System-Level Design (and Modeling for Embedded Systems)

Lecture 2 – Methodologies, Models, Languages

Kim Grüttner kim.gruettner@dlr.de
Henning Schlender henning.schlender@dlr.de
Jörg Walter joerg.walter@offis.de

System Evolution and Operation
German Aerospace Center (DLR)
&
Distributed Computation and Communication
OFFIS

© 2009 Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

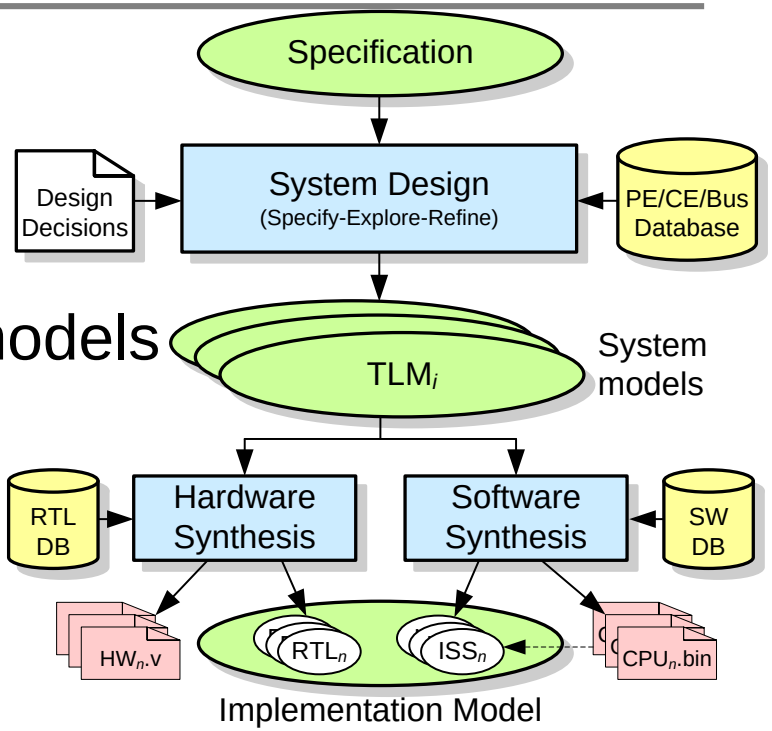
Lecture 2: Outline

- **Introduction**
  - Methodologies, models, languages
- **Methodologies**
  - Bottom-up, top-down, meet-in-the-middle, platform-based
- **Modeling**
  - System design flow
- **System design languages**
  - Goals, requirements
  - Communication and computation

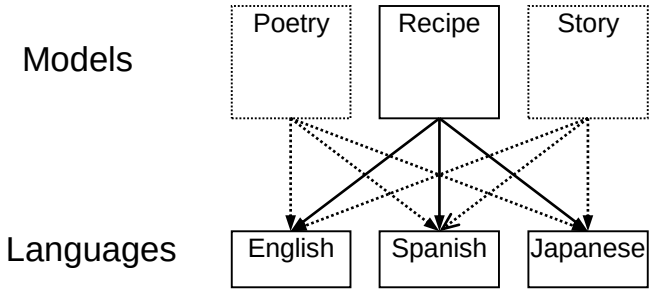
© 2009 A. Gerstlauer

Introduction

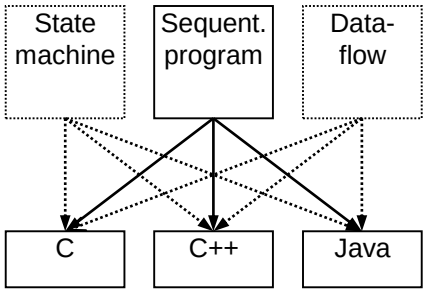
- **Design methodology / design flow**
  - Sequence of design models
  - Flow of transformations between models
  - Component databases and tools
- **Models**
  - Core of design flow definition
    - Apply analysis, synthesis and verification techniques
  - Abstract view of a design
    - Observe, predict and reason about properties
- **Languages**
  - Representation of models in machine-readable form



Models vs. Languages



Recipes vs. English



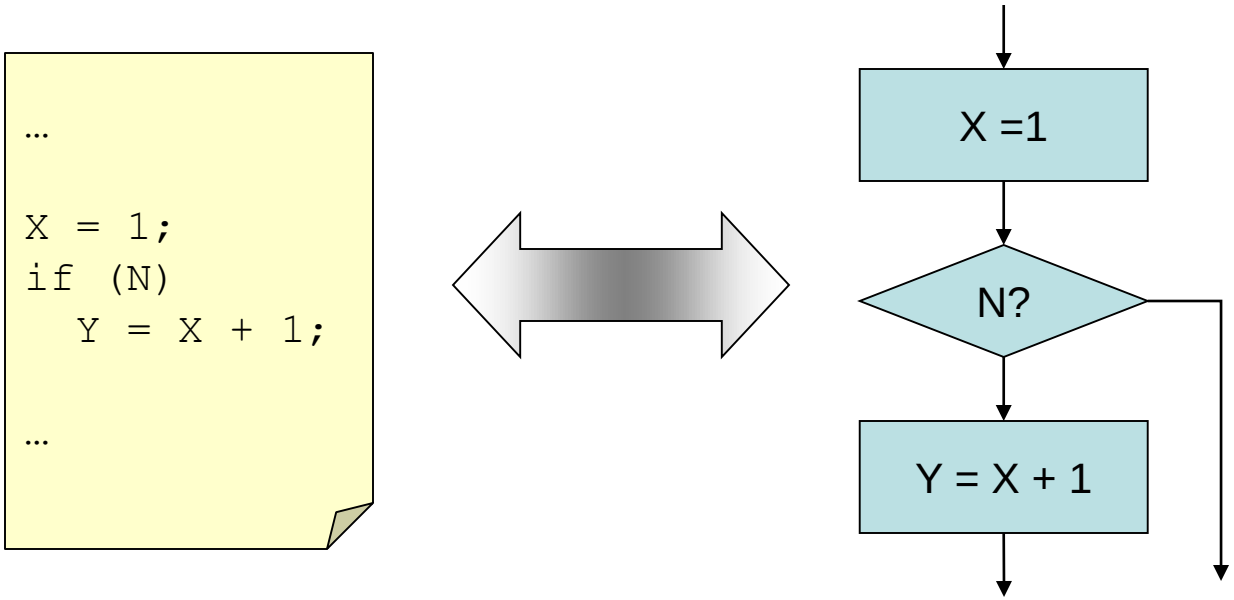
Sequential programs vs. C

- **Computation models describe system behavior**
  - Conceptual notion, e.g., recipe, sequential program
- **Languages capture models**
  - Concrete form, e.g., English, C
- **Variety of languages can capture one model**
  - E.g., sequential program model → C,C++, Java
- **One language can capture variety of models**
  - E.g., C++ → sequential program model, object-oriented model, state machine model
- **Certain languages better at capturing certain models**

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

Text vs. Graphics

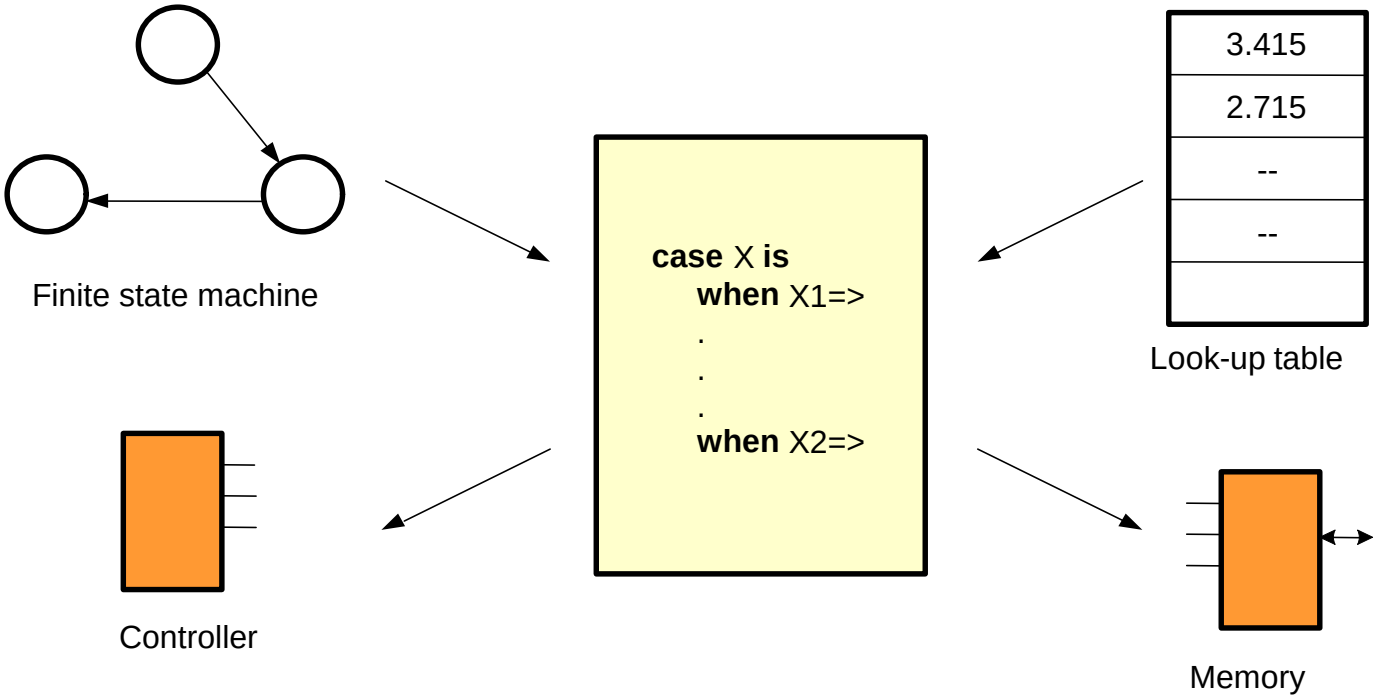
- **Models versus languages not to be confused with text versus graphics**
  - Text and graphics are just two types of languages
    - Text: letters, numbers
    - Graphics: circles, arrows (plus some letters, numbers)



Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

Simulation vs. Synthesis

- **Ambiguous semantics of languages**



- **Simulatable but not synthesizable or verifiable**
  - Impossible to automatically discern implicit meaning
  - Need explicit set of constructs

Source: D. Gajski, UC Irvine

System Design Needs

- **Design models and languages**
  - System specification
    - Functionality, concurrency, computation & communication
  - System architecture
    - Network of PEs, CEs and busses

