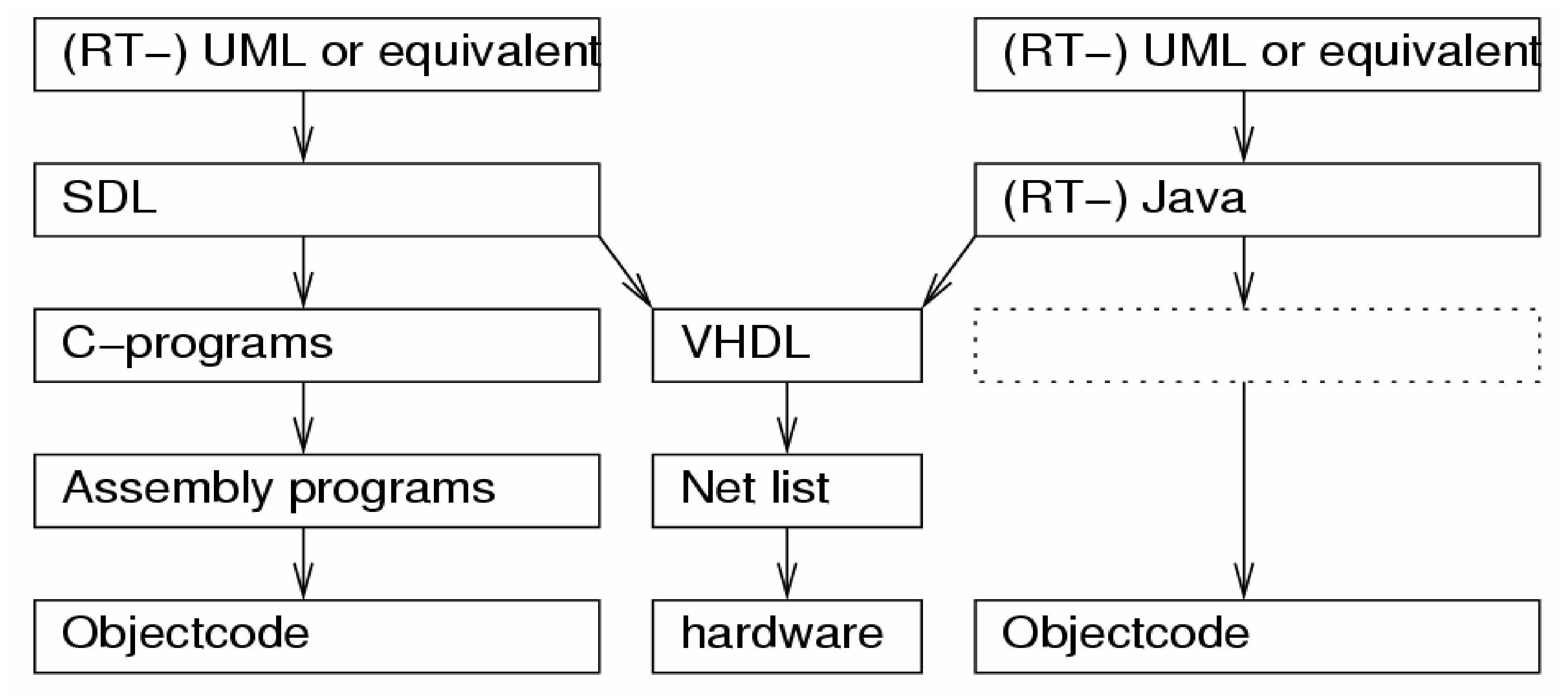
Properties of processes (2)

- Different techniques for **process creation**
  - **Elaboration in the source (c.f. ADA)**
```
declare
  process P1 …
```
  - **explicit fork and join (c.f. Unix)**
```
id = fork();
```
  - **process creation calls**
```
id = create_process(P1);
```

E.g.: StateCharts comprises a static number of processes, nested declaration of processes, and process creation through elaboration in the source.

How to cope with MoC and language problems in practice?

Mixed approaches:



Mixing models may require formal models of MoCs

Mixing models of computation: Ptolemy

Ptolemy (UC Berkeley) is an environment for simulating multiple models of computation.

http://ptolemy.berkeley.edu/

Code generation not available for all models

Mixing MoCs: Ptolemy
(Focus on executable models; "mature" models only)

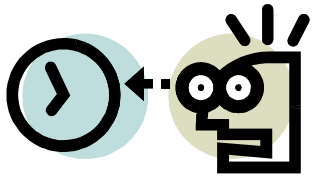
Communication/ local computations	**Shared memory**	**Message passing Synchronous | Asynchronous**
Undefined components		
Communicating finite state machines	FSM, synchronous/reactive MoC	
Data flow		Kahn networks, SDF, dynamic dataflow, discrete time
Petri nets		
Discrete event (DE) model	DE	Experimental distributed DE
Von Neumann model		CSP |
Wireless	Special model for wireless communication	
Continuous time	Partial differential equations	

Mixing models of computation: UML
(Focus on support of early design phases)

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Undefined components	*use cases*	*| sequence charts, timing diagrams*	
Communicating finite state machines	State diagrams		
Data flow	(Not useful)	Data flow	
Petri nets		activity charts	
Discrete event (DE) model	-	-	
Von Neumann model	-	-	

UML for embedded systems?

Initially not designed for real-time.



Initially lacking features:

