

System Modeling

Models of Computation and their Applications

Axel Jantsch

Laboratory for Electronics and Computer Systems (LECS)

Royal Institute of Technology, Stockholm, Sweden

January 28, 2008



Contents

- Introduction
- Finite State Machines
- Petri Nets
- The Untimed Model of Computation
- The Synchronous Model of Computation
- The Timed Model of Computation
- Integration of Different Computational Models
- Tightly Coupled Process Networks
- Nondeterminism and Probability
- Applications



Introduction

- Motivation
- System and Model Properties
- Rugby Meta Model
- Case Study: A Design Project



Why do we need models? To perform various design tasks!

- Performance modeling
- Functional modeling and specification
- Design and synthesis
- Validation and verification
- Test vector generation
- Test coverage analysis
- Architecture evaluation and mapping
- Technology mapping
- Placement and routing



What is a Model?

Model: *A model is a simplification of another entity, which can be a physical thing or another model. The model contains exactly those characteristics and properties of the modeled entity which are relevant for a given task. A model is *minimal* with respect to a task, if it does not contain any other characteristics than those relevant for the task.*

- A model relates to an entity
- A model is a simplification of that entity
- A model is related to a task and an objective
- A model may relate to a not yet existing entity

Properties of Models

- Inherent property:** The property is inherent in every model.
E.g. the finite state space of a finite state machine model.
- Static property:** The property can be statically evaluated.
E.g. the required memory of a finite state machine model.
- Dynamic property:** The property can only be dynamically evaluated. E.g. the required memory of a C program.

Heterogeneous Models are Necessary

- A system consists of different parts. E.g. data flow and control flow dominated parts.
- Different objectives apply for different parts. E.g. the system and its environment.
- Different parts are developed by different people and tools. E.g. HW and SW.

What is a System?

A **system** is

- “an aggregation or assemblage of things so combined by nature or man as to form an integral or complex whole”
[[Encyclopedia America](#)]
- “a regularly interacting or independent group of items forming a unified whole” [[Webster's Dictionary](#)]
- “a combination of components that act together to perform a function not possible with any of the individual parts”
[[IEEE Standard Dictionary of Electrical and Electronic Terms](#)]



The Input-Output Modeling Process

$$\vec{u}(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_p(t) \end{bmatrix} \quad \vec{y}(t) = \begin{bmatrix} y_1(t) \\ \vdots \\ y_m(t) \end{bmatrix}$$

$$y_1(t) = f_1(u_1(t), \dots, u_p(t))$$

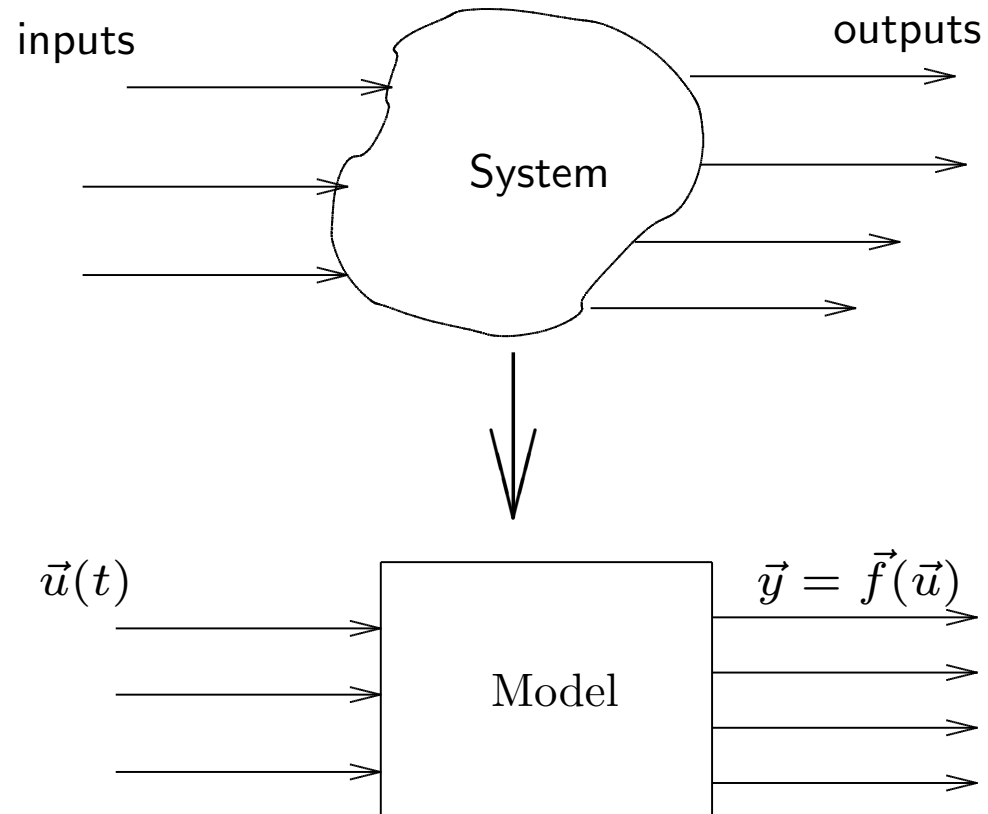
$$\vdots$$

$$y_m(t) = f_m(u_1(t), \dots, u_p(t))$$

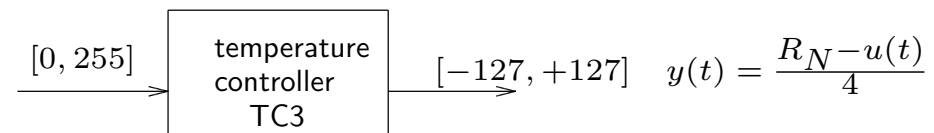
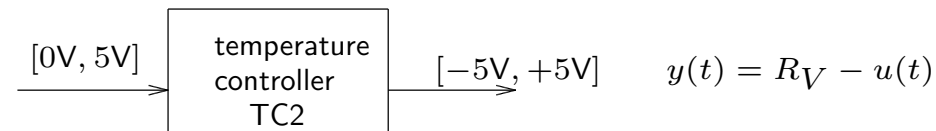
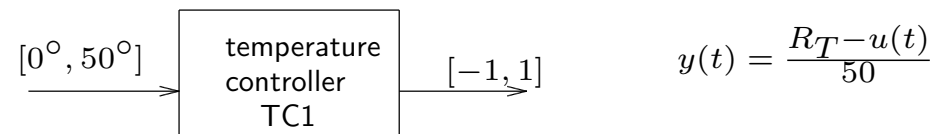
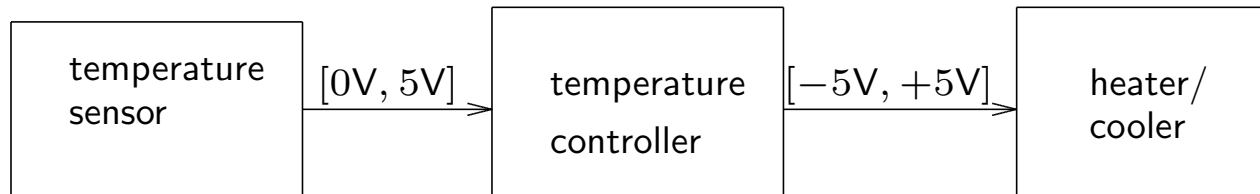
$$\vec{y}(t) = \vec{f}(\vec{u}(t)) = \begin{bmatrix} f_1(u_1(t), \dots, u_p(t)) \\ \vdots \\ f_m(u_1(t), \dots, u_p(t)) \end{bmatrix}$$



A Mathematical Model of a Real System



Example Temperature Controller



Static and Dynamic Systems

Definition: A **static system** is one where the output $y(t)$ is independent of past values of the input $u(t')$, $t' < t$ for all t

Definition: A **dynamic system** is one where the output $y(t)$ depends on the current input value $u(t)$ and on at least another input value $u(t')$ with $t' < t$, $y(t) = f(u(t), u(t'))$.



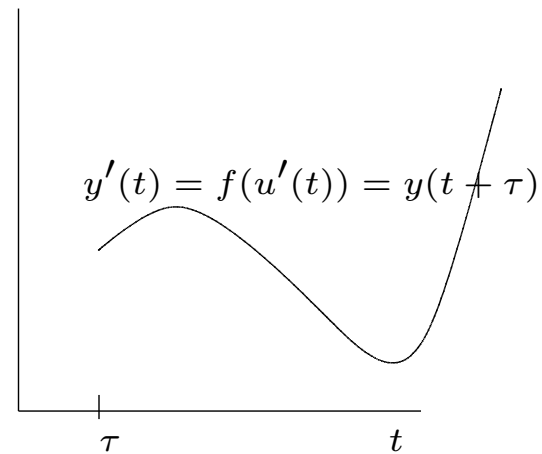
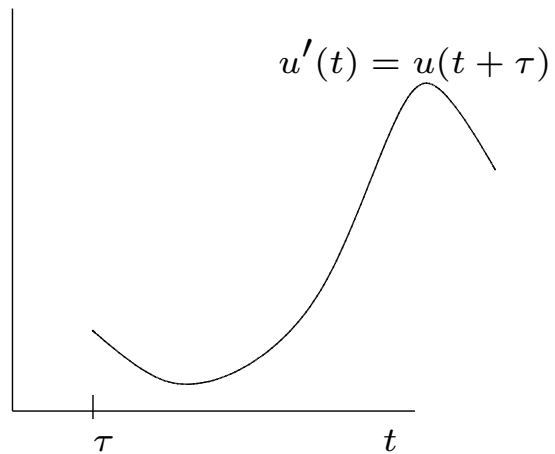
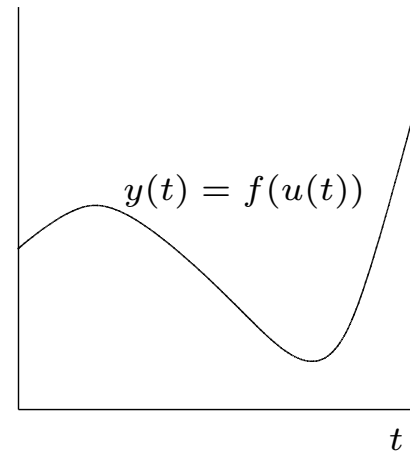
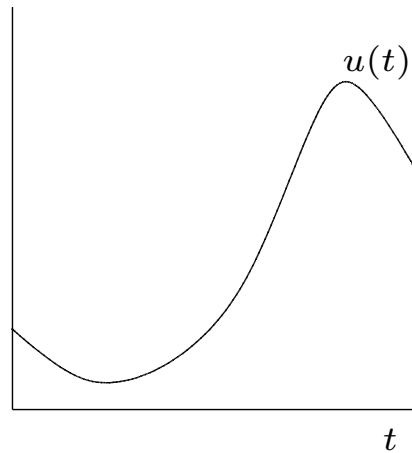
Time-Varying and Time-Invariant Systems

Definition: A model M with $\vec{y}(t) = \vec{f}(\vec{u}(t))$ is **time-invariant** if, supplied with input variables $\vec{u}'(t) = \vec{u}(t + \tau)$, it defines output variables $\vec{y}'(t) = \vec{f}(\vec{u}'(t)) = \vec{y}(t + \tau)$ for any τ .

Definition: In contrast, a **time-varying** model is one for which this property does not hold, i.e. where the output function explicitly depends on t : $\vec{y}(t) = \vec{f}(\vec{u}(t), t)$.



A Time-Invariant System



The Concept of State

Definition: The **state** of a system at time t_0 is the information required such that the output $\vec{y}(t)$ for all $t \geq t_0$ is uniquely determined by this information and the inputs $\vec{u}(t), t \geq t_0$.

The state of a system at time t_0 is called the **initial state** (\vec{x}_0).

The equations required to specify the state for all $t \geq t_0$, given the initial state and the input $\vec{u}(t), t \geq t_0$, are called **state equations**.

The **state space** X of the system is the set of all possible values of the state.



The State Space

Continuous-state models: The state space X is a continuum, e.g. $X = \mathbb{R}$ or $X = \mathbb{R}^n$

Discrete-state models: The state space is a discrete set, e.g. $X = \mathbb{N}$ or $X = \{0, 1\}$ or $X = \{\text{“blue”}, \text{“red”}, \text{“green”}\}$.