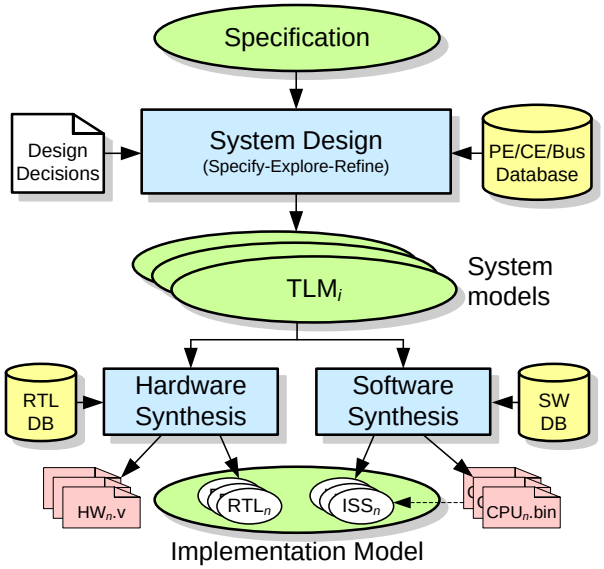
➢ **Well-defined, rigorous system-level semantics**

➢ Unambiguous, explicit abstractions, models
  – Objects and composition rules

➢ Systematic flow from specification to implementation
  – Transformations and refinements

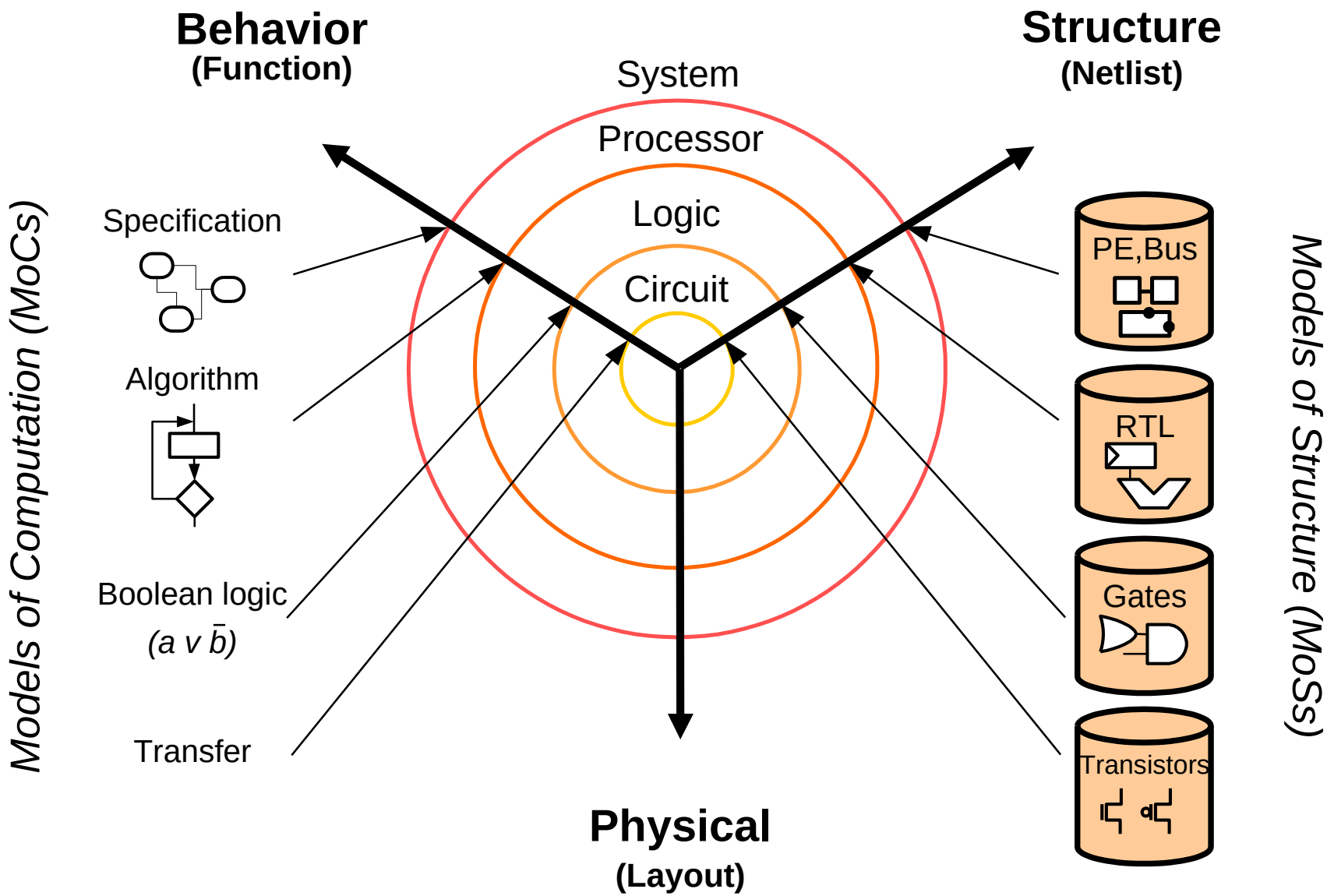
➢ **Design flow**

➢ Design automation for synthesis and verification



© 2009 A. Gerstlauer

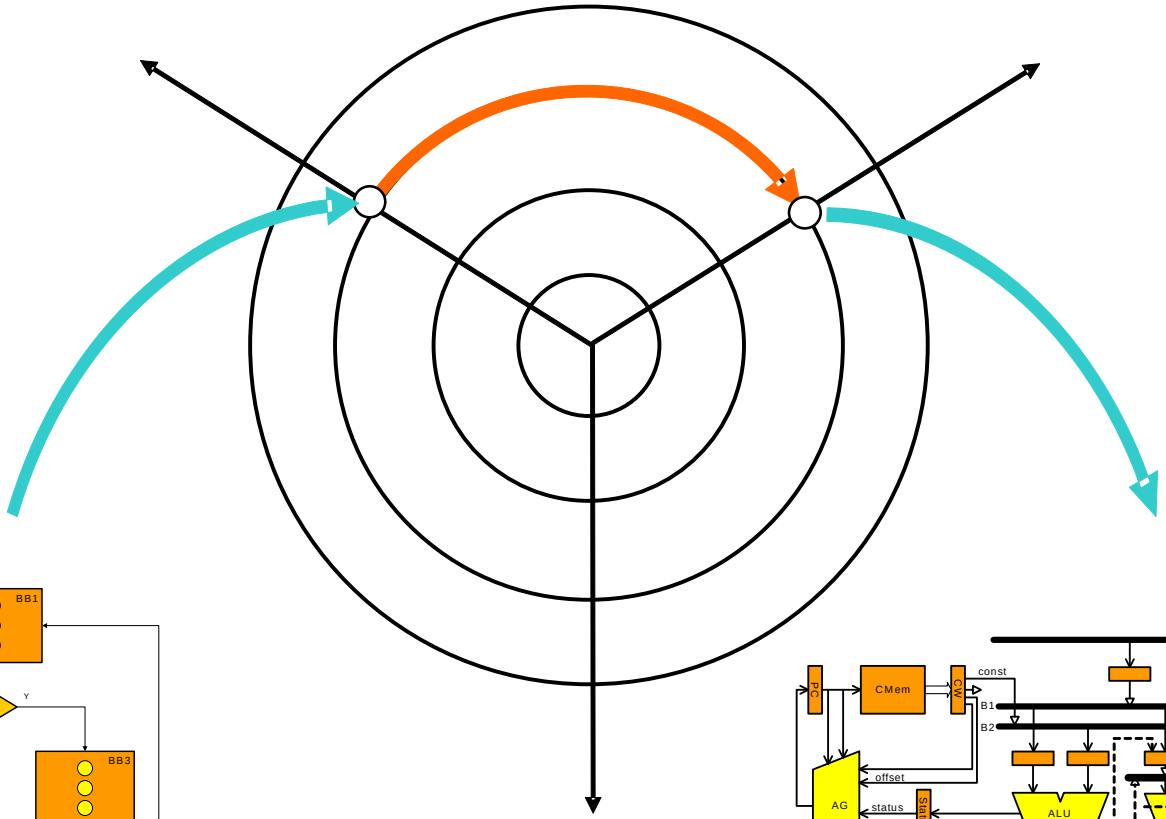
Y-Chart

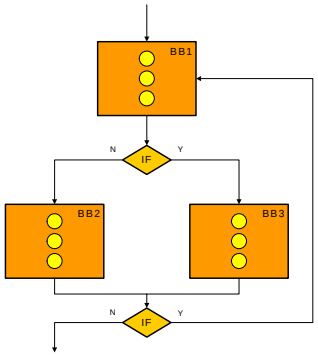


© 2009 A. Gerstlauer

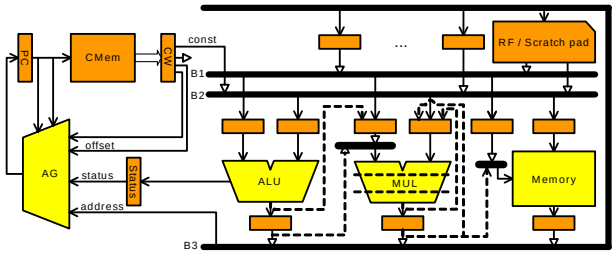
Processor Synthesis

- **Software processor**
  - Compilation and linking
- **Hardware processor**
  - High-level synthesis





Algorithm model (program)

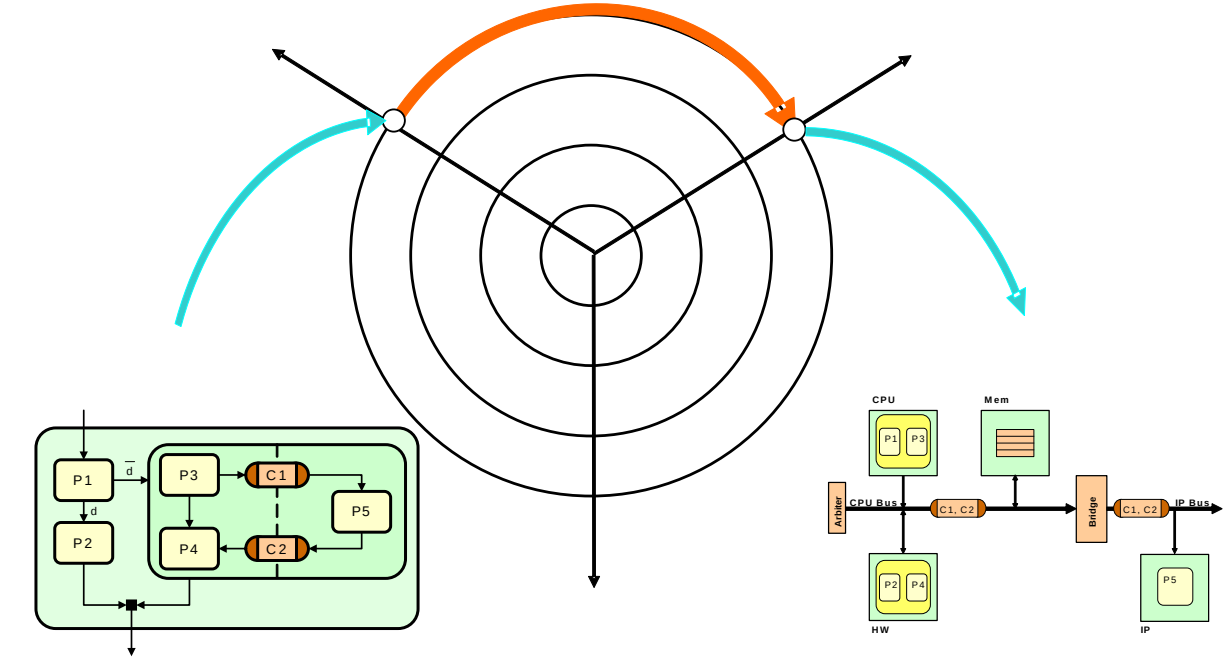


Microarchitecture model (RTL)