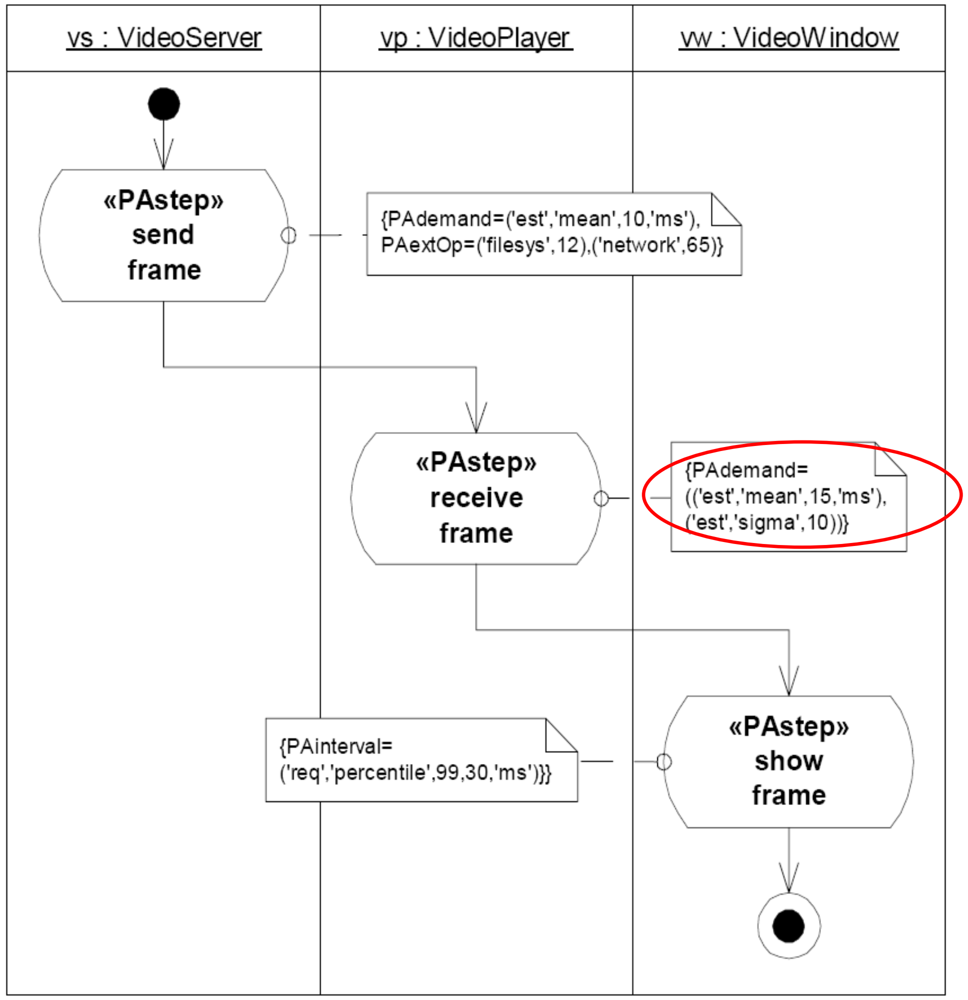
- Partitioning of software into tasks and processes
- specifying timing
- specification of hardware components

Projects on defining profiles for embedded/real-time systems

- Schedulability, Performance and Timing Analysis
- SysML (System Modeling Language)
- UML Profile for SoC
- Modeling and Analysis of Real-Time Embedded Systems
- UML/SystemC, …

Profiles may be incompatible

Example: Activity diagram with annotations



vs : VideoServer
vp : VideoPlayer
vw : VideoWindow
«PAstep» send frame
{PAdemand=('est','mean',10,'ms'), PAextOp=('filesys',12),('network',65)}
«PAstep» receive frame
{PAdemand= (('est','mean',15,'ms'), ('est','sigma',10))}
{PAinterval= ('req','percentile',99,30,'ms')}}
«PAstep» show frame

Figure 8-10 Details of the "send video" subactivity with performance annotations

[http://www.omg.org/cgi-bin/doc?ptc/2002-03-02]

See also W. Müller et al.: UML for SoC, http://jerry.c-lab.de/uml-soc/

Modeling levels

Levels, at which modeling can be done:

- Algorithmic level: just the algorithm
- Processor/memory/switch (PMS) level
- Instruction set architecture (ISA) level: function only
- Transaction level modeling (TML): memory reads & writes are just “transactions“ (not cycle accurate)
- Register-transfer level: registers, muxes, adders, … (cycle accurate, bit accurate)
- Gate-level: gates
- Layout level

Tradeoff between accuracy and simulation speed

System level

- Term not clearly defined.
- Here: denotes the entire embedded system,
  system into which information processing is embedded, and possibly also the environment.
- Models may include mechanics + information processing.
  May be difficult to find appropriate simulators.
  Solutions: VHDL-AMS, SystemC or MATLAB.
  MATLAB+VHDL-AMS support partial differential equations.
- Challenge to model information processing parts of the system such that the simulation model can be used for the synthesis of the embedded system.

Algorithmic level

- Simulating the algorithms that we intend to use within the embedded system.
- No reference is made to processors or instruction sets.
- Data types may still allow a higher precision than the final implementation.
- If data types have been selected such that every bit corresponds to exactly one bit in the final implementation, the model is said to be **bit-true**.
  non-bit-true $\rightarrow$ bit-true should be done with tool support.
- Single process or sets of cooperating processes.

Algorithmic level: Example:
-MPEG-4 full motion search -

```
for (z=0; z<20; z++)
 for (x=0; x<36; x++) {x1=4*x;
  for (y=0; y<49; y++) {y1=4*y;
   for (k=0; k<9; k++) {x2=x1+k-4;
    for (l=0; l<9; ) {y2=y1+l-4;
     for (i=0; i<4; i++) {x3=x1+i; x4=x2+i;
      for (j=0; j<4;j++) {y3=y1+j; y4=y2+j;
       if (x3<0 || 35<x3||y3<0||48<y3)
        then_block_1; else else_block_1;
       if (x4<0|| 35<x4||y4<0||48<y4)
        then_block_2; else else_block_2;
}}}}}}
```

Instruction level

Algorithms already compiled for the instruction set.
Model allows counting the executed number of instructions.

Variations:

- Simulation only of the effect of instructions
- **Transaction-level modeling**: each read/write is one transaction, instead of a set of signal assignments
- **Cycle-true simulations**: exact number of cycles
- **Bit-true simulations:** simulations using exactly the correct number of bits

Instruction level: example

Assembler (MIPS)	Simulated semantics
`and $1,$2,$3`	**`Reg[1]:=Reg[2]`** $\wedge$ **`Reg[3]`**
`or $1,$2,$3`	**`Reg[1]:=Reg[2]`** $\vee$ **`Reg[3]`**
`andi $1,$2,100`	**`Reg[1]:=Reg[2]`** $\wedge$ **`100`**
`sll $1,$2,10`	**`Reg[1]:=Reg[2] << 10`**
`srl $1,$2,10`	**`Reg[1]:=Reg[2] >> 10`**

Register transfer level (RTL)