A State Space Model for Continuous Time, Continuous State Systems

State equations:

$$\dot{\vec{x}}(t) = \vec{g}(\vec{x}(t), \vec{u}(t), t)$$

Initial state:

$$\vec{x}(0) = \vec{x}_0$$

Output equations:

$$\vec{y}(t) = \vec{f}(\vec{x}(t), \vec{u}(t), t)$$



A State Space Model for Discrete Time Systems

State equations:

$$\vec{x}(t + 1) = \vec{g}(\vec{x}(t), \vec{u}(t), t)$$

Initial state:

$$\vec{x}(0) = \vec{x}_0$$

Output equations:

$$\vec{y}(t) = \vec{f}(\vec{x}(t), \vec{u}(t), t)$$



A State Space Model for Time-Invariant, Discrete Time Systems

State equations:

$$\vec{x}(t + 1) = \vec{g}(\vec{x}(t), \vec{u}(t))$$

Initial state:

$$\vec{x}(0) = \vec{x}_0$$

Output equations:

$$\vec{y}(t) = \vec{f}(\vec{x}(t), \vec{u}(t))$$



Linear and Non-linear Systems

Definition: A function $f : A \rightarrow A$ is linear if and only if

$$f(a_1x_1 + a_2x_2) = a_1f(x_1) + a_2f(x_2) \text{ for all } a_1, a_2, x_1, x_2 \in A.$$

A function $\vec{f} : A^n \rightarrow A^n$ is linear if and only if

$$\vec{f}(a_1\vec{x}_1 + a_2\vec{x}_2) = a_1\vec{f}(\vec{x}_1) + a_2\vec{f}(\vec{x}_2) \text{ for all } a_1, a_2 \in A, x_1, x_2 \in A^n,$$

where A^n is the set of vectors of length n with elements of A .

A system with state function \vec{g} and output function \vec{f} is linear if and only if both functions \vec{f} and \vec{g} are linear.

Matrix Equations for Discrete, Linear Systems

$$\begin{aligned}\vec{x}(t+1) &= \mathbf{A}(t) \vec{x}(t) + \mathbf{B}(t) \vec{u}(t) \\ \vec{y}(t) &= \mathbf{C}(t) \vec{x}(t) + \mathbf{D}(t) \vec{u}(t)\end{aligned}$$

where

- $\mathbf{A}(t)$... $n \times n$ matrix,
- $\mathbf{B}(t)$... $n \times p$ matrix,
- $\mathbf{C}(t)$... $m \times n$ matrix,
- $\mathbf{D}(t)$... $m \times p$ matrix,
- n ... number of state variables,
- m ... number of output variables,
- p ... number of input variables.



Matrix Equations for Discrete, Linear, Time-invariant Systems

$$\begin{aligned}\vec{x}(t+1) &= \mathbf{A} \vec{x}(t) + \mathbf{B} \vec{u}(t) \\ \vec{y}(t) &= \mathbf{C} \vec{x}(t) + \mathbf{D} \vec{u}(t)\end{aligned}$$

with constant matrices \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{D} .



Deterministic, Stochastic and Nondeterministic Systems

Definition: A system model is **deterministic** if the output function \vec{f} and the state function \vec{g} are functions in the sense that they evaluate a given argument always and unambiguously to the same result.

A system model is **stochastic** if at least one of their output variables is a random variable.

A system model is **nondeterministic** if a given input may result in different outputs.



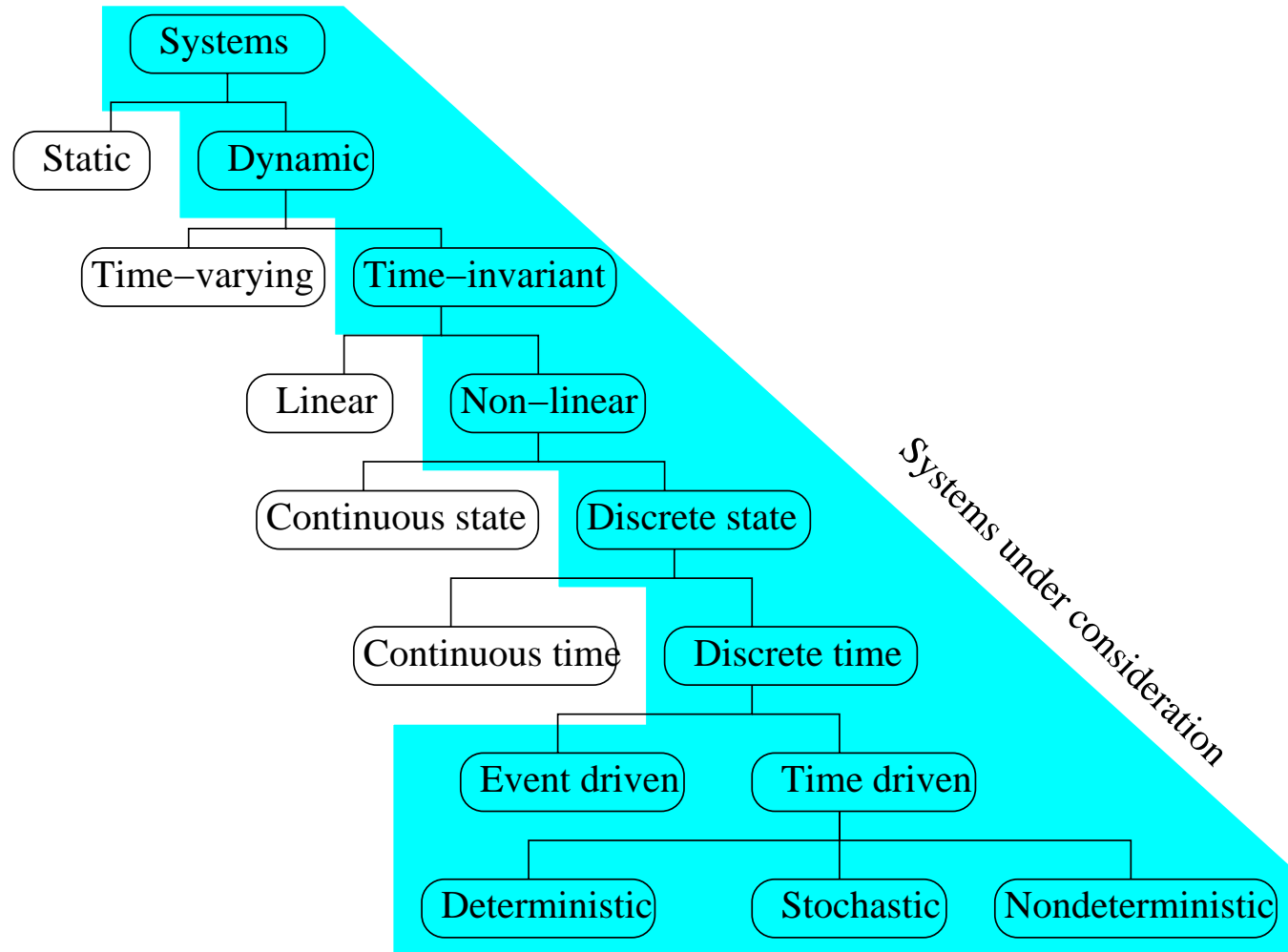
Events

- Events are associated with a time instance and have no duration.
- Examples:
 - ★ Arrival of a message;
 - ★ Change of a signal value;
 - ★ Change of a state;
 - ★ A counter exceeding a given threshold value;
 - ★ An elapsed time period;
 - ★ etc.

Time-driven and Event-driven

- In **time-driven systems** the advance of time causes the system to become active.
- In **event-driven systems** the occurrence of an event causes the system to become active.