© 2009 A. Gerstlauer

System Synthesis

- **Structure**
  - Partitioning, mapping
- **Timing**
  - Scheduling



Specification model (processes) **Architecture model (TLM)**

Lecture 2: Outline

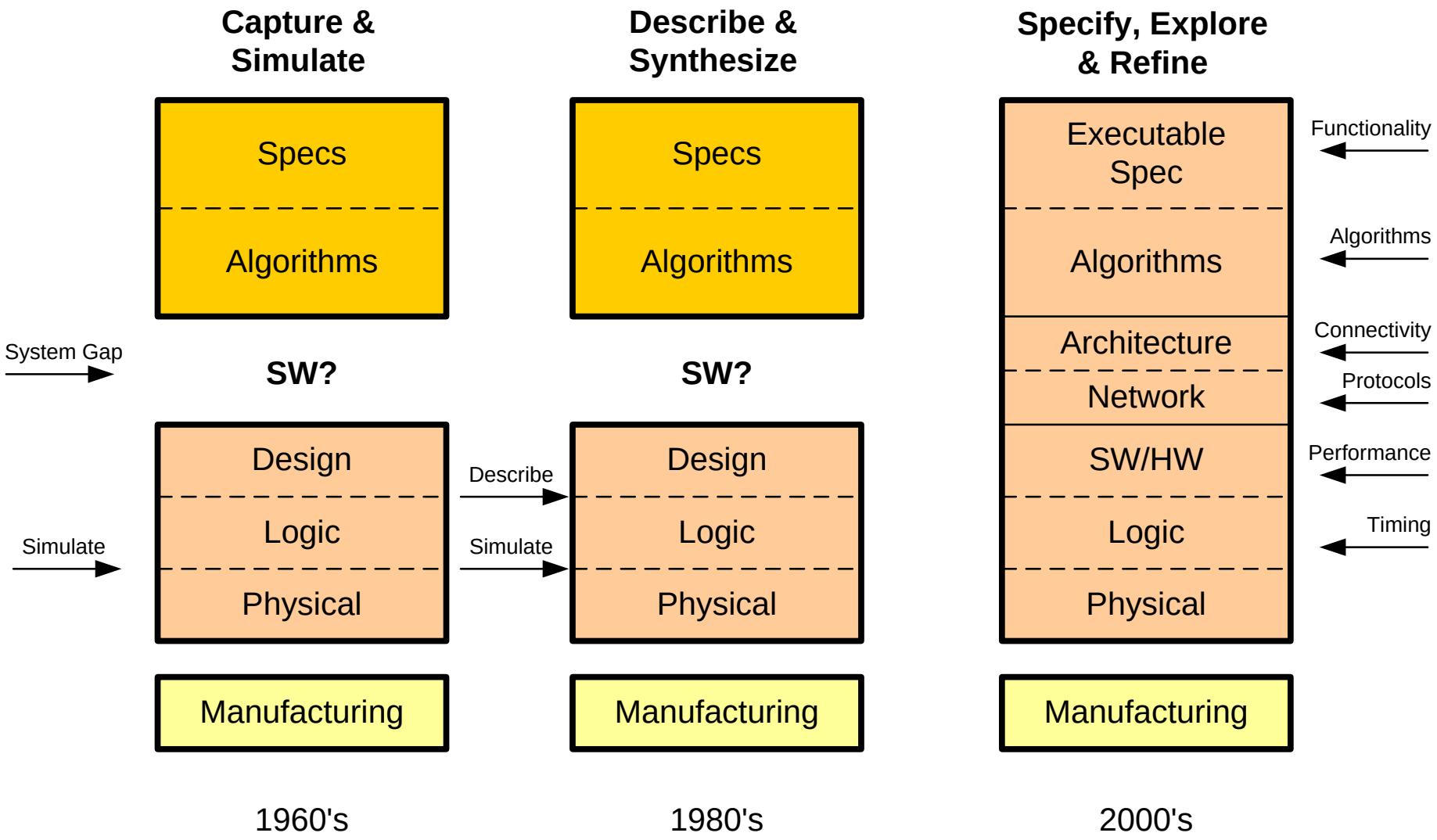
- ✓ Introduction

- **Methodologies**
  - Bottom-up, top-down, meet-in-the-middle, platform-based

- Modeling

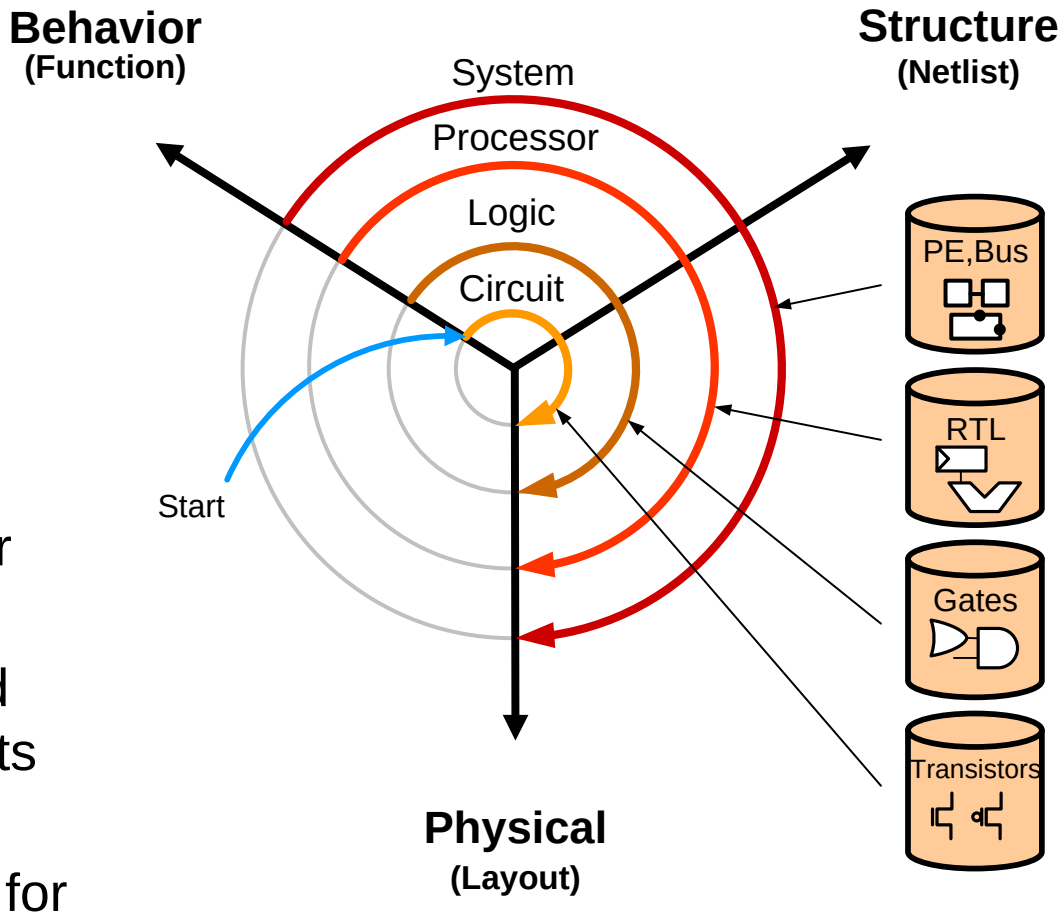
- System design languages

Evolution of Design Flows

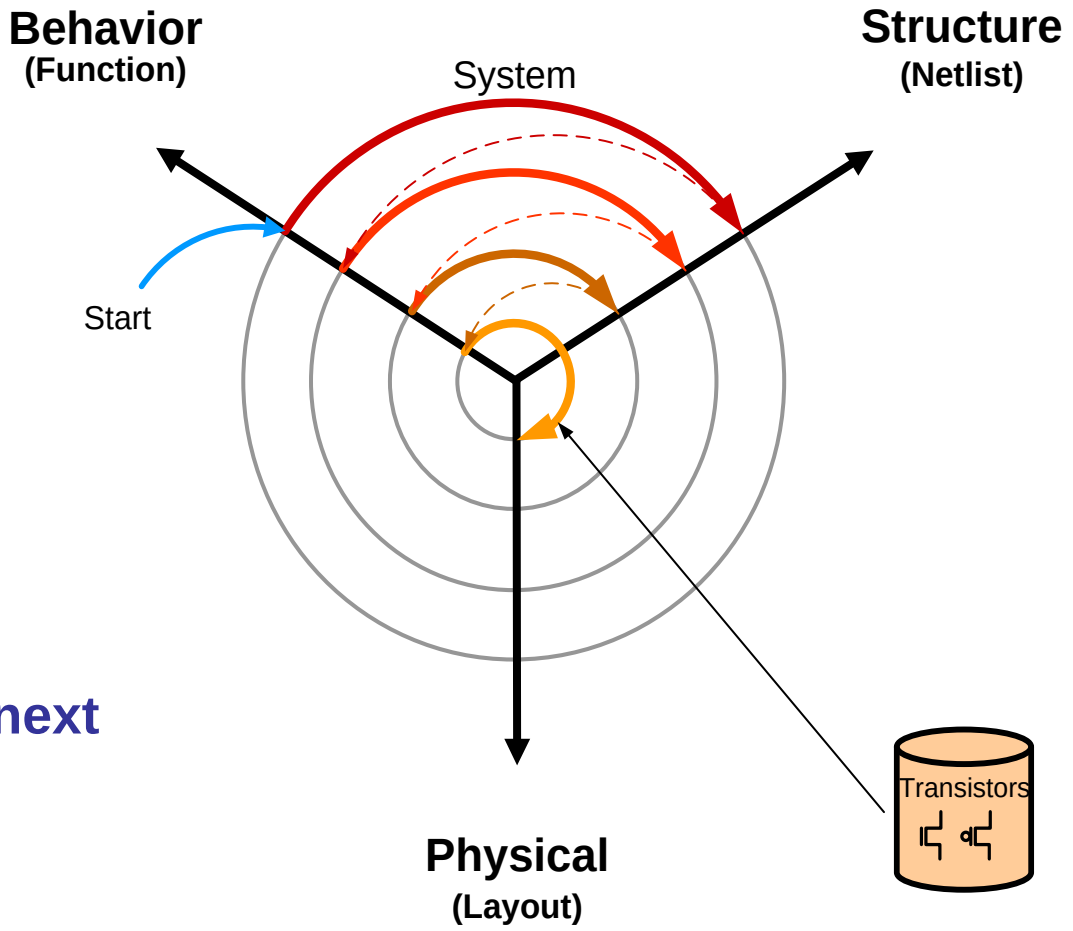


Bottom-Up Methodology

- **Each level generates library for the next higher level**
  - Circuit: Standard cells for logic level
  - Logic: RTL components for processor level
  - Processor: Processing and communication components for system level
  - System: System platforms for different applications
- **Floorplanning and layout on each level**