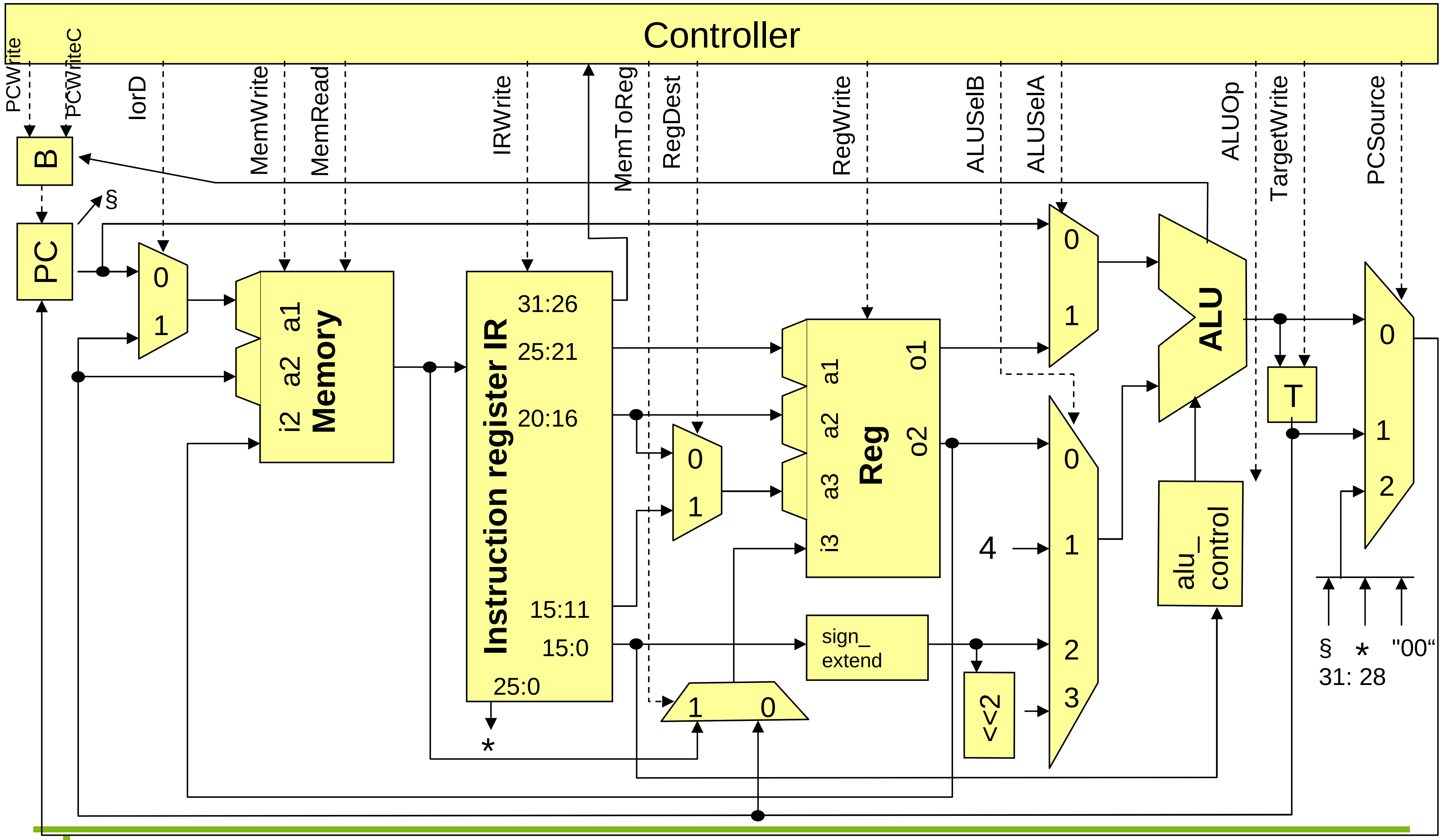
Modelling of all components at the register-transfer level, including

- arithmetic/logic units (ALUs),
- registers,
- memories,
- muxes and
- decoders.

Models at this level are always cycle-true.

Automatic synthesis from such models is **not** a major challenge.

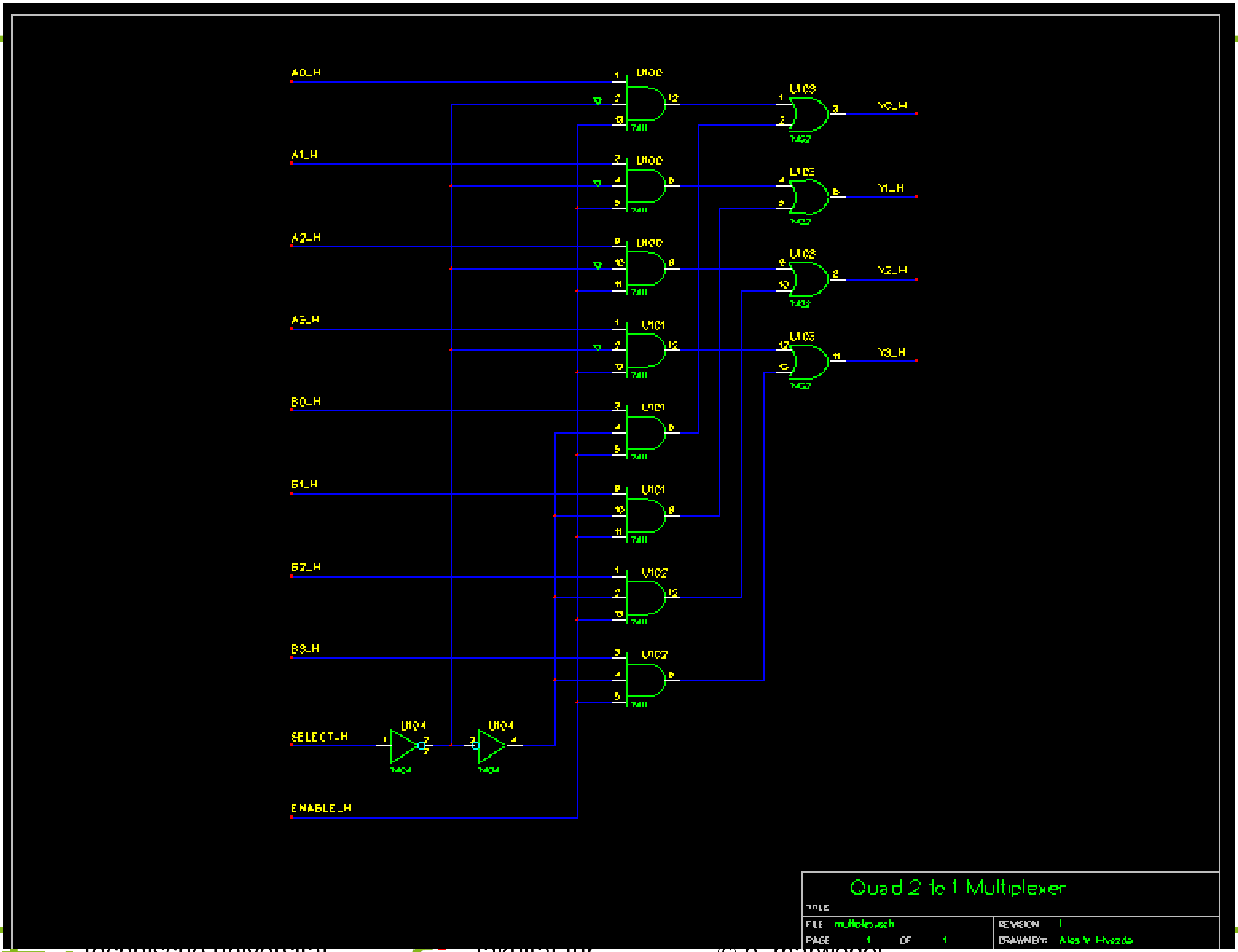
Register transfer level: example (MIPS)



Gate-level models

- Models contain gates as the basic components.
- Information about signal transition probabilities
  ☞ can be used for power estimations.
- Delay calculations can be more precise than for RTL.
  Typically no information about the length of wires
  (still estimates).
- Term sometimes also denotes Boolean functions
  (No physical gates; only considering the behavior of the gates).
  Such models should be called “Boolean function models”.