System Classification Summary

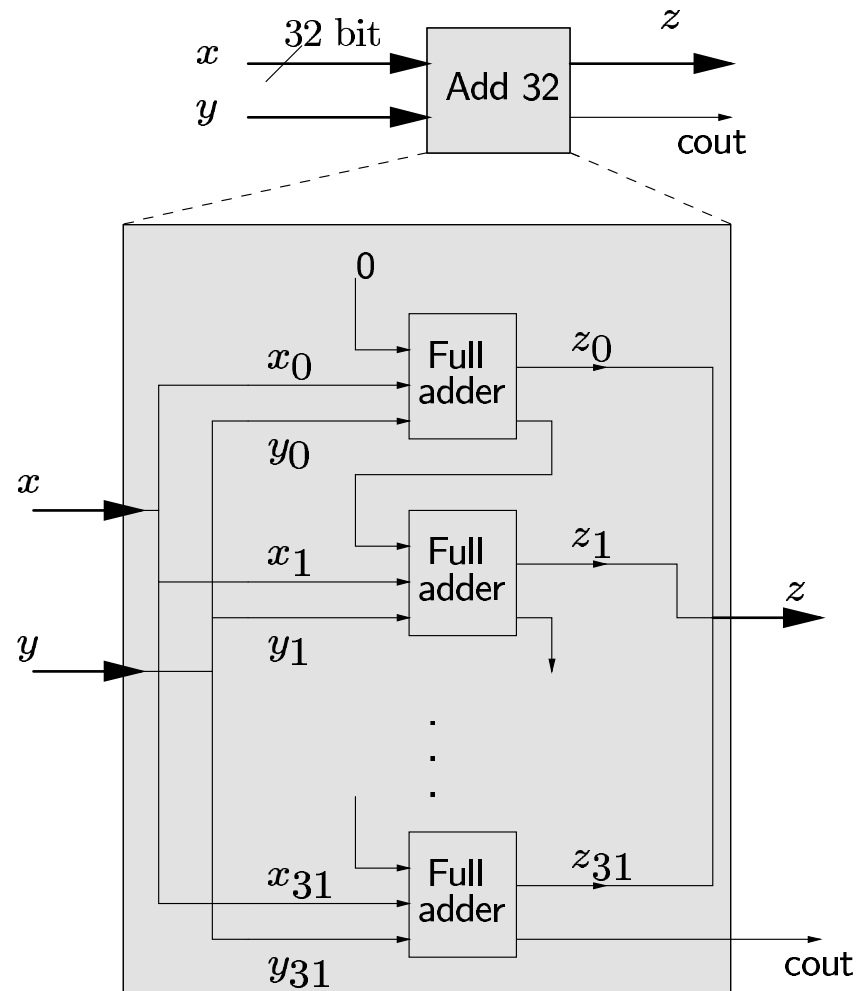


The Rugby Meta-Model

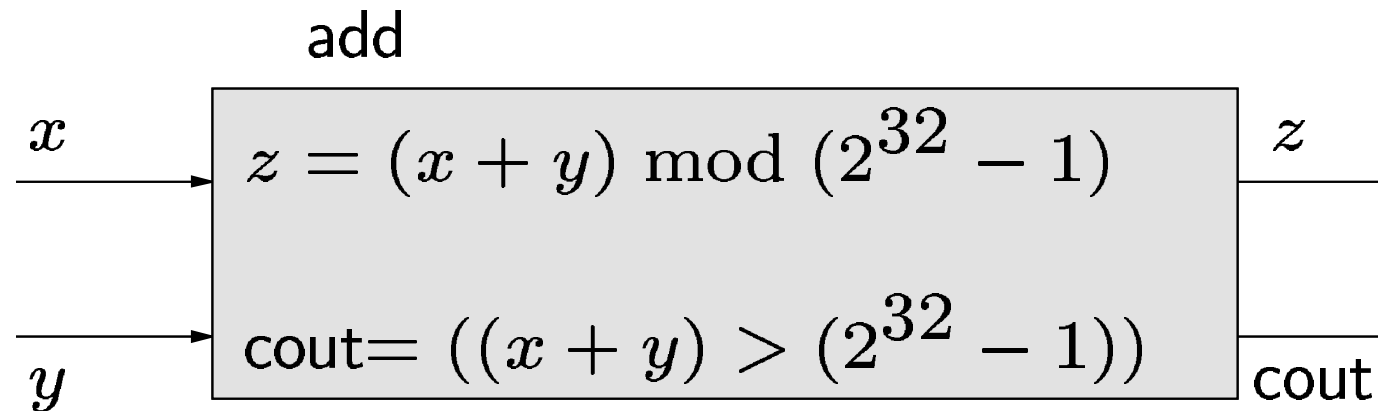
Abstraction in four domains

- Computation
- Communication
- Time
- Data

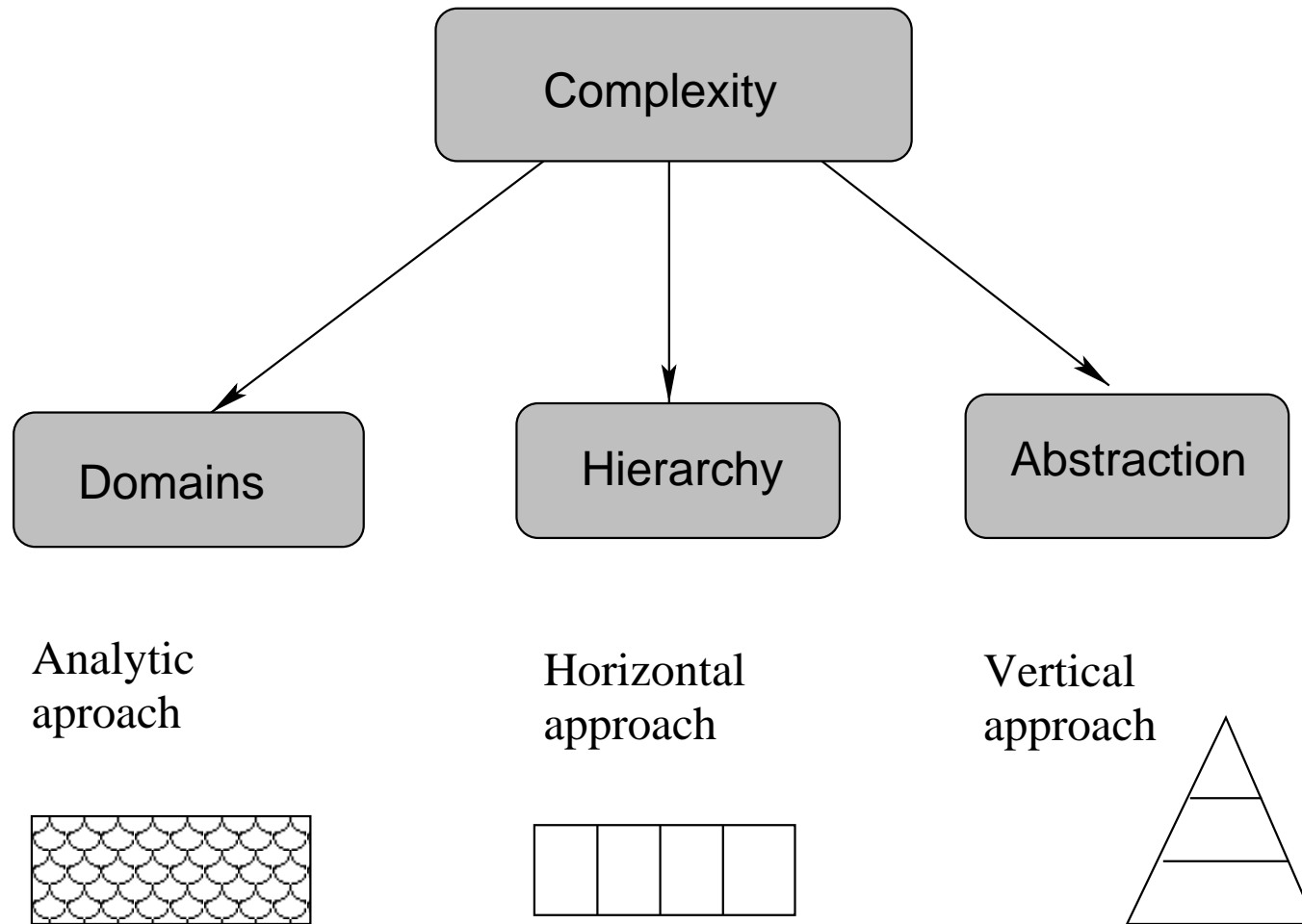
Example of a Hierarchy



Example of an Abstraction



Ways to Handle Complexity



Hierarchy, Abstraction, Domain

Hierarchy: A hierarchy is a, possibly recursive, partitioning of a design model such, that the details of each part is hidden into a lower hierarchical level.

⇓ Moving down the hierarchy **displays** information.

⇑ Moving up the hierarchy **hides** information.

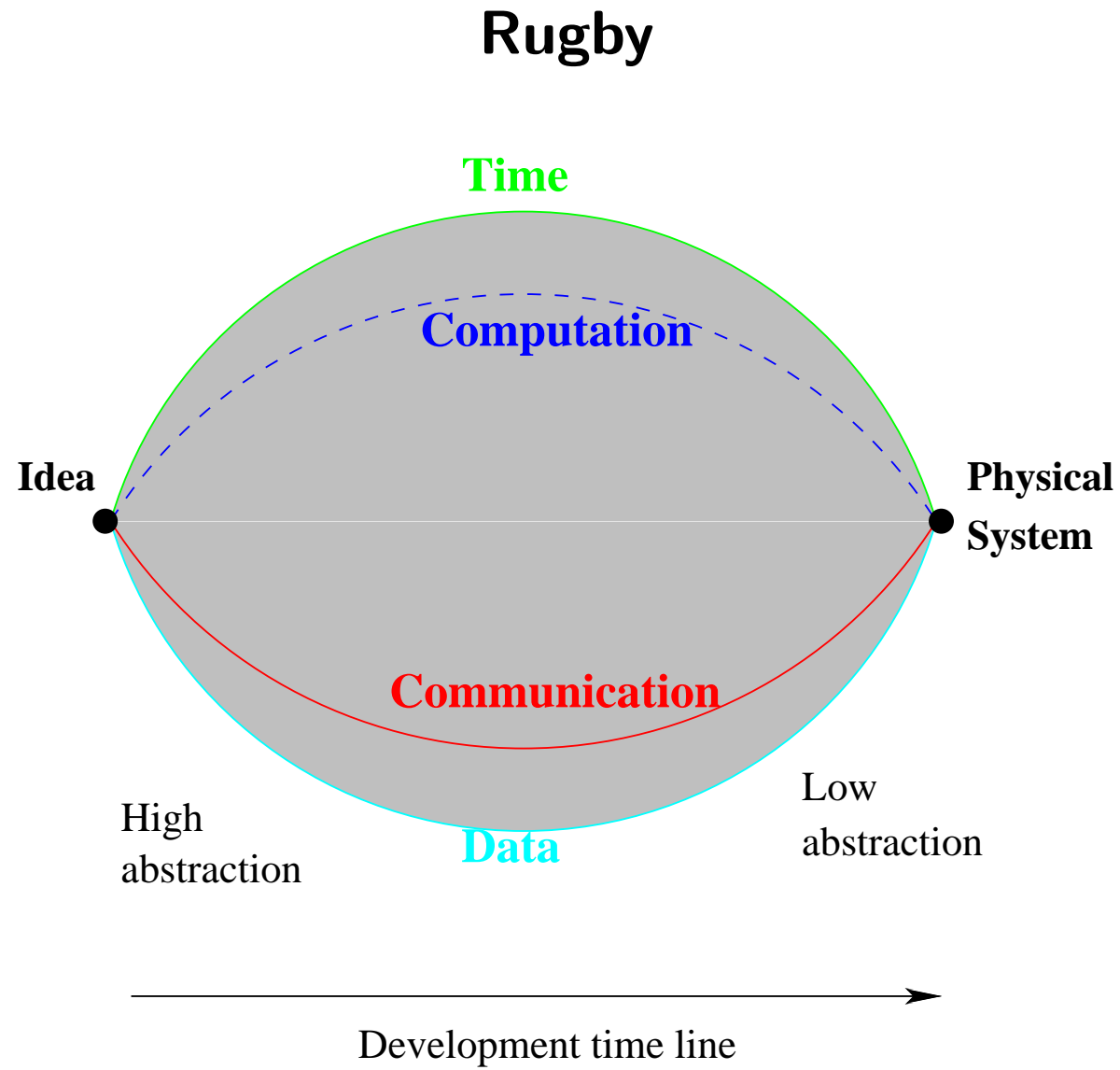
Abstraction: An abstraction level defines the modeling concepts and their semantics for representing a system. The type of information available at different levels is different. A higher level ignores some irrelevant information of a lower level or encodes it using different concepts.

⇓ Moving down an abstraction level **adds** information.

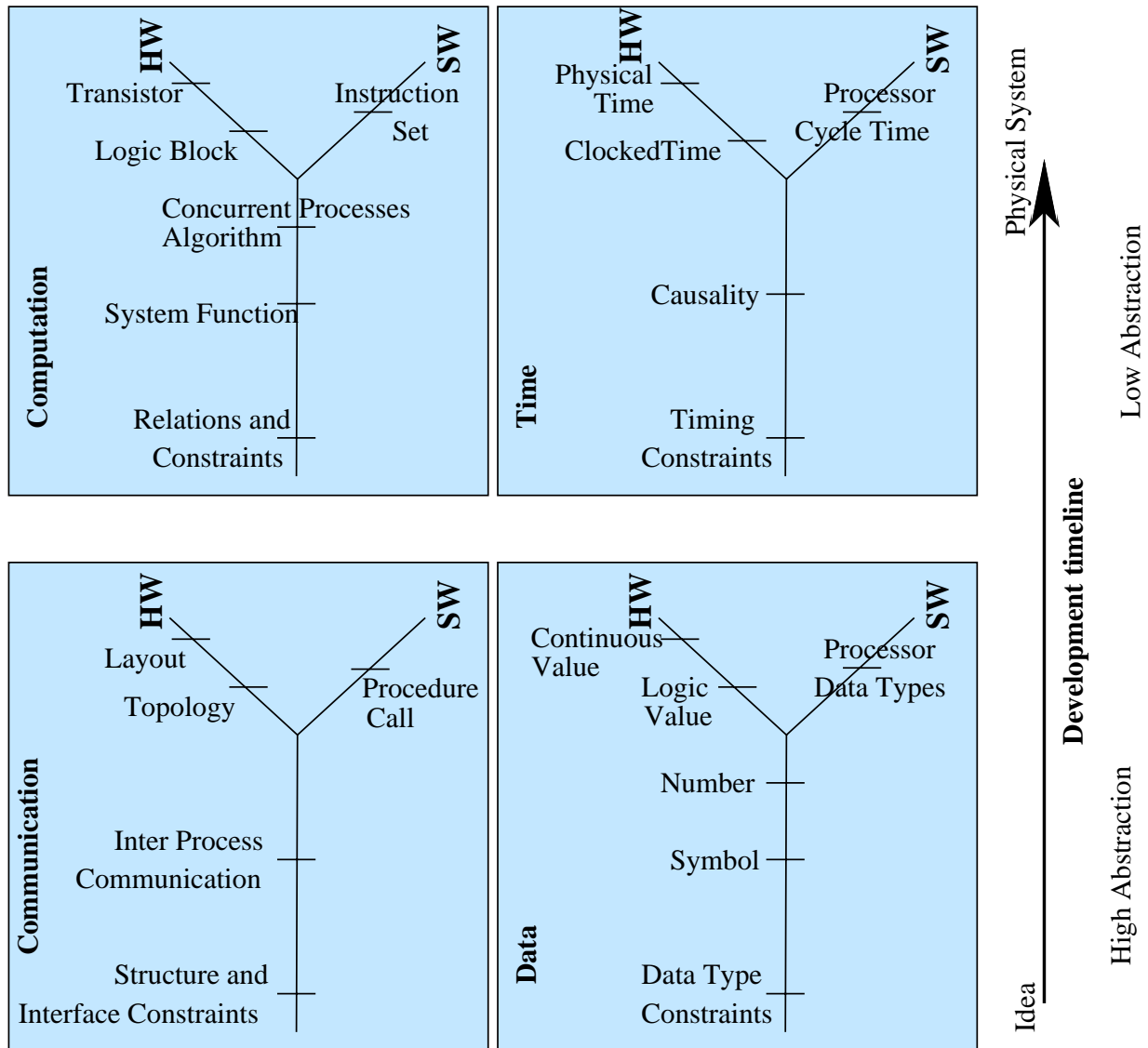
⇑ Moving up an abstraction level **removes** information.