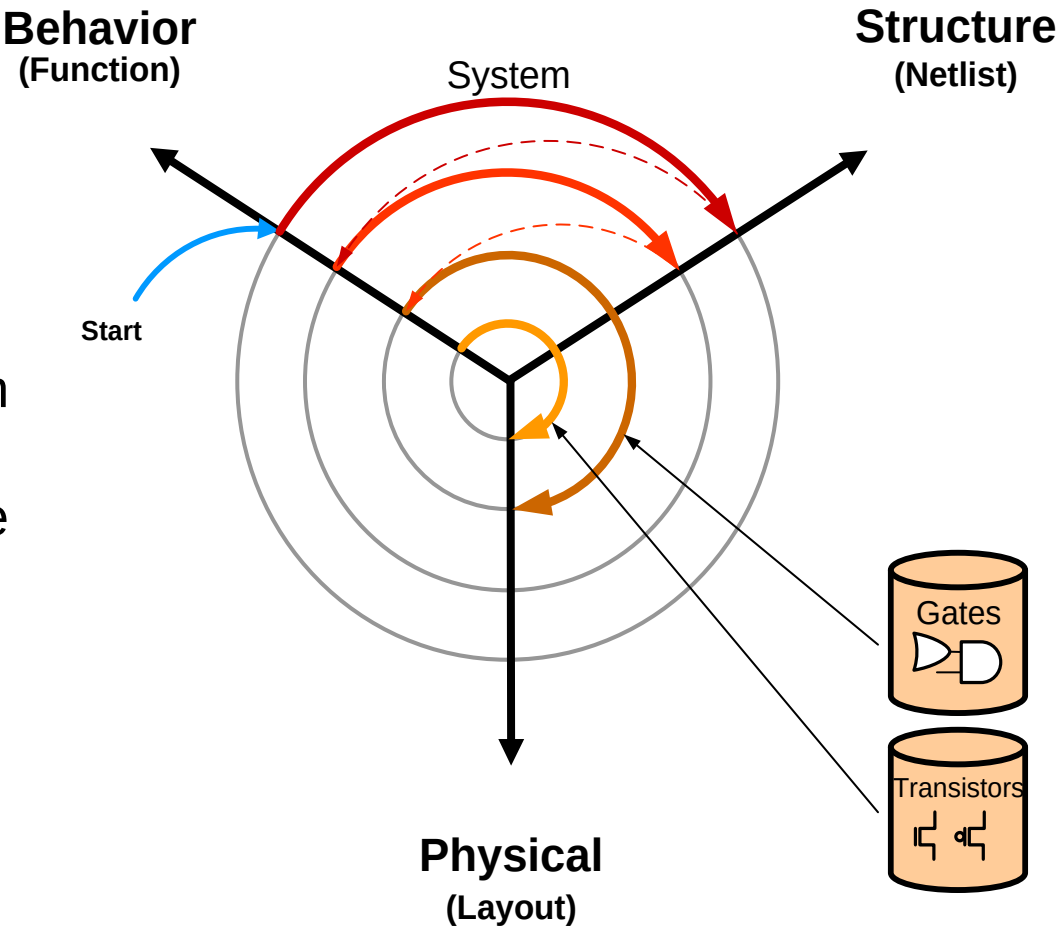
Top-down Methodology



- **Functional description is converted into component netlist on each level**
- **Each component function is decomposed further on the next abstraction level**
- **Layout is given only for transistor components**

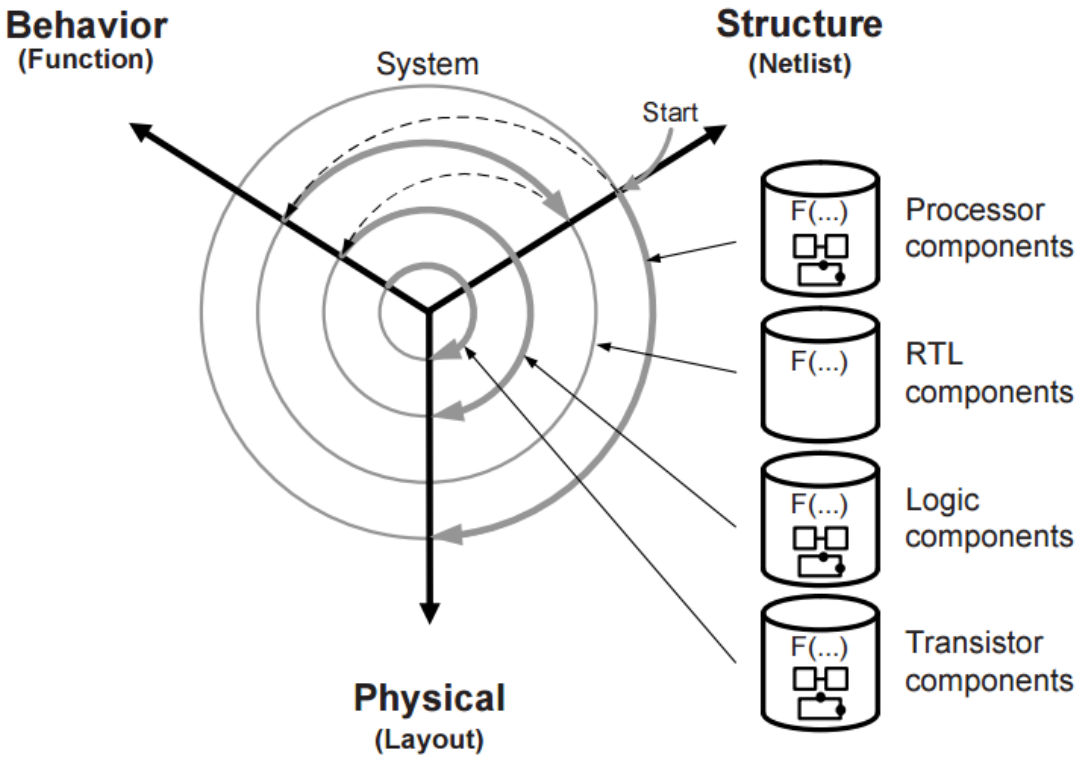
Meet-in-the-Middle Methodology

- **Gate netlist is hand-off**
- **Three levels of synthesis**
  - System is synthesized with processor components
  - Processor components are synthesized with RTL library
  - RTL components are synthesized with standard cells
- **Two levels of layout**
  - System layout is performed with standard cells
  - Standard cells layout with transistors



© 2009 A. Gerstlauer

Platform-Based Design



- **Meet-in-the-middle at the system level**
  - System platform with standard components
  - System design reduced to mapping of specification onto pre-defined platform

© 2009 A. Gerstlauer

Lecture 2: Outline

- ✓ Introduction
- ✓ Methodologies
- **Modeling**
  - System design flow
- System design languages

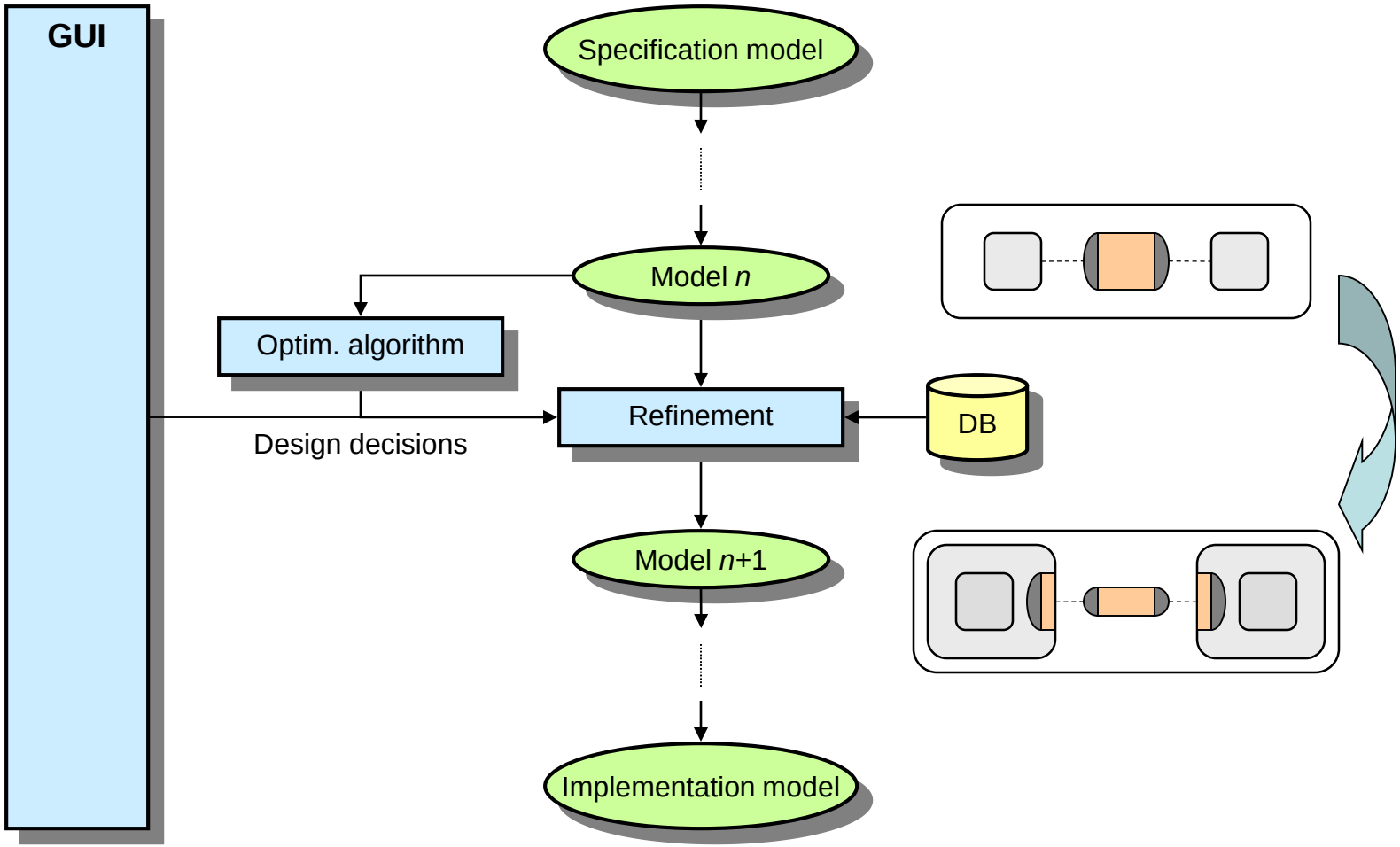
System Modeling

- **Basis of any design flow and design automation**
  - Inputs and outputs of design steps
    - Capability to capture complex systems
    - Precise, complete and unambiguous
  - Models at varying levels of abstraction
    - Level and granularity of implementation detail
    - Speed vs. accuracy
- **Design models as an abstraction of a design instance**
  - Representation of some aspect of reality
    - Virtual prototyping for validation through simulation or formal analysis
  - Specification for further implementation/synthesis
    - Describe desired functionality
  - Documentation & specification
    - Abstraction to hide details that are not relevant or not yet known
    - Different parts of the model or different use cases for the same model