Gate-level models: Example

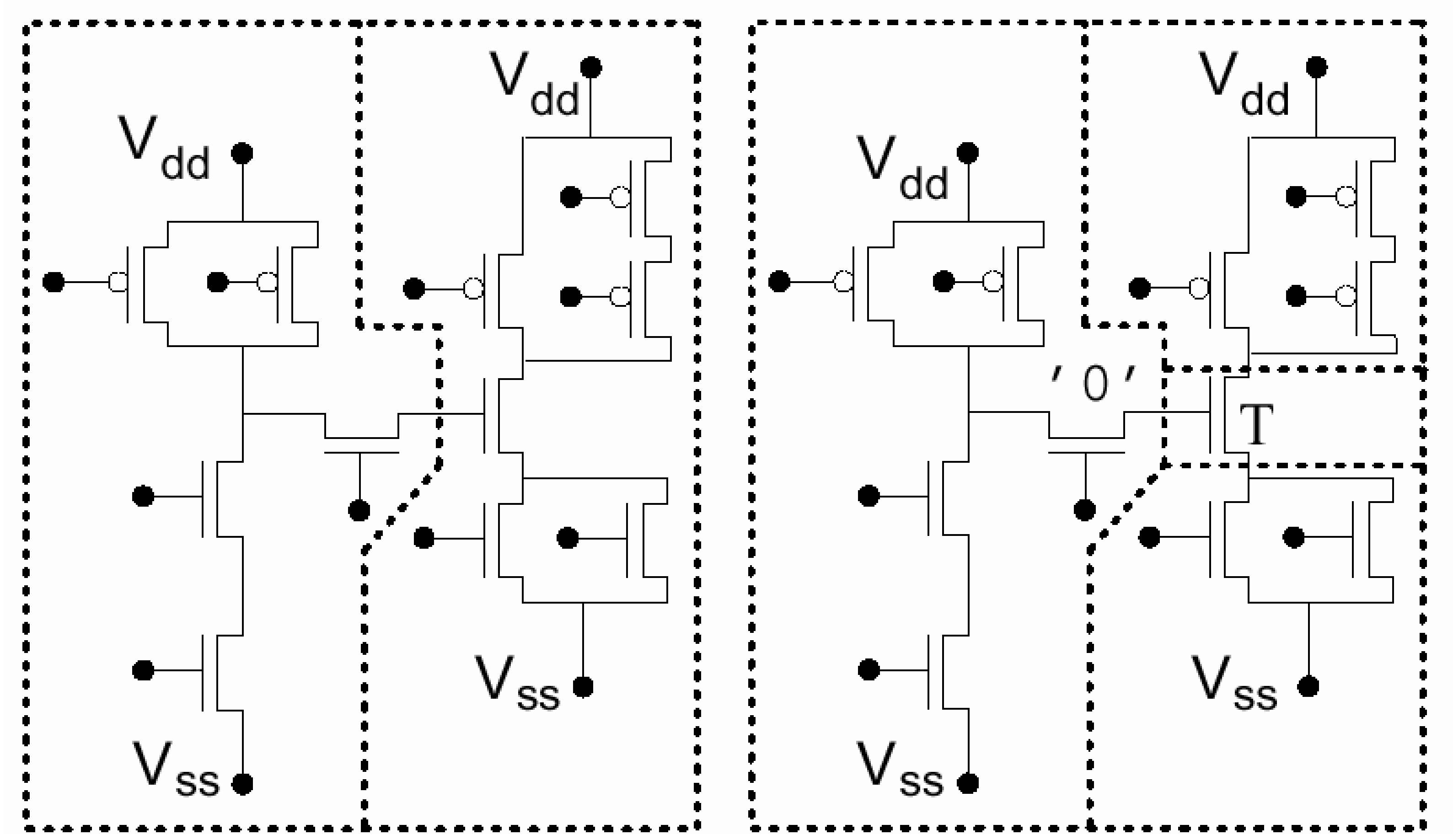


source: http://geda.seul.org/screenshots/screenshot-schem2.png

Switch-level models

- Switch level models use switches (transistors) as their basic components.
- Switch level models use digital values models.
- In contrast to gate-level models, switch level models are capable of reflecting **bidirectional** transfer of information.

Switch level model: example

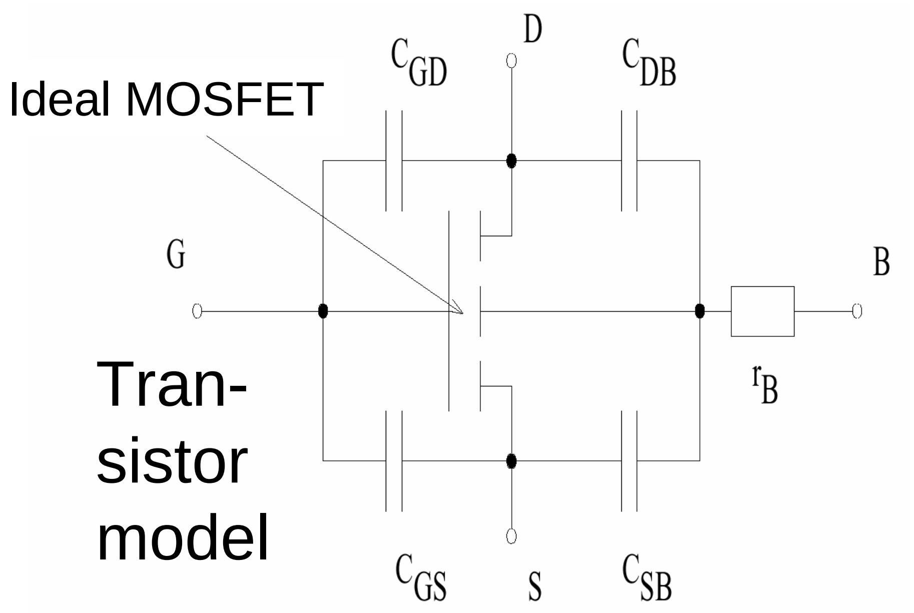


Source: http://vada1.skku.ac.kr/ClassInfo/ic/vlsicad/chap-10.pdf

Circuit level models: Example

- Models circuit theory. Its components (current and voltage sources, resistors, capacitances, inductances and possibly macro-models of semiconductors) form the basis of simulations at this level.