Domain: A domain is an aspect of a model which can logically be analyzed independently from other aspects.

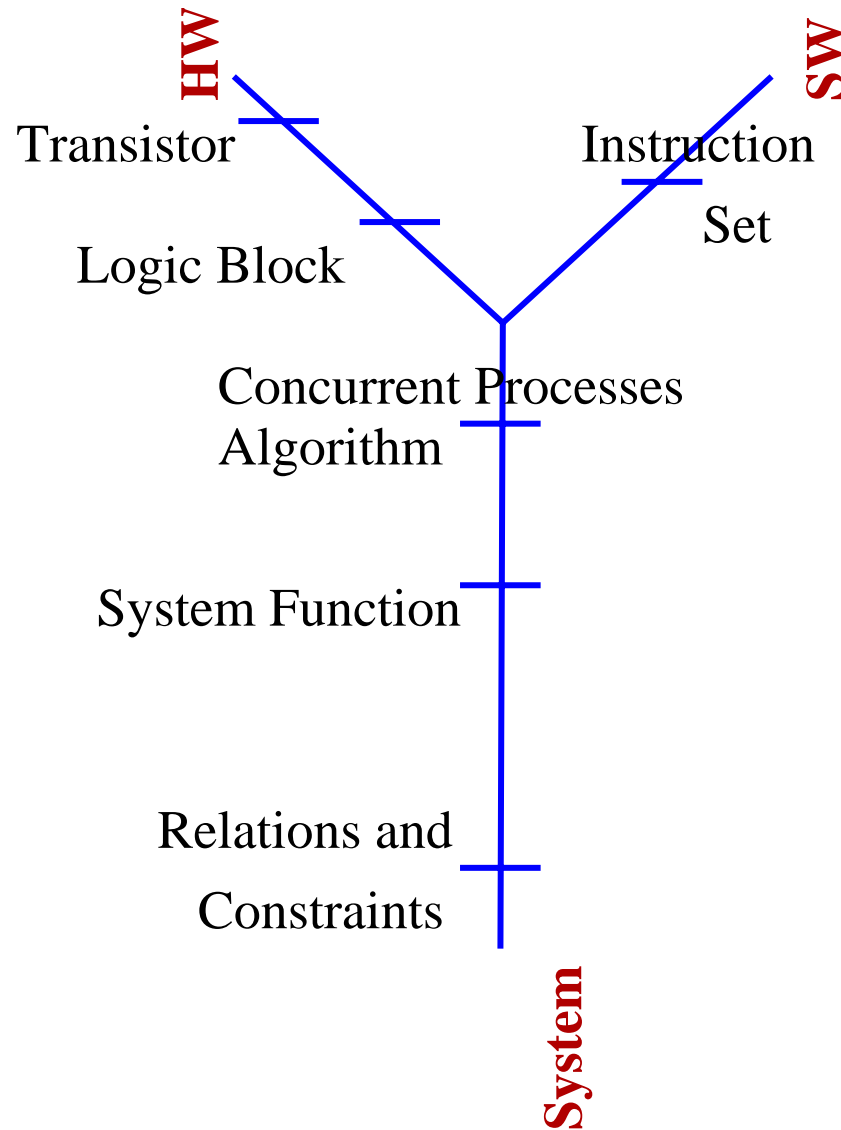




The Domains in Rugby



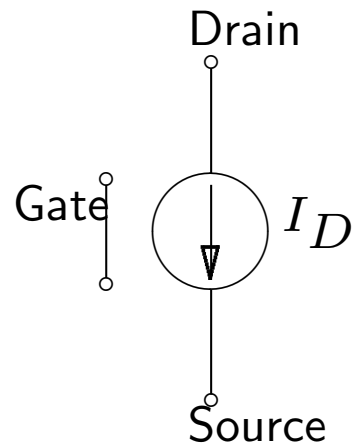
The Computation Domain



A MOS Transistor Model

$V_{DS} > V_{GS} - V_T$ (conducting state):

$$I_D = \frac{k'_n W W}{2L} (V_{GS} - V_T)^2 (1 + \lambda V_{DS})$$



$V_{DS} < V_{GS} - V_T$ (sub-threshold state):

$$I_D = \frac{k'_n W W}{L} \left((V_{GS} - V_T) V_{DS} - \frac{V_{DS}^2}{2} \right)$$

where

$$V_T = V_{T0} + \gamma (\sqrt{|-2\phi_F + V_{SB}|} - \sqrt{|-2\phi_F|})$$

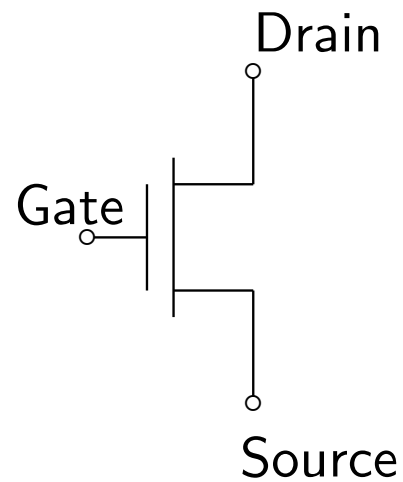
V_{DS} ... drain-source voltage

V_{GS} ... gate-source voltage

V_T ... threshold voltage

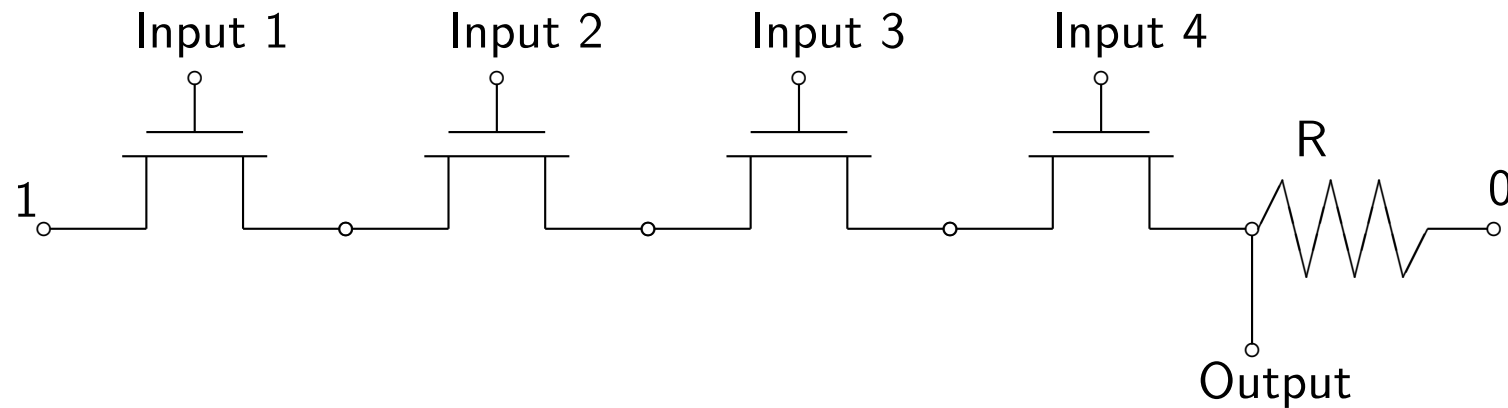
I_D ... drain-source current

A Transistor as Switch



Gate	Drain	Source
0	0	undefined
0	1	undefined
1	0	0
1	1	1

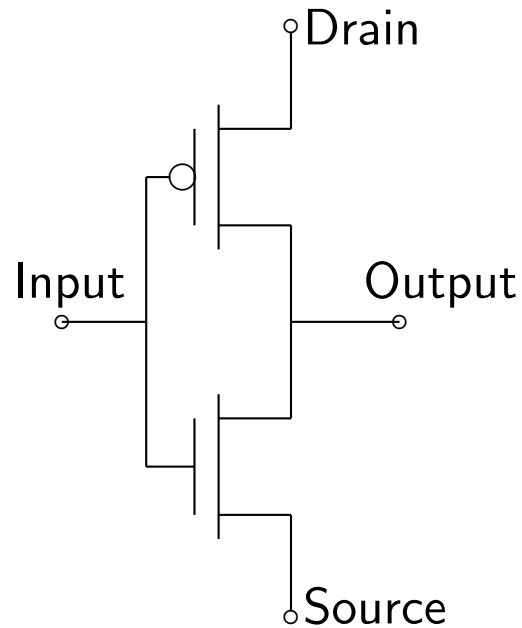
An AND Gate as Transistor Network



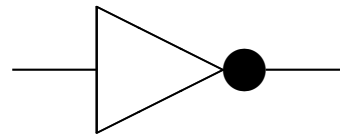
Two problems with arbitrary transistor networks:

- Output is not defined when input is 0.
- Voltage drop between drain and source is relevant but not visible.

An Inverter as Transistor Network



(a)



(b)

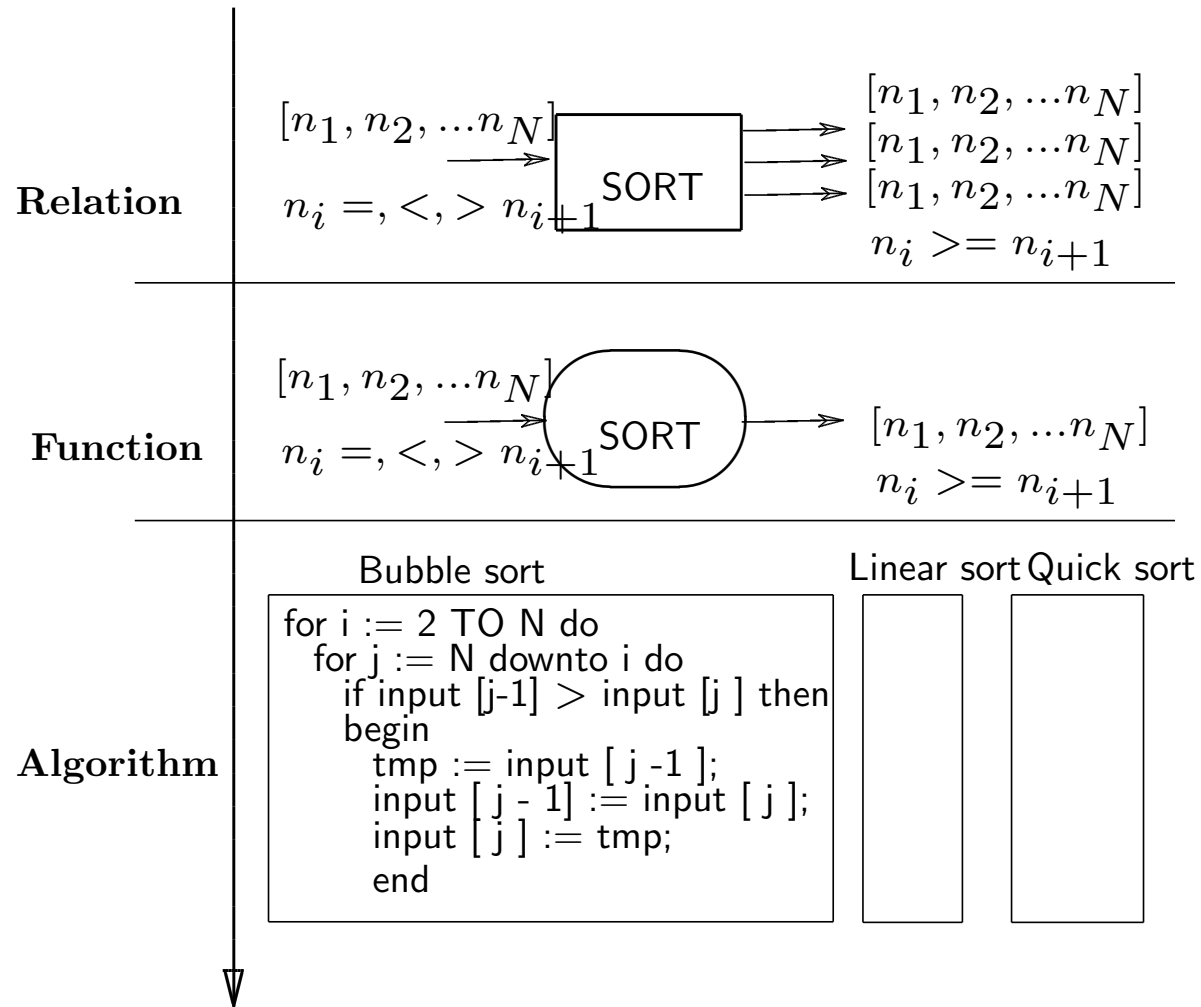
Input	Output
1	0
0	1

(c)

Gate Based Abstraction Level

1. The primitive elements are defined by simple models, i.e. small truth tables in this case.
2. The primitive elements can be implemented in a wide range of technologies.
3. The model holds even for arbitrarily large networks of primitive elements.