© 2009 A. Gerstlauer

Design Automation

- **Synthesis = Decision making + model refinement**



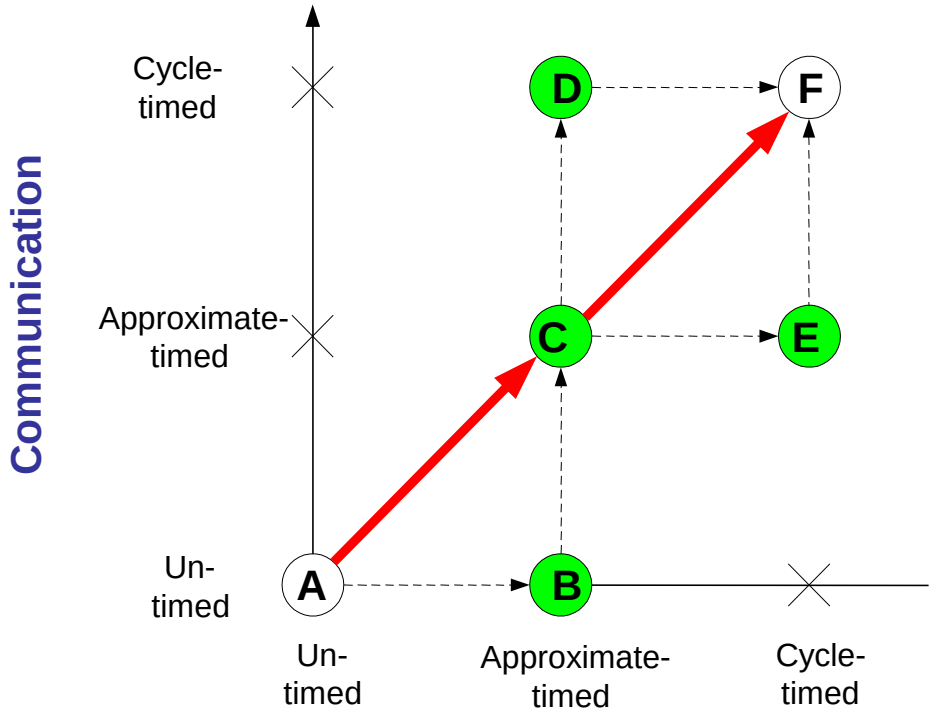
- **Successive, stepwise model refinement**
- **Layers of implementation detail**

System Design Flow

- **Abstraction based on level of detail & granularity**
  - Computation and communication

➢ **System design flow**

➢ Path from model A to model F



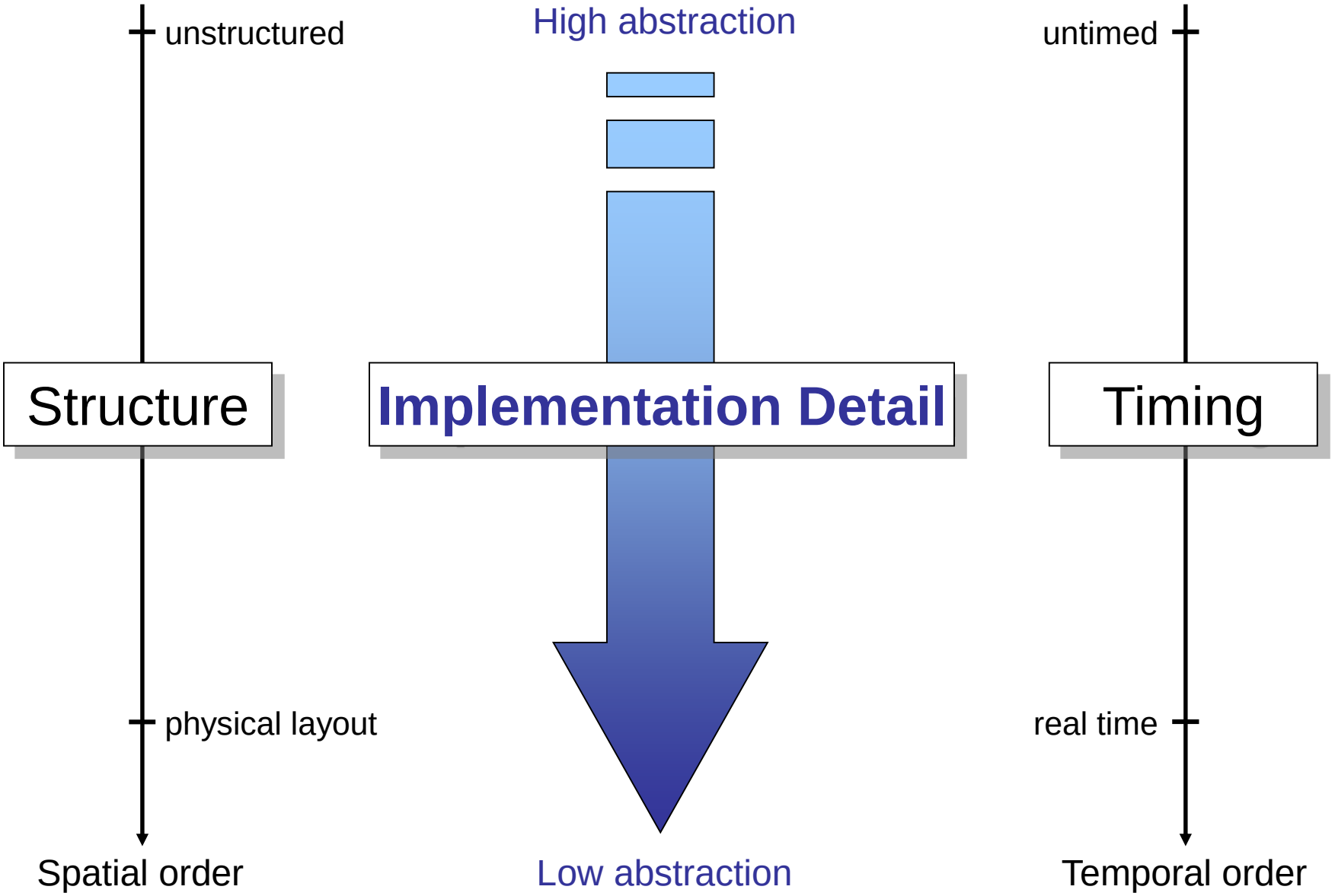
A. System specification model
B. Timed functional model
C. Transaction-level model (TLM)
D. Bus cycle-accurate model (BCAM)
E. Computation cycle-accurate model (CCAM)
F. Cycle-accurate model (CAM)