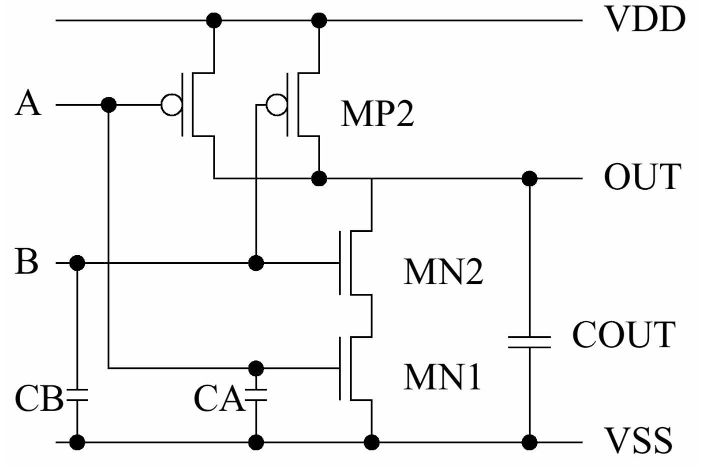
Simulations involve partial differential equations.
Linear if and only if the behavior of semiconductors is linearized.

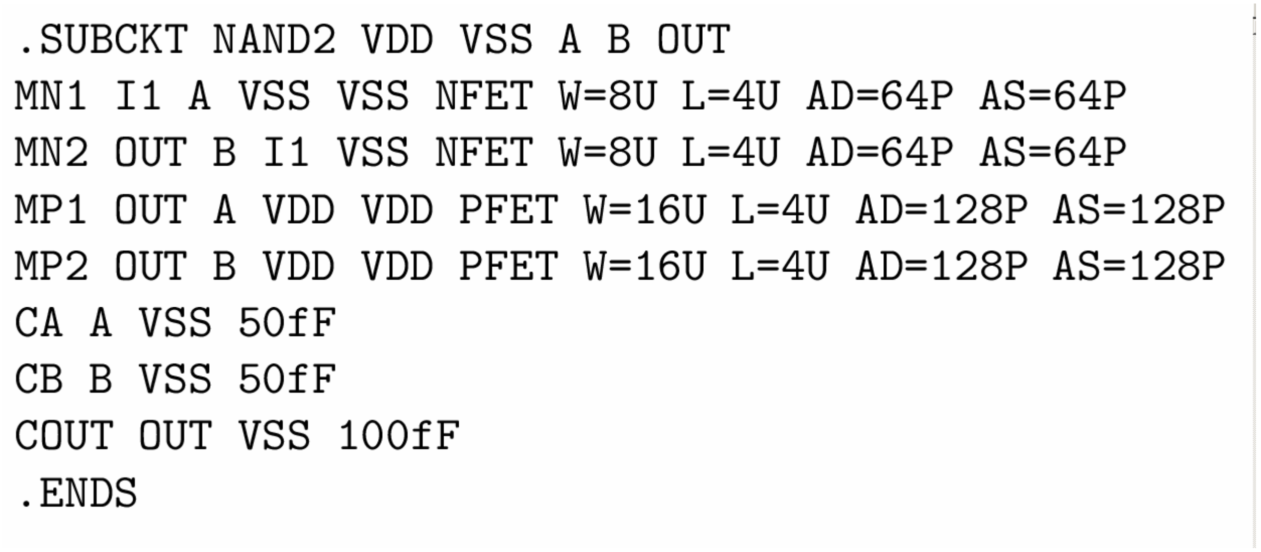


Circuit level models: SPICE

The most frequently used simulator at this level is SPICE [Vladimirescu, 1987] and its variants.

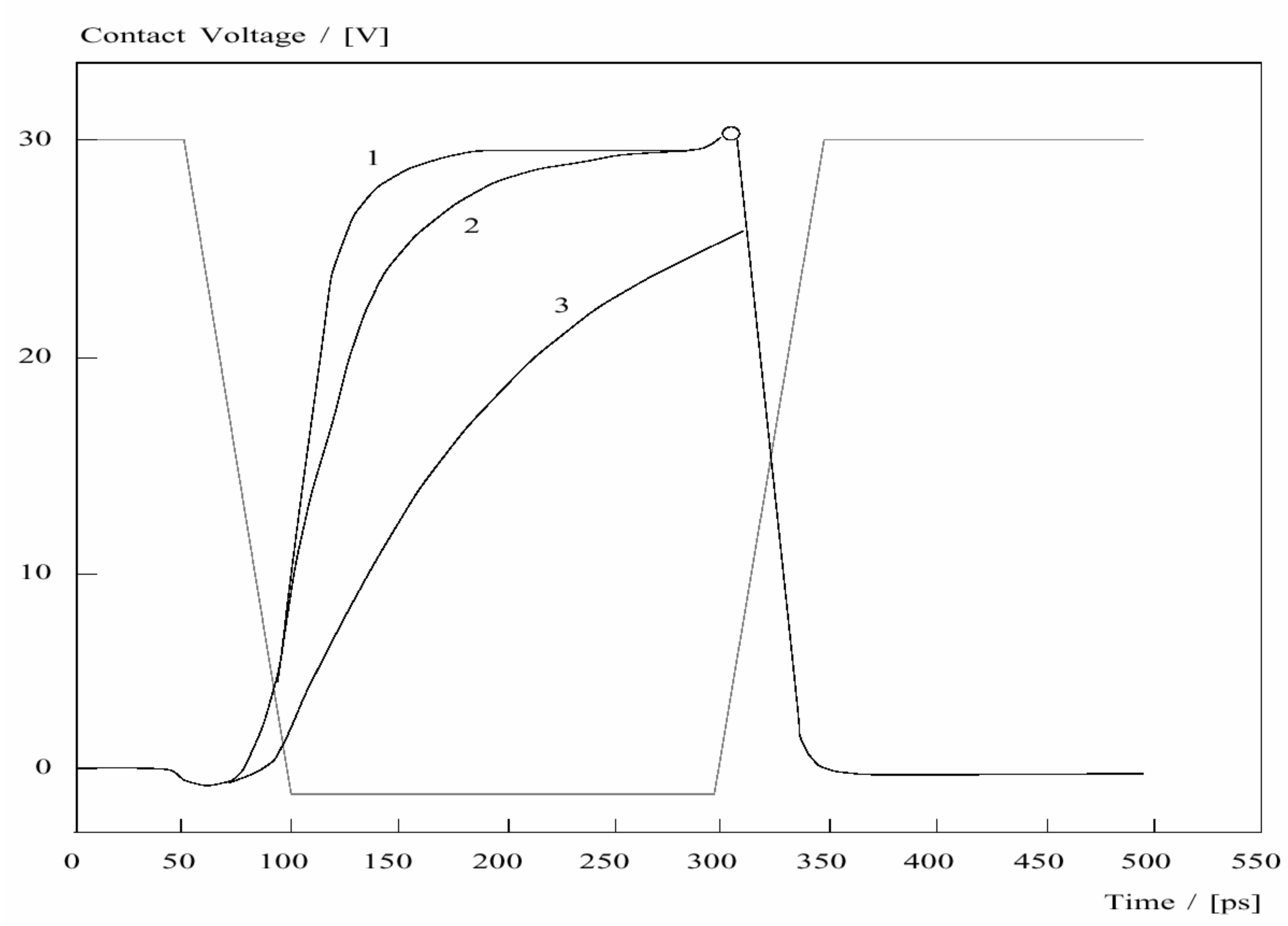
Example:





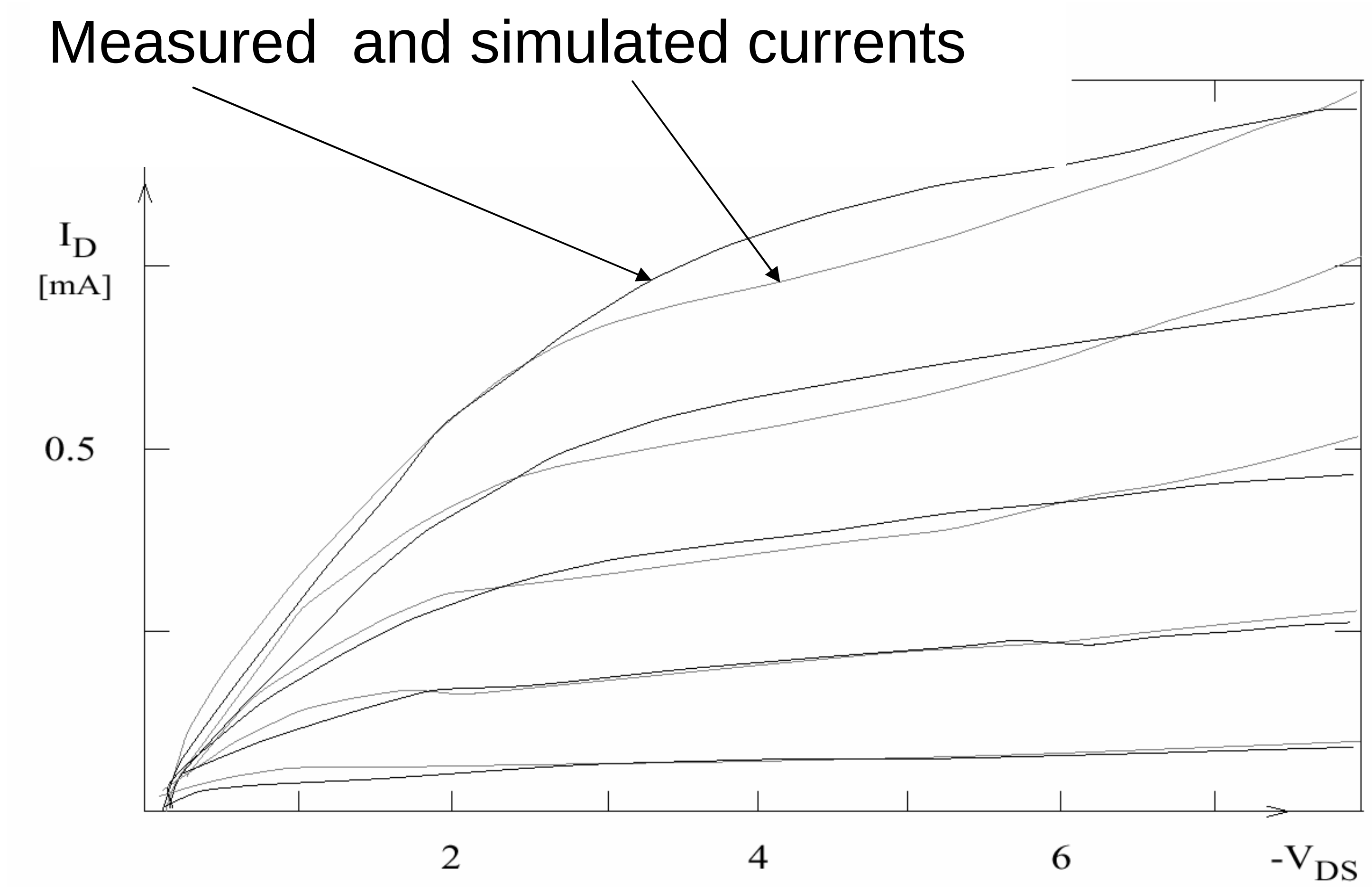
```
.SUBCKT NAND2 VDD VSS A B OUT
MN1 I1 A VSS VSS NFET W=8U L=4U AD=64P AS=64P
MN2 OUT B I1 VSS NFET W=8U L=4U AD=64P AS=64P
MP1 OUT A VDD VDD PFET W=16U L=4U AD=128P AS=128P
MP2 OUT B VDD VDD PFET W=16U L=4U AD=128P AS=128P
CA A VSS 50fF
CB B VSS 50fF
COUT OUT VSS 100fF
.ENDS
```