Algorithms, Functions, Relations



Sorting Defined as Relation

sort (IntArray A) \rightarrow (IntArray B)

Precondition: true

Post condition: $\forall a \in A : a \in B$

$\wedge \forall b \in B : b \in A$

$\wedge \forall i, j \in \text{Integer}, b_i, b_j \in B : i < j \Rightarrow b_i \leq b_j$

Sorting Defined as Function

$$\begin{aligned} \text{sort} &:: [\text{Integer}] \rightarrow [\text{Integer}] \\ \text{sort} \quad [] &= [] \\ \text{sort} \quad (x : xs) &= (\text{sort} (\text{selectLT } x \ xs)) \ ++ \ [x] \\ &\quad \ ++ \ (\text{sort} (\text{selectGE } x \ xs)) \end{aligned}$$

Sorting Defined as Algorithms

```
Array sort_s (iarray) {  
  Array oarray = EArray; // initialized to the empty array  
  int x = iarray[0];  
  oarray = sort_s(selectLT (x,iarray));  
  append(oarray,x);  
  append(oarray,sort_(selectGE (x,iarray)));  
  return (oarray); }
```

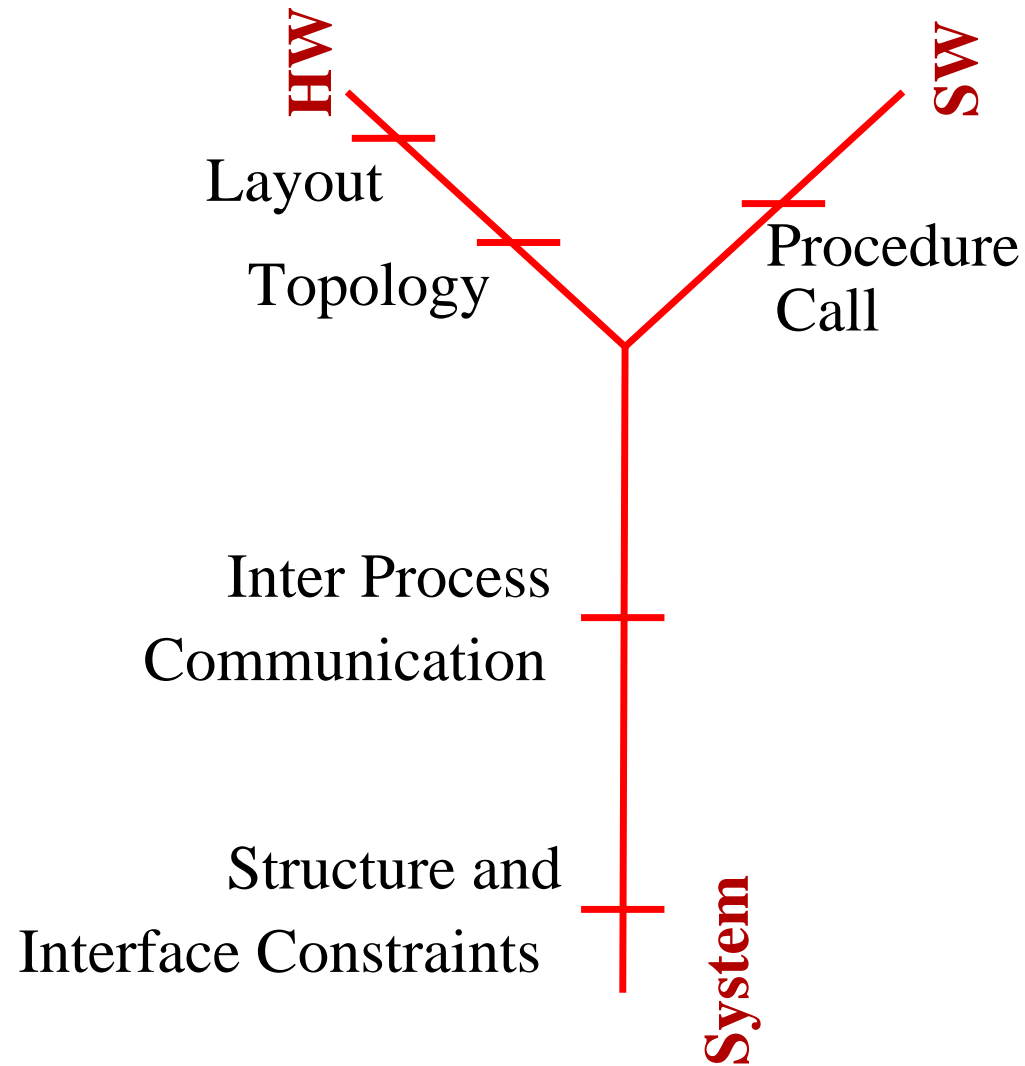
a sequential algorithm

```
Array sort_p (iarray) {  
  Array oarray,oarray2 = EArray;  
  int x = iarray[0];  
  par{ oarray = sort_p(selectLT (x,iarray))  
      | oarray2 = sort_p(selectGE (x,iarray))  
      }  
  append(oarray,x); append(oarray,oarray2);  
  return (oarray); }
```

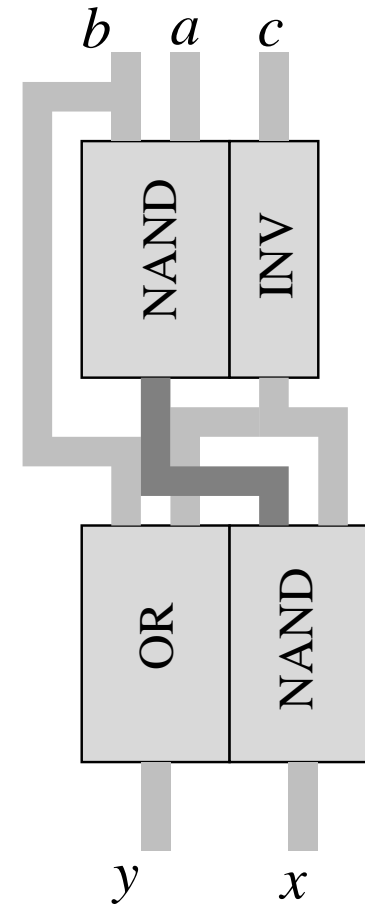
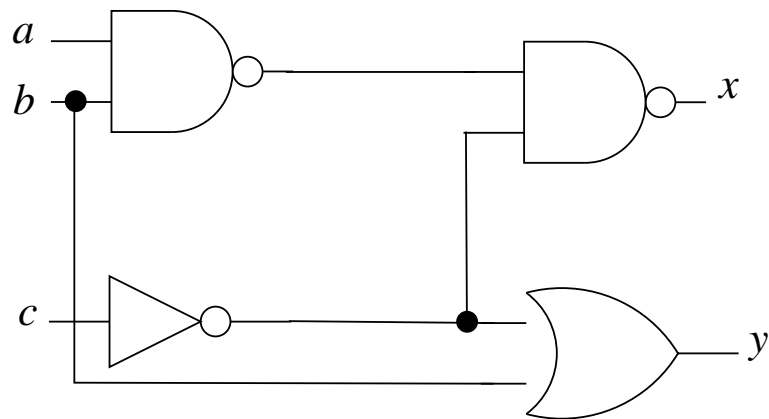
a parallel algorithm