➢ **Design methodology and modeling flow**

➢ Set of models and transformations between models

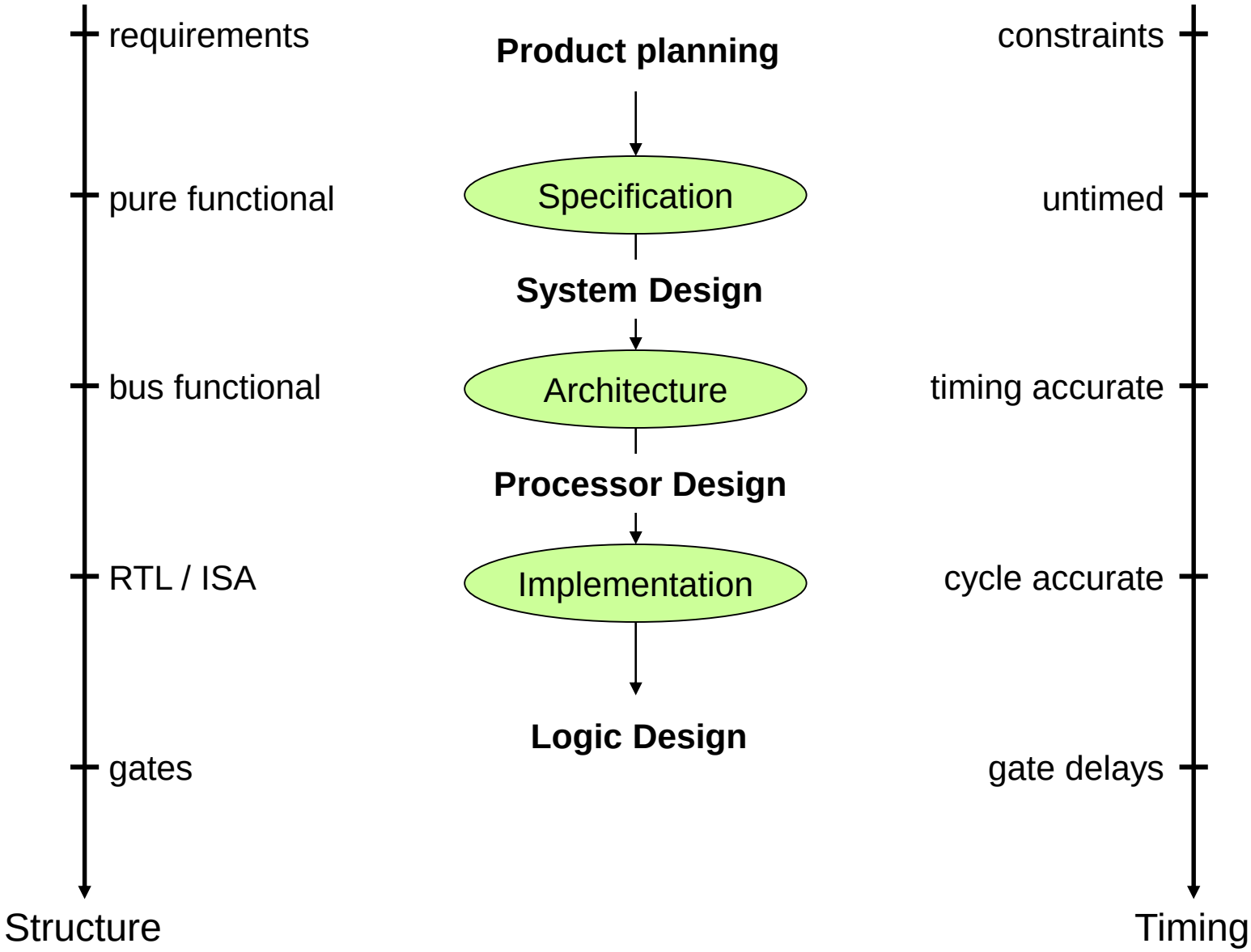
© 2009 A. Gerstlauer

Abstraction Levels



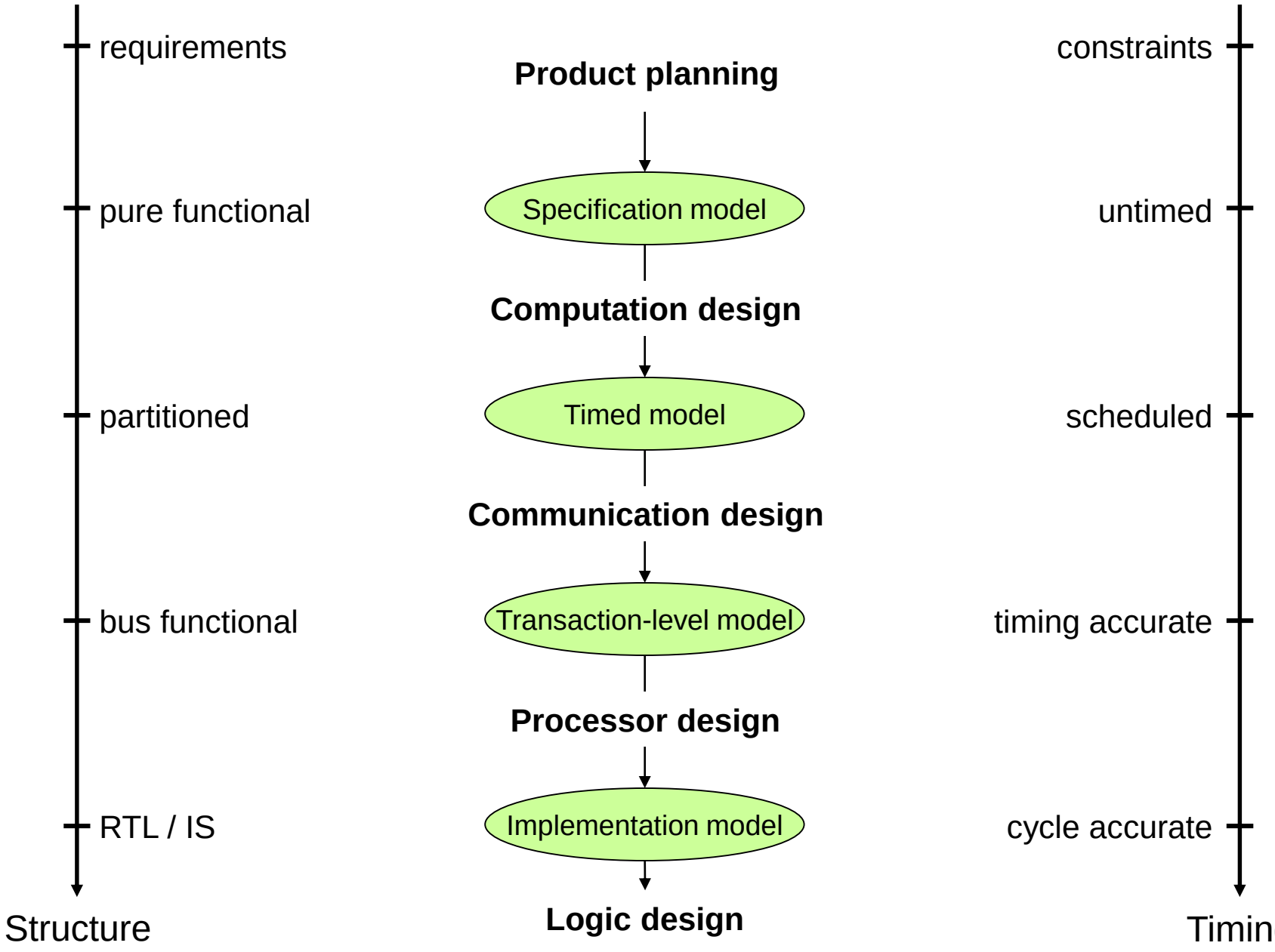
© 2009 A. Gerstlauer

Top-Down Design Flow



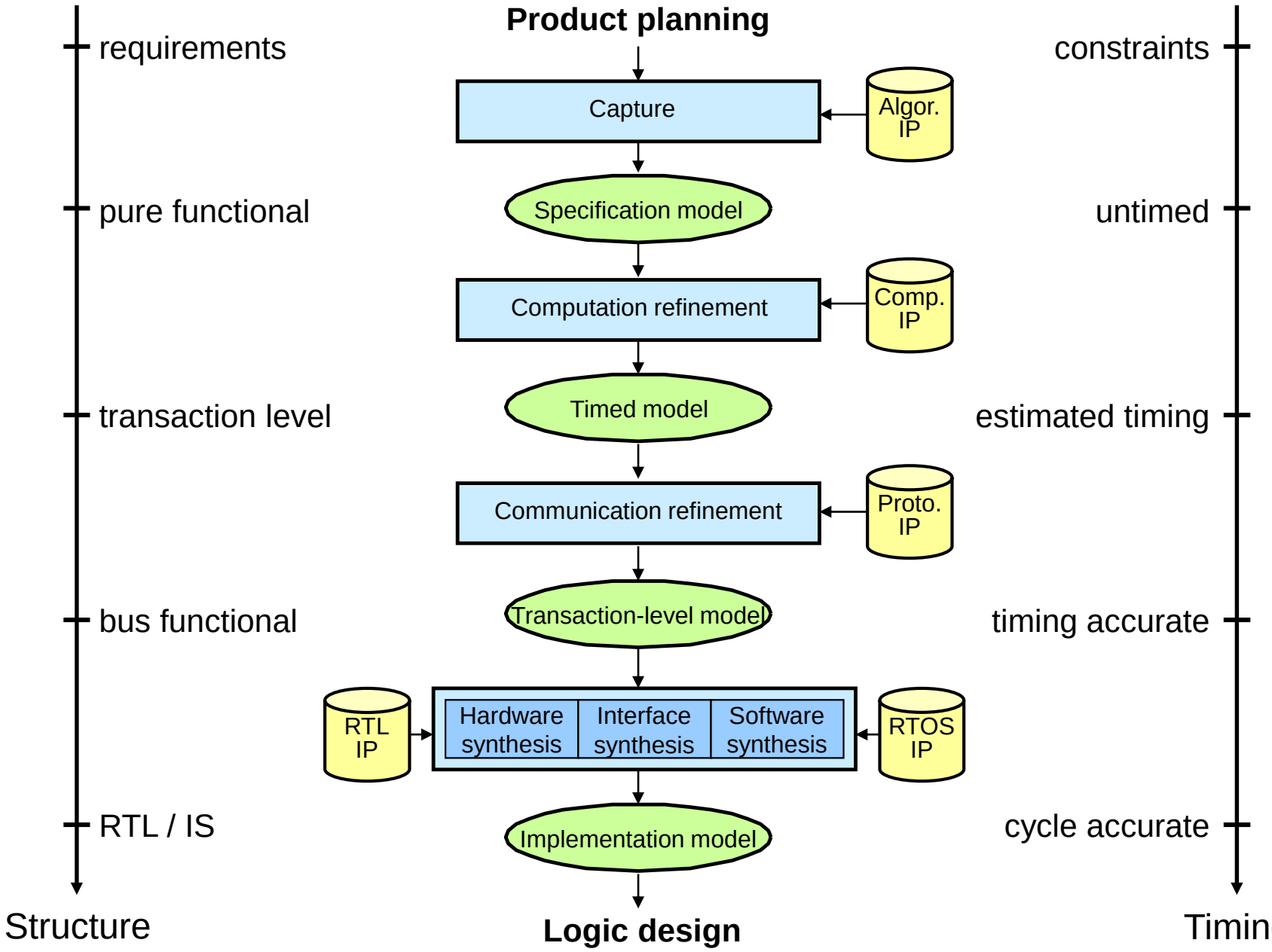
© 2009 A. Gerstlauer

Top-Down Design Flow



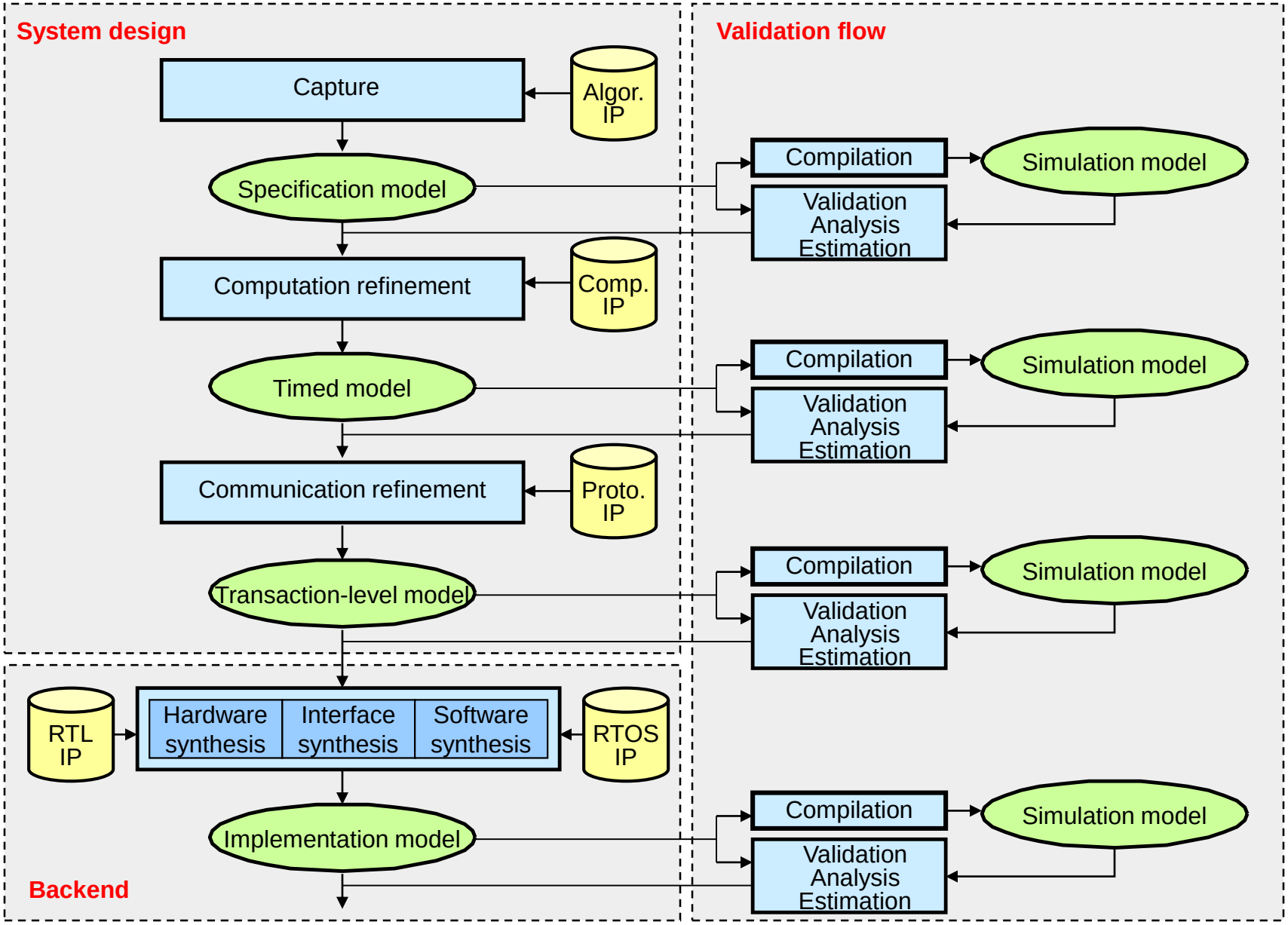
© 2009 A. Gerstlauer

Top-Down Design Flow



© 2009 A. Gerstlauer

Design Methodology



© 2009 A. Gerstlauer

Lecture 2: Outline

- ✓ Introduction
- ✓ Methodologies
- ✓ Modeling
- **System design languages**
  - Goals, requirements
  - Communication and computation