Circuit level models: sample simulation results



Device level

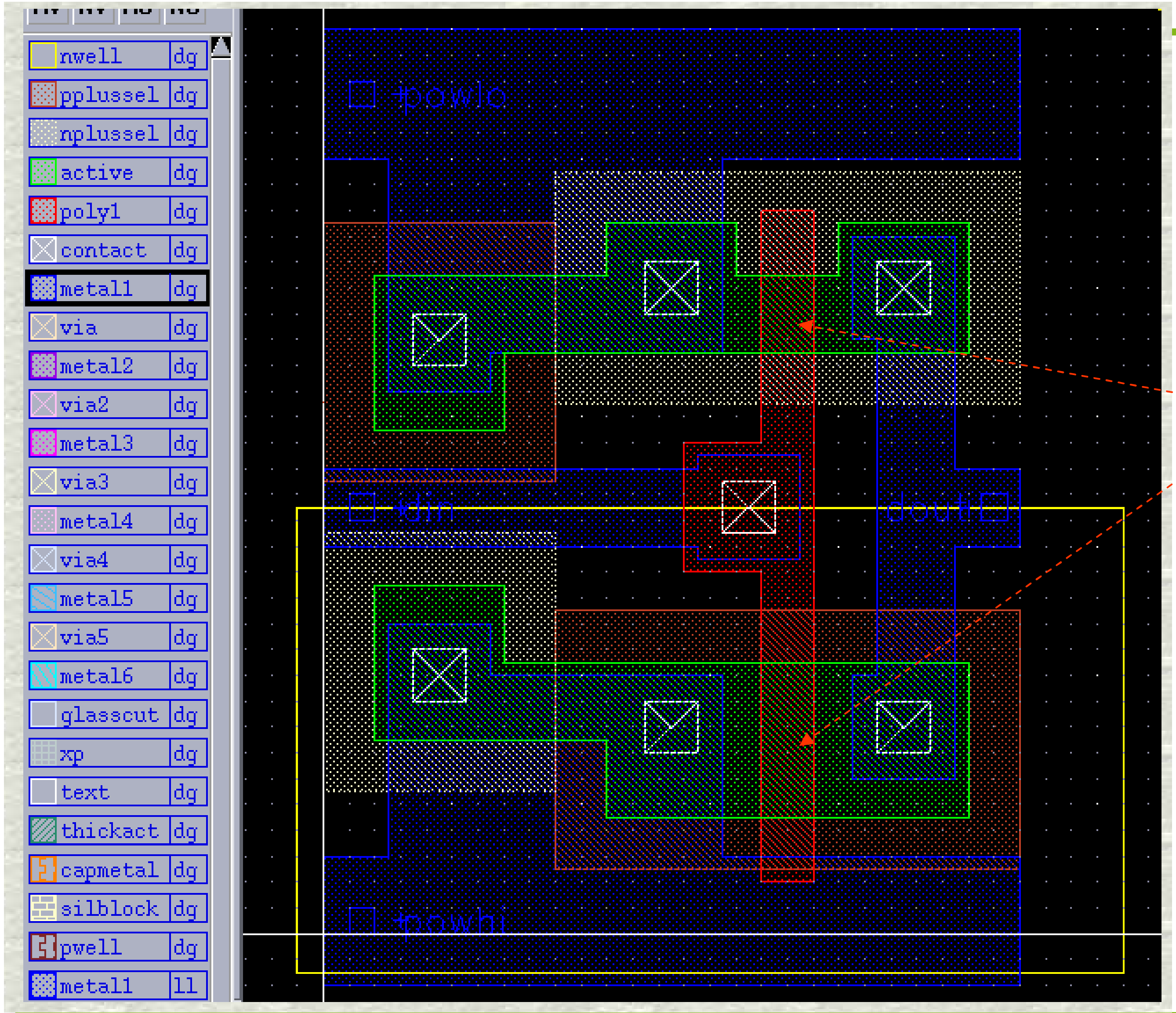
Simulation of a single device (such as a transistor). Example (SPICE-simulation [IMEC]):

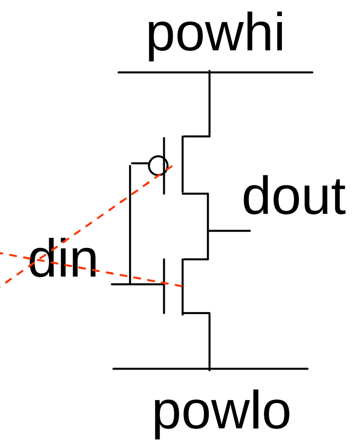


Layout models

- Reflect the actual circuit layout,
- include **geometric** information,
- cannot be simulated directly:
  behavior can be deduced by correlating the layout model with a behavioral description at a higher level or by extracting circuits from the layout.
- Length of wires and capacitances frequently extracted from the layout,
  back-annotated to descriptions at higher levels
  (more precision for delay and power estimations).