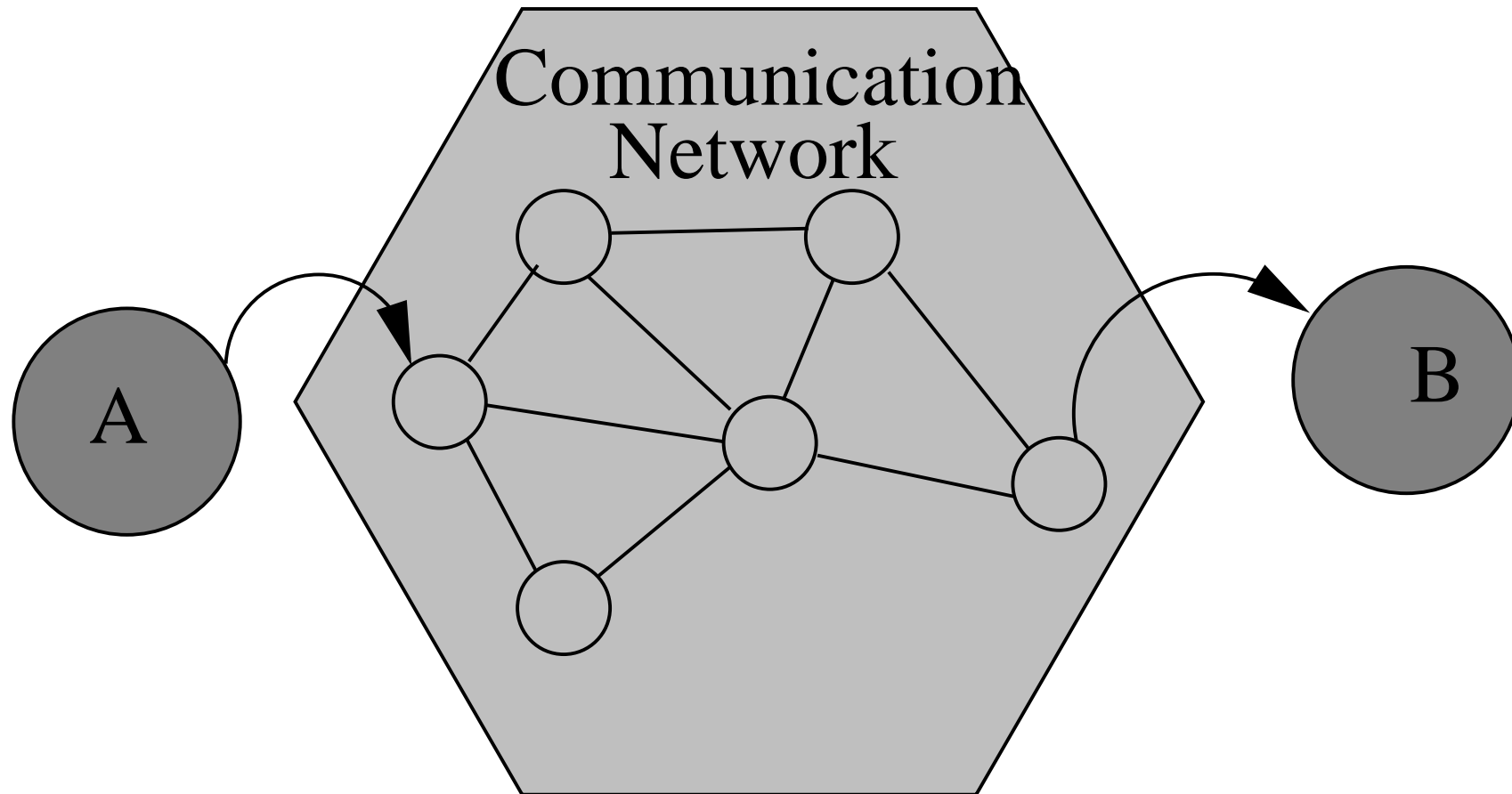
The Communication Domain



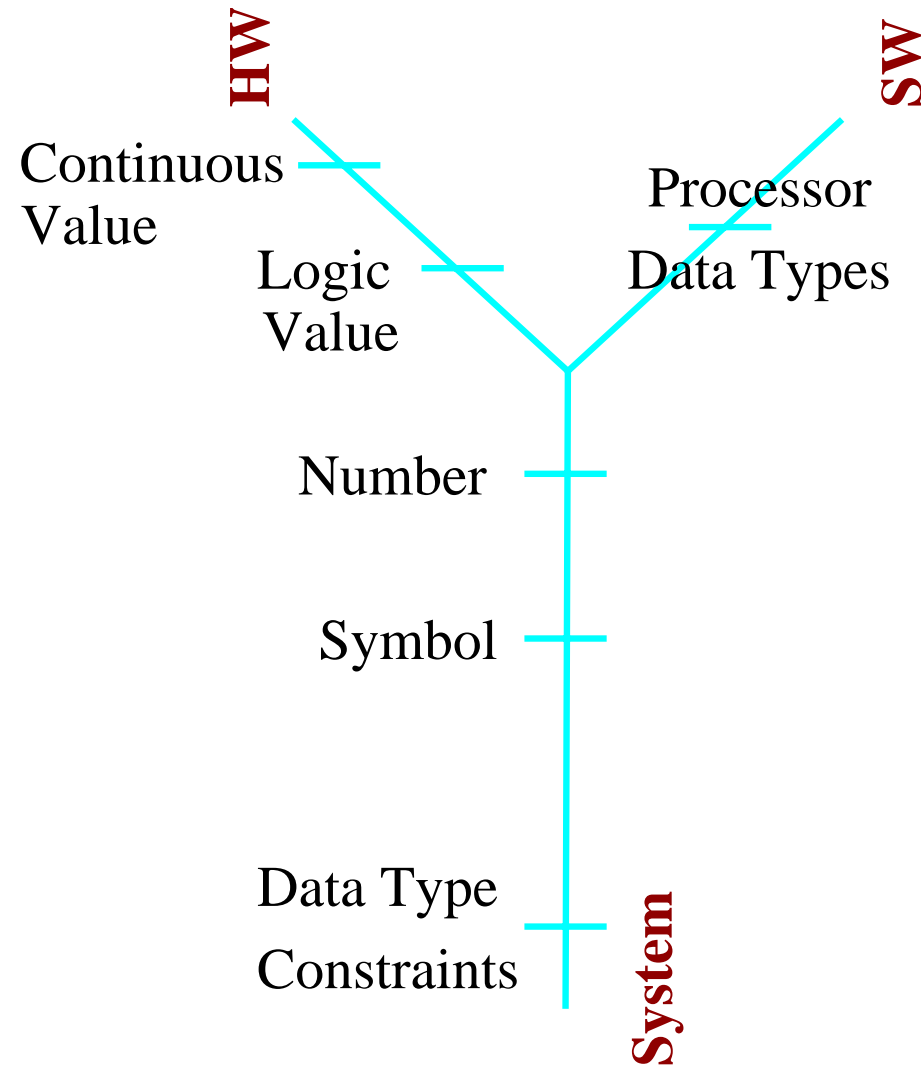
Communication at the Gate and Layout Level



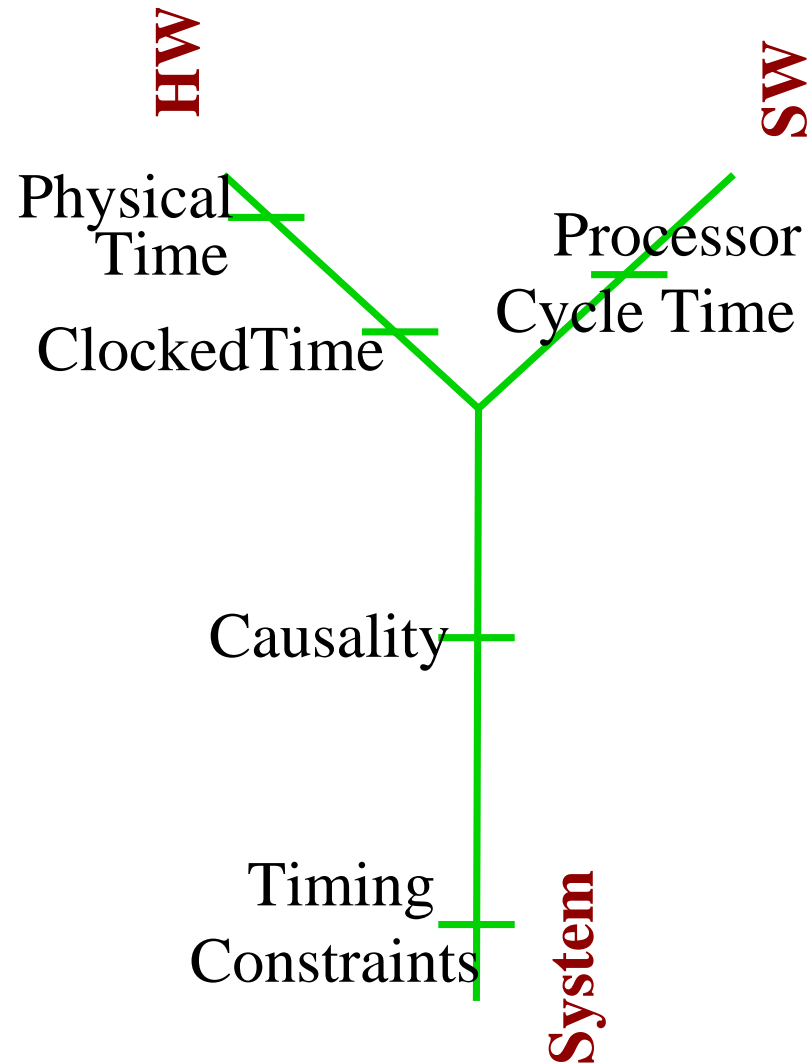
Communication between Processes



The Data Domain



The Time Domain



Notation for Abstraction Levels: Computation

Relations and Constraints	RelationConstraints	RC
System Functions	SystemFunctions	SF
Concurrent Processes, Algorithms	Algorithm	Alg
Logic Block	LogicBlock	LB
Transistor	Transistor	Tran
Instruction Set	InstructionSet	Inst

Notation for Abstraction Levels: Communication

Structural and Interface Constraints	InterfaceConstraints	IC
Inter Process Communication	InterProcessComm	IPC
Topology	Topology	Top
Layout	Layout	Lay
Procedure Call	ProcCall	PC

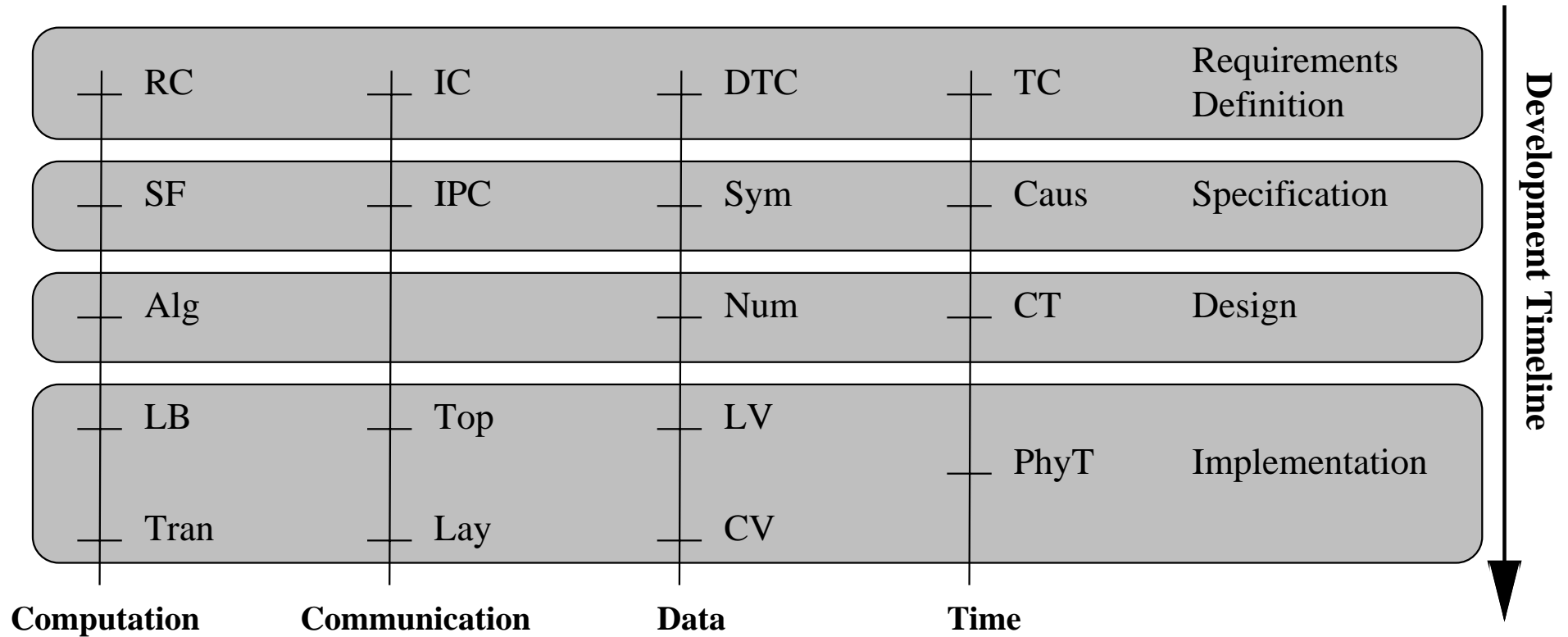
Notation for Abstraction Levels: Data

Data Type Constraints	DataTypeConstraints	DTC
Symbol	Symbol	Sym
Number	Number	Num
Logic Value	LogicValue	LV
Continuous Value	ContValue	CV
Processor Data Types	ProcessorDataTypes	PDT

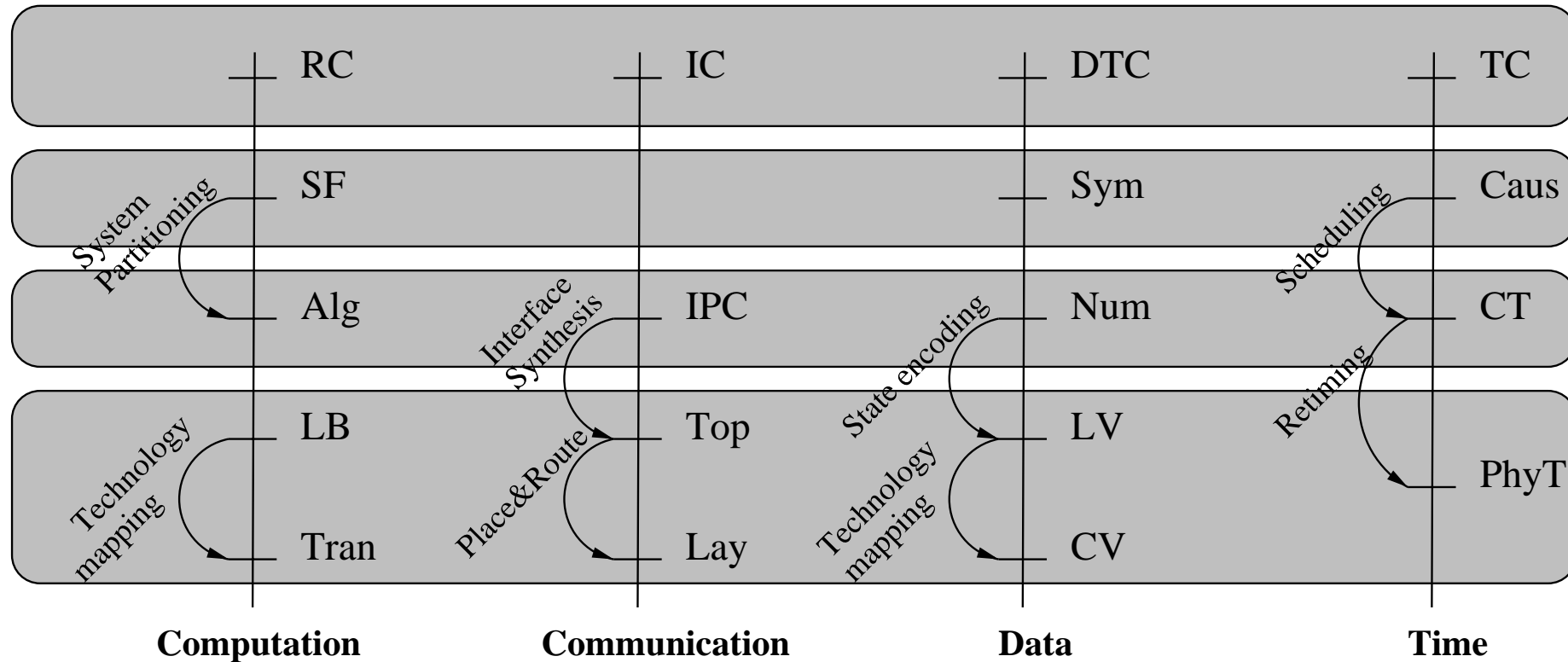
Notation for Abstraction Levels: Time

Timing Constraints	TimingConstraints	TC
Causality	Causality	Caus
Clocked Time	ClockedTime	CT
Physical Time	PhysicalTime	PhyT
Processor Cycle Time	ProcessorCycleTime	PCT