Languages

- **Represent a model in machine-readable form**
  - Apply algorithms and tools
- **Syntax defines grammar**
  - Possible strings over an alphabet
  - Textual or graphical
- **Semantics defines meaning**
  - Mapping onto an abstract state machine model
    - Operational semantics
  - Mapping into a mathematical domain (e.g. functions)
    - Denotational semantics
- **Semantic model vs. design models**
  - Basic semantic models can represent many design models
    - Discrete event model for hardware and system simulation
  - Design models can be represented in different languages

© 2009 A. Gerstlauer

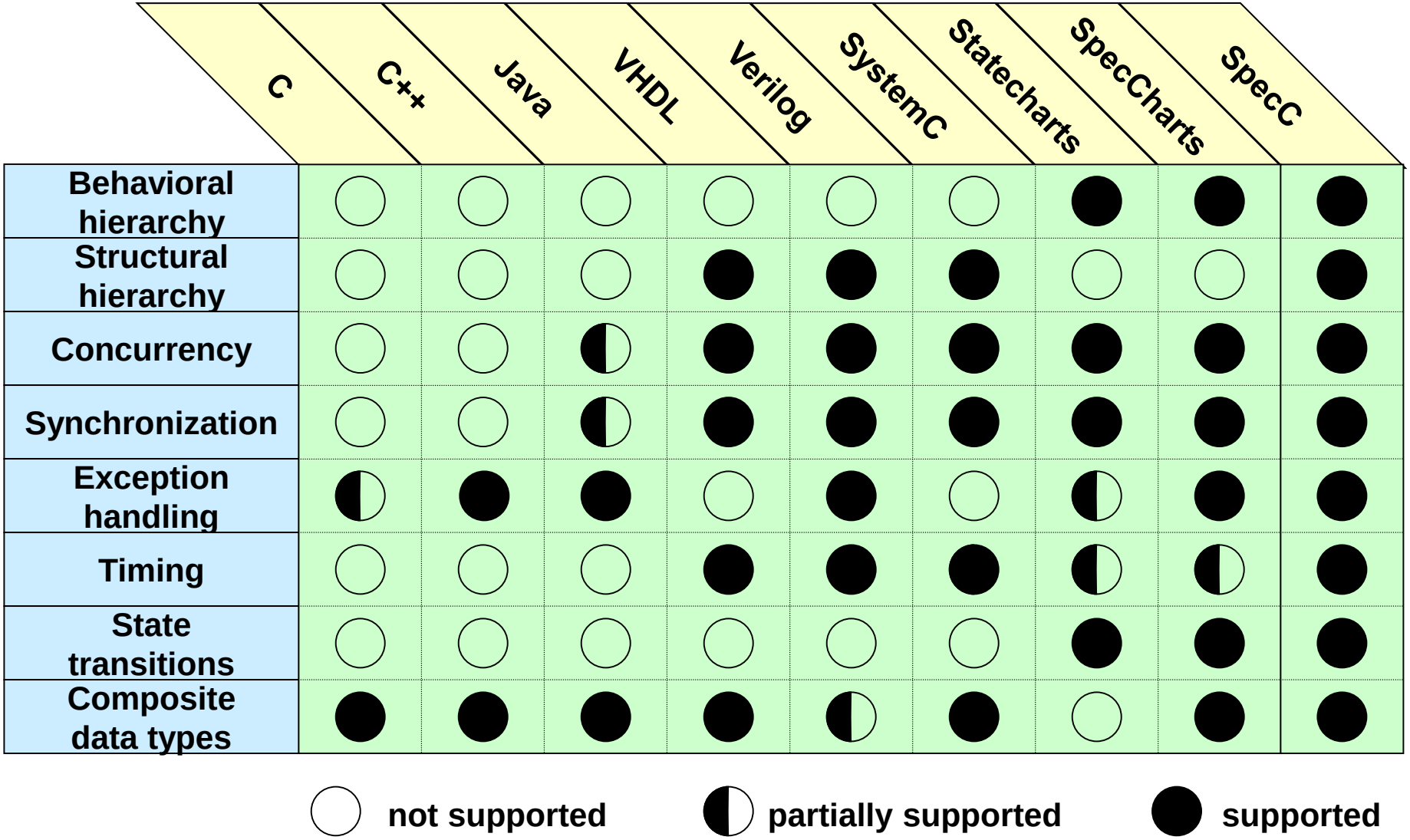
Evolution of Design Languages

- **Netlists**
  - Structure only: components and connectivity
  - ➢ Gate-level [EDIF], system-level [SPIRIT/XML]
- **Hardware description languages (HDLs)**
  - Event-driven behavior: signals/wires, clocks
  - Register-transfer level (RTL): boolean logic
  - ➢ Discrete event [VHDL, Verilog]
- **System-level design languages (SLDLs)**
  - Software behavior: sequential functionality/programs
  - ➢ C-based, event-driven [SpecC, SystemC, SystemVerilog]

- **Goals**
  - Executability
    - Validation through simulation
  - Synthesizability
    - Implementation in HW and/or SW
    - Support for IP reuse
  - Modularity
    - Hierarchical composition
    - Separation of concepts
  - Completeness
    - Support for all concepts found in embedded systems
  - Orthogonality
    - Orthogonal constructs for orthogonal concepts
    - Minimality
  - Simplicity

Source: R. Doemer, UC Irvine

- **Requirements**



Source: R. Doemer, UC Irvine

System-Level Design Languages (SLDLs)

- **C/C++**
  - ANSI standard programming languages, software design
  - Traditionally used for system design because of practicality, availability
- **SystemC**
  - C++ API and class library
  - Standardized by Open SystemC Initiative (OSCI), IEEE Std 1666™-2005
- **SpecC**
  - C extension
  - Developed at UC Irvine, standard by SpecC Technology Open Consortium (STOC)
- **SystemVerilog**
  - Verilog with C extensions for testbench development
- **Matlab/Simulink**
  - Specification and simulation in engineering, algorithm design
- **Unified Modeling Language (UML)**
  - Software specification, graphical, extensible (meta-modeling)
  - Modeling and Analysis of Real-time and Embedded systems (MARTE) profile
- **IP-XACT**
  - XML schema for IP component documentation, standard by SPIRIT consortium
- **Rosetta (formerly SLDL)**
  - Formal specification of constraints, requirements
- **SDL**
  - Telecommunication area, standard by ITU
- **…**

Source: R. Doemer, UC Irvine

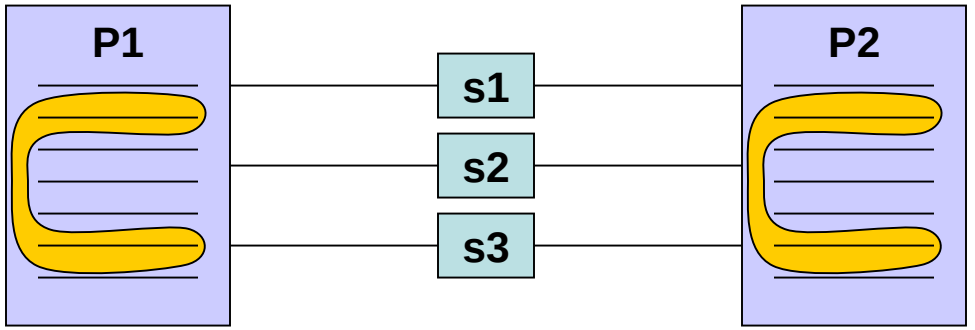
Separation of Concerns

- **Fundamental principle in modeling of systems**
- **Clear separation of concerns**
  - Address separate issues independently
- **System-Level Description Language (SLDL)**
  - Orthogonal concepts
  - Orthogonal constructs
- **System-level Modeling**
  - Computation
    - encapsulated in modules / behaviors
  - Communication
    - encapsulated in channels

Source: R. Doemer, UC Irvine

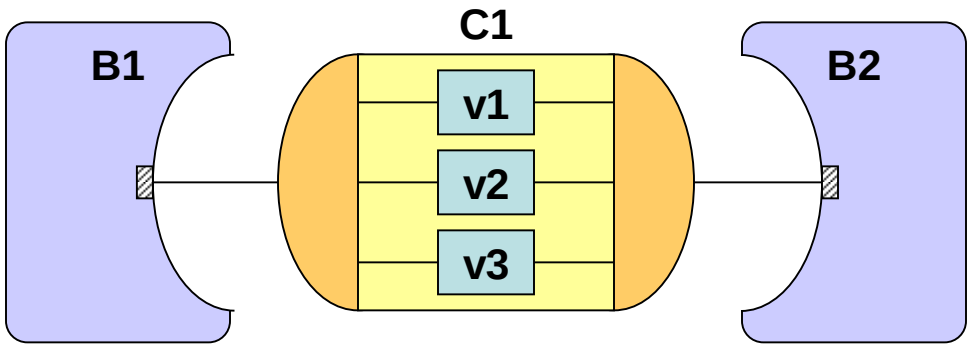
Computation vs. Communication

- **Traditional model**



  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible

- **SpecC model**

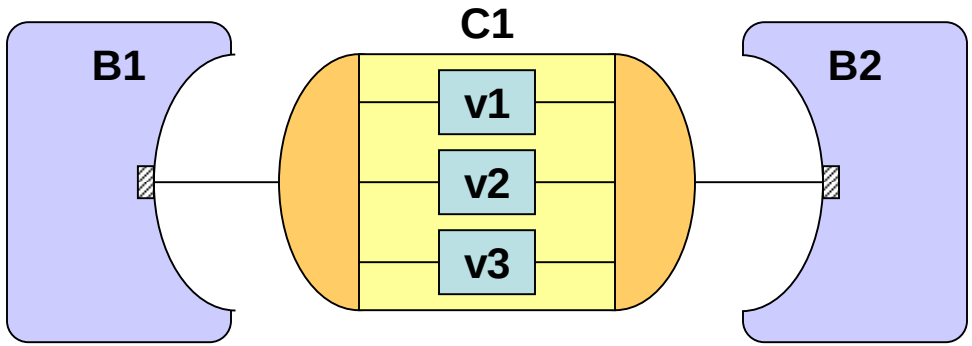


  - Behaviors and channels
  - Separation of computation and communication
  - Plug-and-play