Layout models: Example

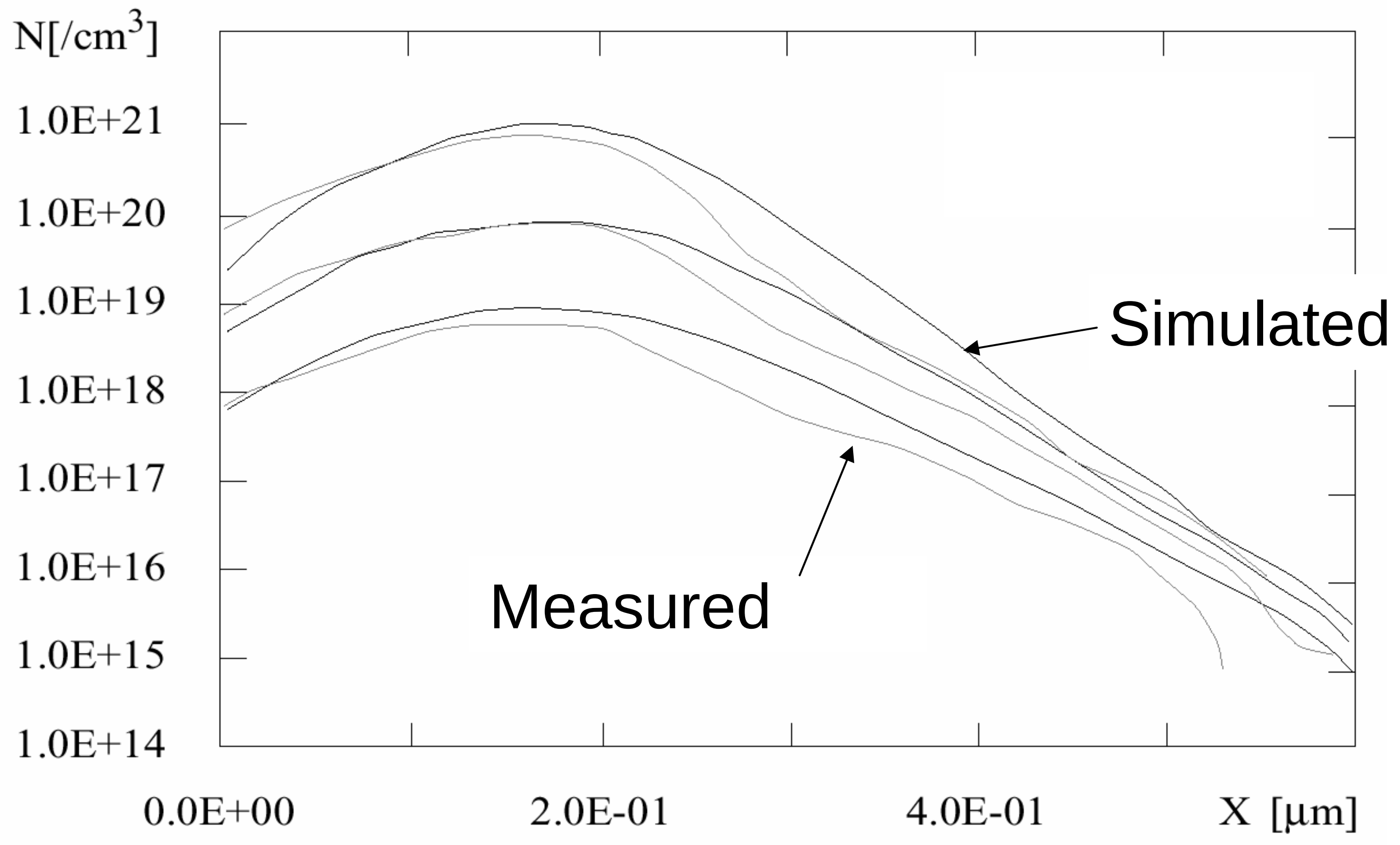




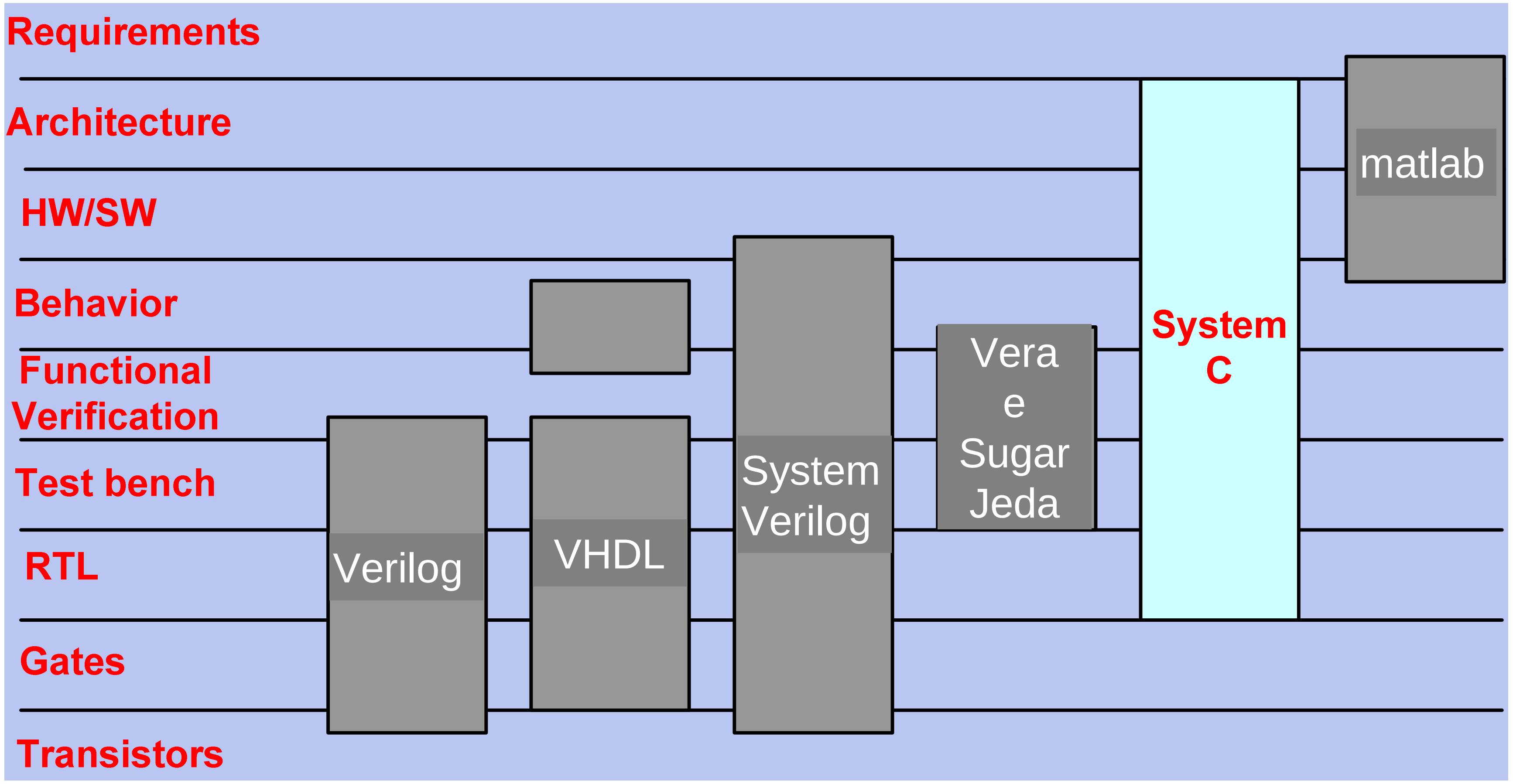
© Mosis (http://www.mosis.org/Technical/Designsupport/polyflowC.html);
Tool: Cadence

Process models

Model of fabrication process; Example [IMEC]:
Doping as a function of the distance from the surface

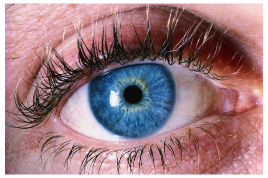


Levels covered by the different languages



What‘s the bottom line?

- The prevailing technique for writing embedded SW has inherent problems; some of the difficulties of writing embedded SW are not resulting from design constraints, but from the modeling.
- However, there is no ideal modeling technique.
- The choice of the technique depends on the application.
- Check code generation from non-imperative models
- There is a tradeoff between the power of a modeling technique and its analyzability.
- It may be necessary to combine modeling techniques.
- **In any case, open your eyes & think about the model before you write down your spec! Be aware of pitfalls.**
- You may be forced to use imperative models, but you can still implement, for example, finite state machines or KPNs in Java.



Summary

- Comparison of models
  - Expressiveness vs. analyzability
  - Process creation
  - Different modeling levels
  - Mixing models of computation
    - Ptolemy & UML
    - Using FSM and KPN models in imperative languages, etc.