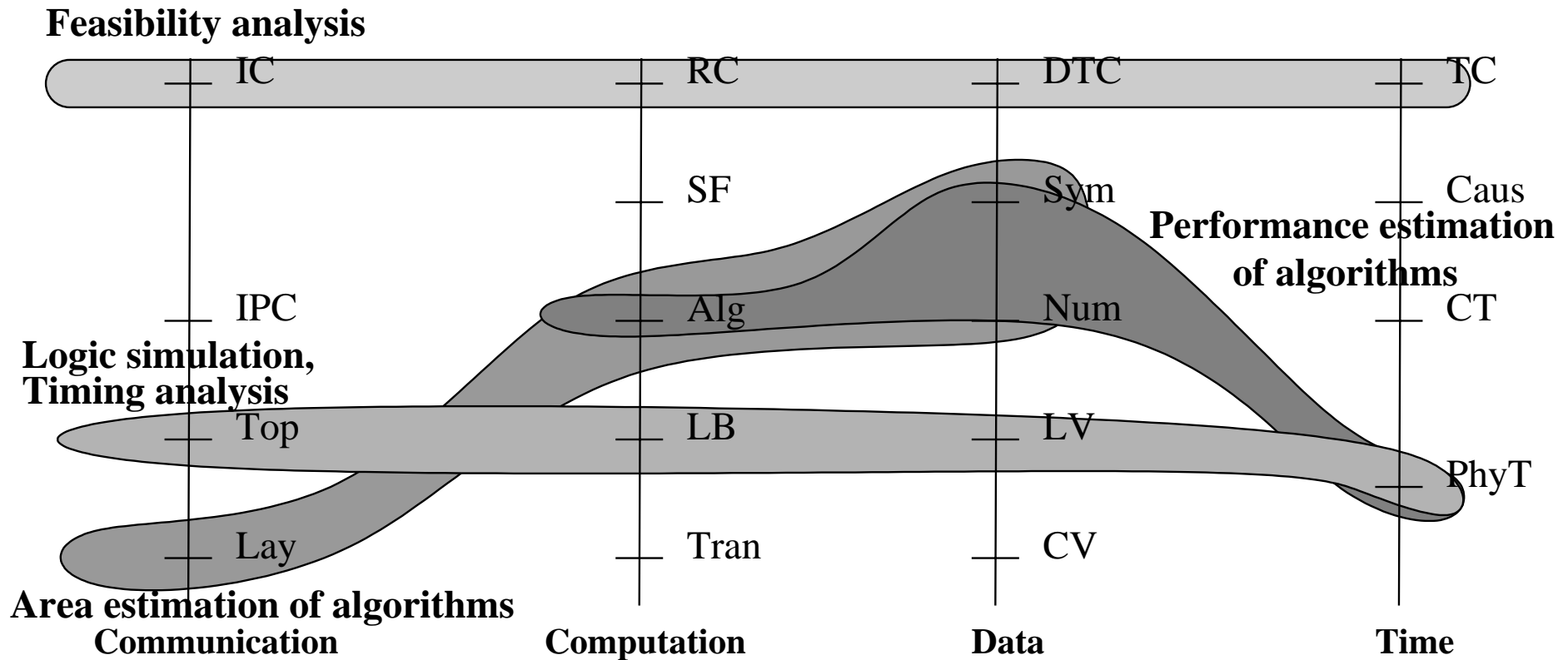
Abstraction Levels in Design Phases



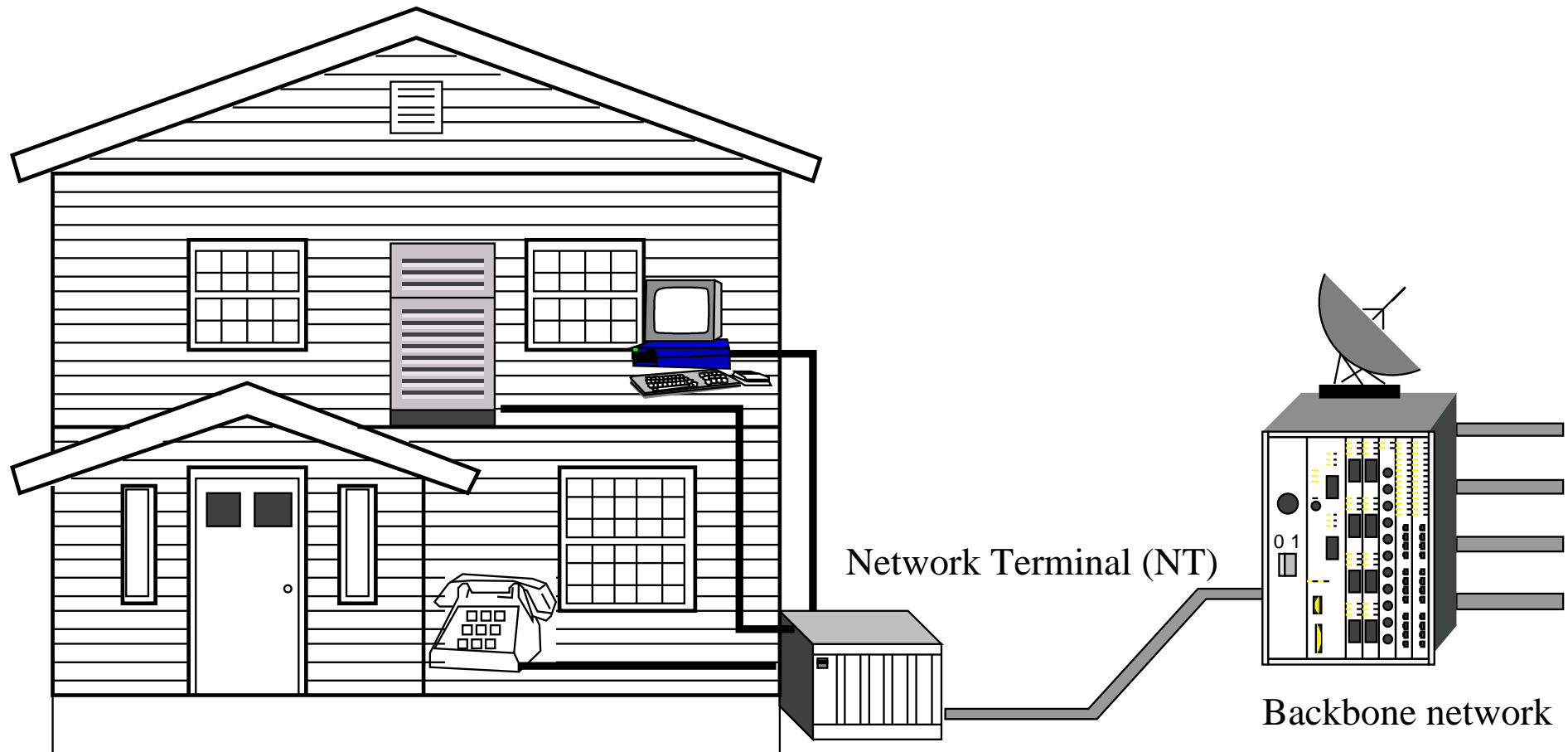
Design Activities in Terms of Abstraction Levels



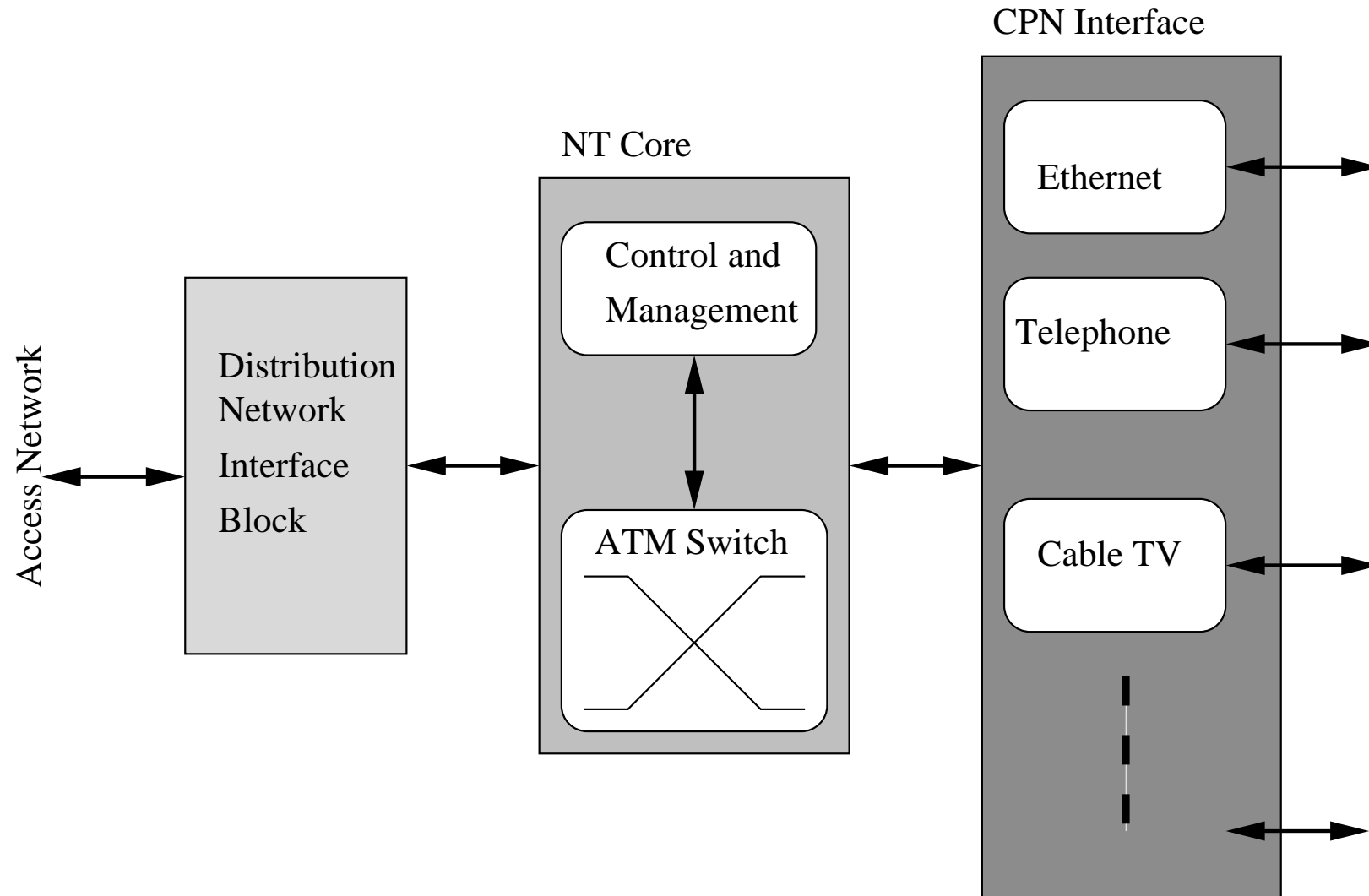
Analysis Activities in Terms of Abstraction Levels



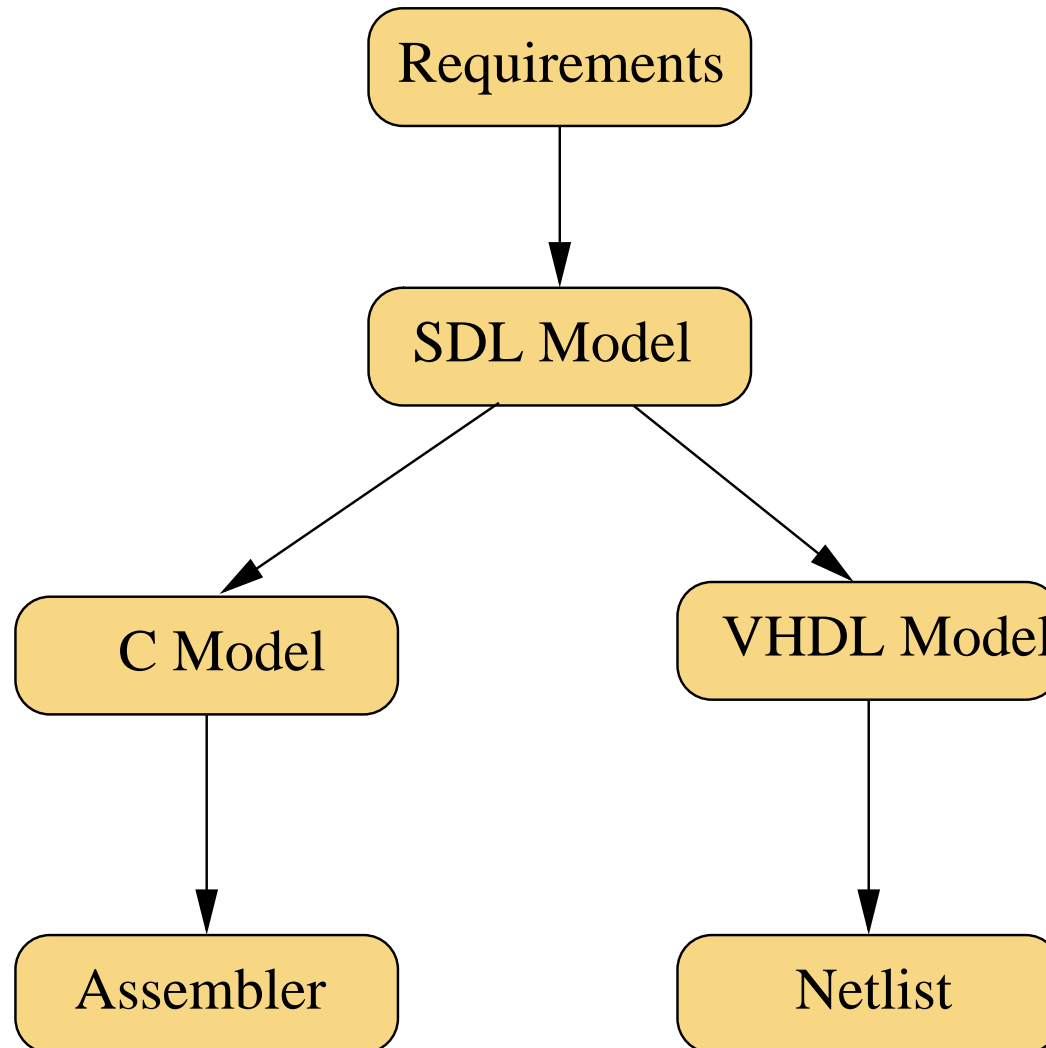
The Network Terminal Case Study



The Network Terminal Case Study



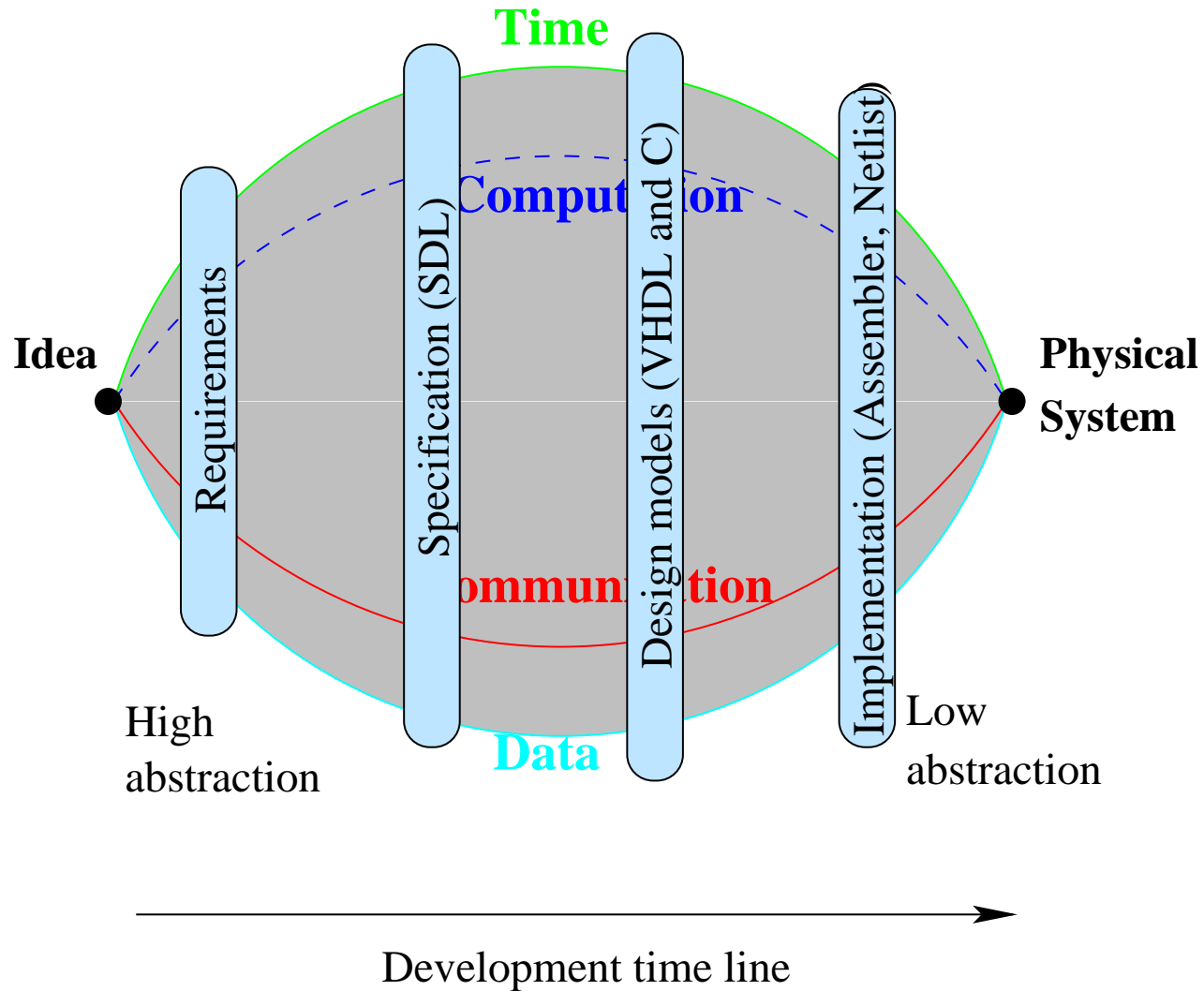
The NT Design Flow



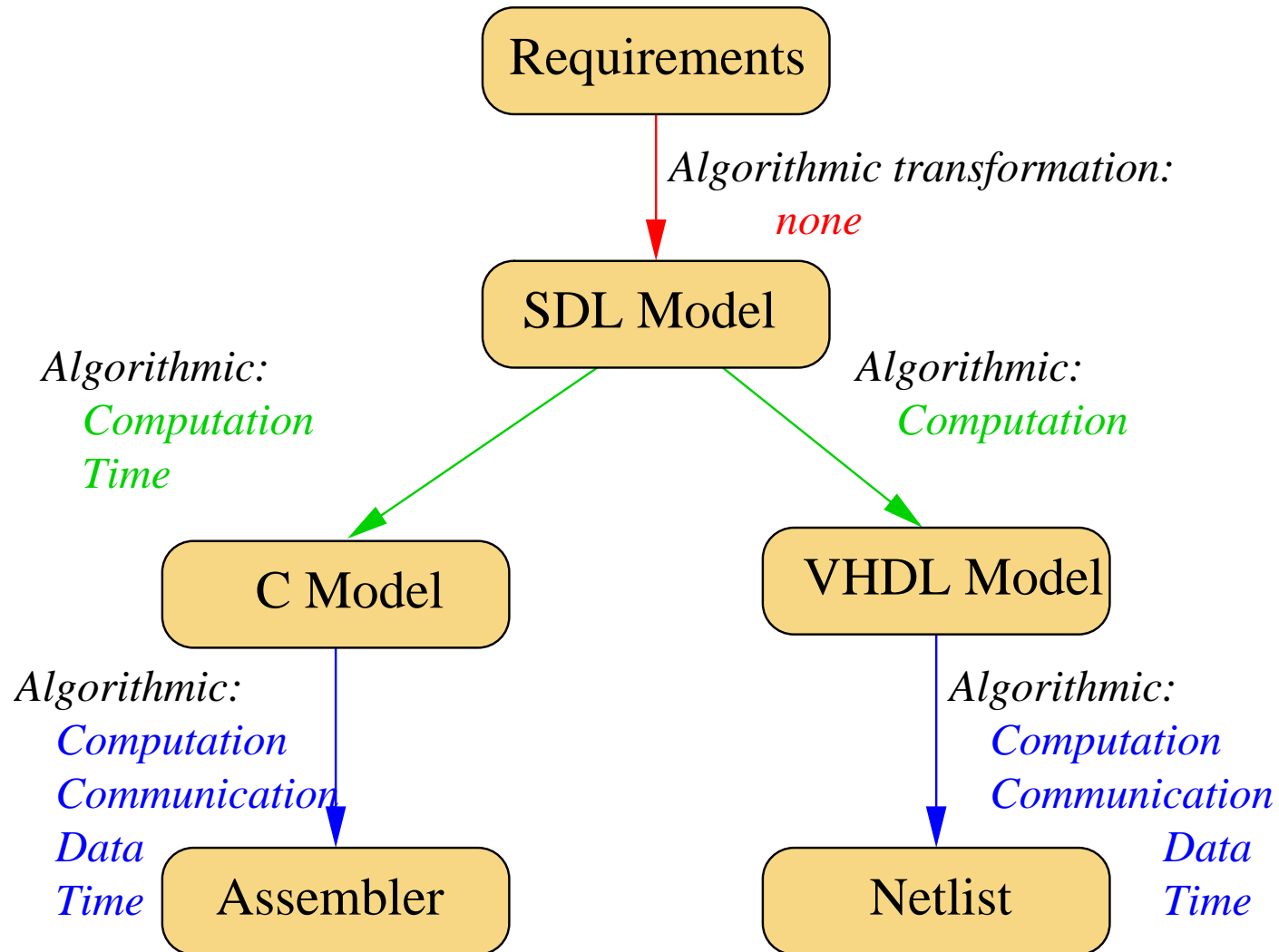
Models in the NT Design

Models	Abstraction Levels				
	Computation	Communication	Data	Time	
Requirements	RC	IC	DTC	TC	
System (SDL)	Alg	IPC	Sym	Caus	
HW	VHDL	Alg	Top	Sym,LV	CT
	Netlist	LB	Top	LV	PhyT
SW	C	ALg	PC	Sym	CT
	Assembler	Inst	PC	PDT	PCT

The Network Terminal Case Study



Transformations in the NT Design



Summary of the Rugby Model

- Rugby defines four domains
 - ★ Data
 - ★ Time
 - ★ Communication
 - ★ Computation
- Rugby defines several abstraction levels from constraints to discrete logic.
- Models, design techniques and tools can be analysed with Rugby.