Source: R. Doemer, UC Irvine

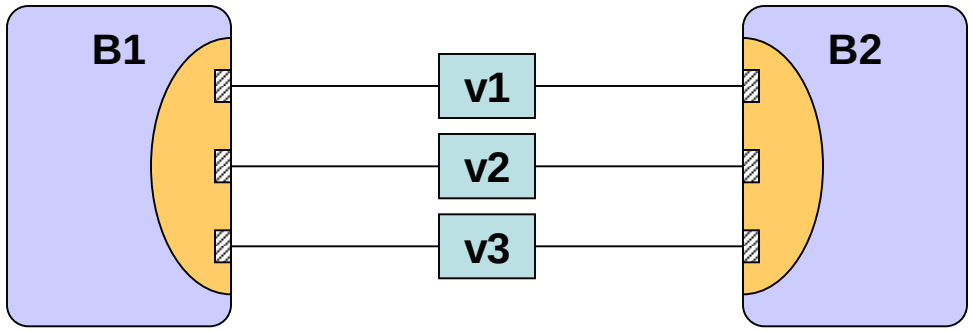
Computation vs. Communication

- **Protocol Inlining**
  - Specification model
  - Exploration model



    - Computation in behaviors
    - Communication in channels

  - Implementation model

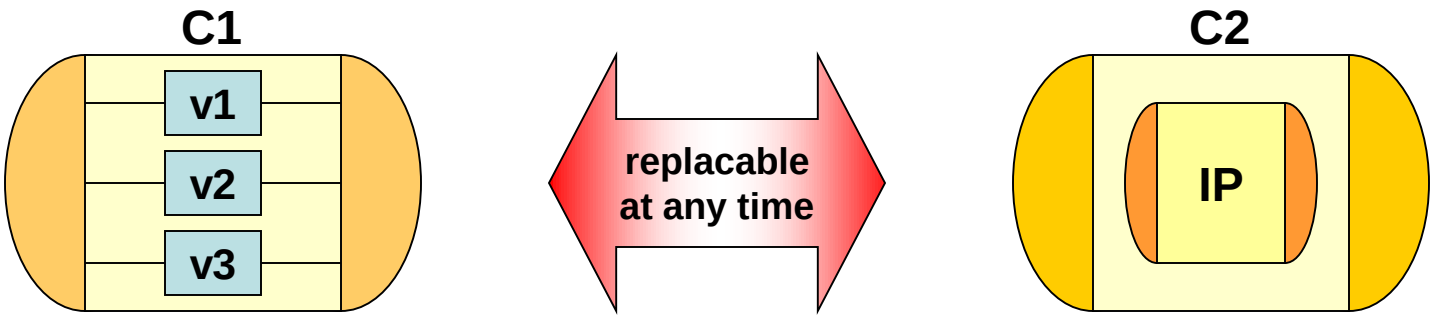


    - Channel disappears
    - Communication inlined into behaviors
    - Wires exposed

Source: R. Doemer, UC Irvine

Intellectual Property (IP)

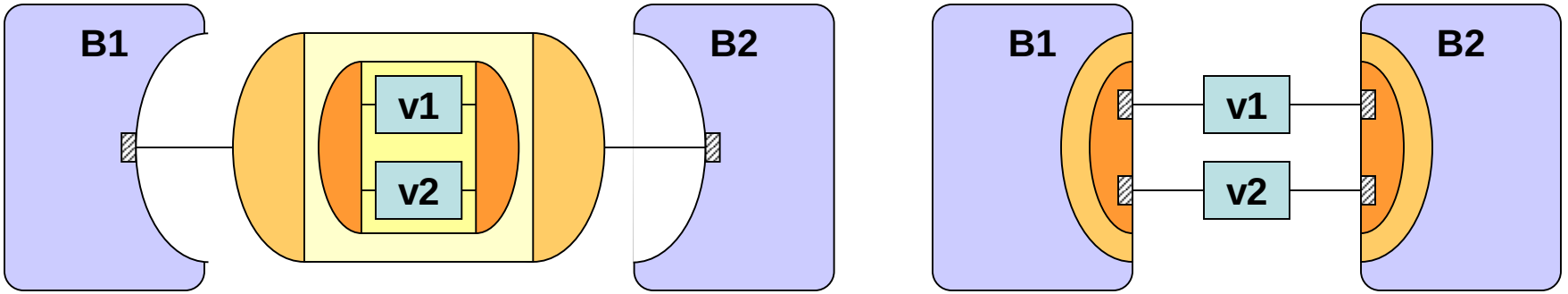
- **Communication IP: Channel with wrapper**



Virtual channel

IP protocol channel in wrapper

- **Protocol inlining with hierarchical channel**



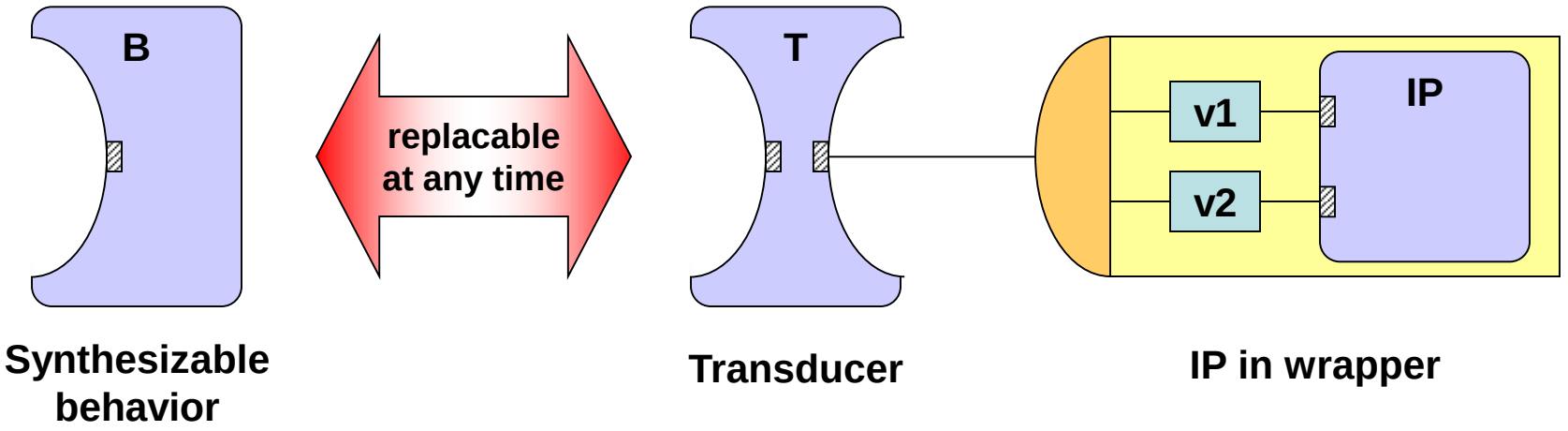
before

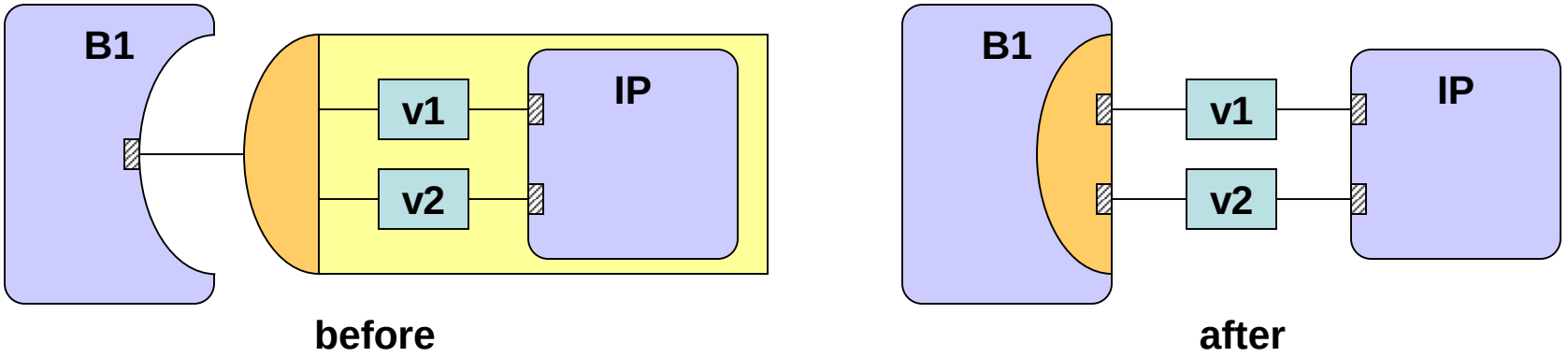
after

Source: R. Doemer, UC Irvine

Intellectual Property (IP)

- **Computation IP: Wrapper model**

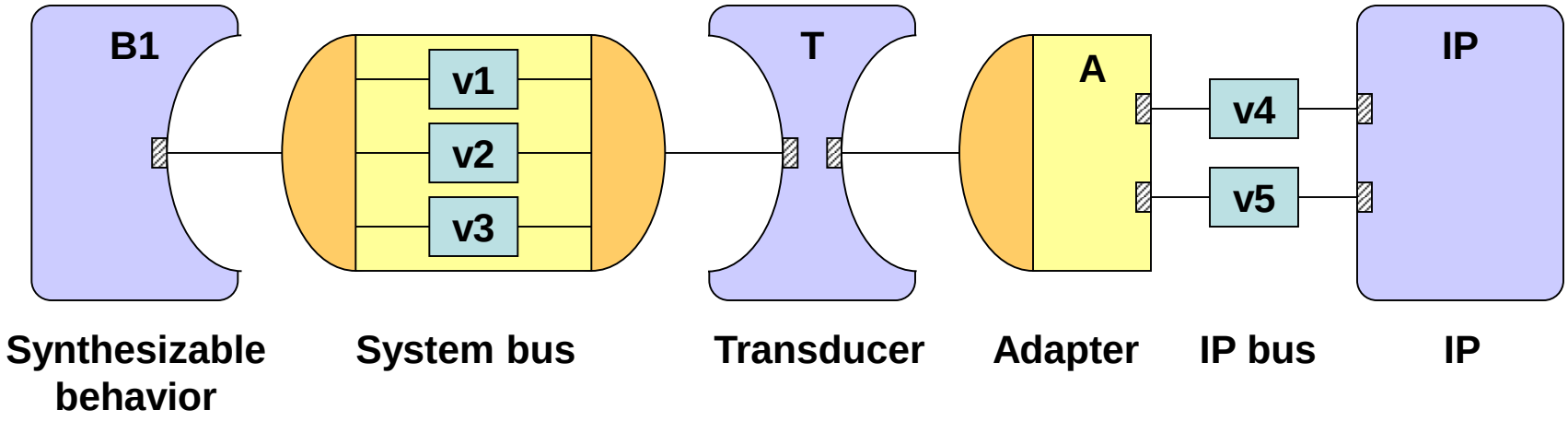




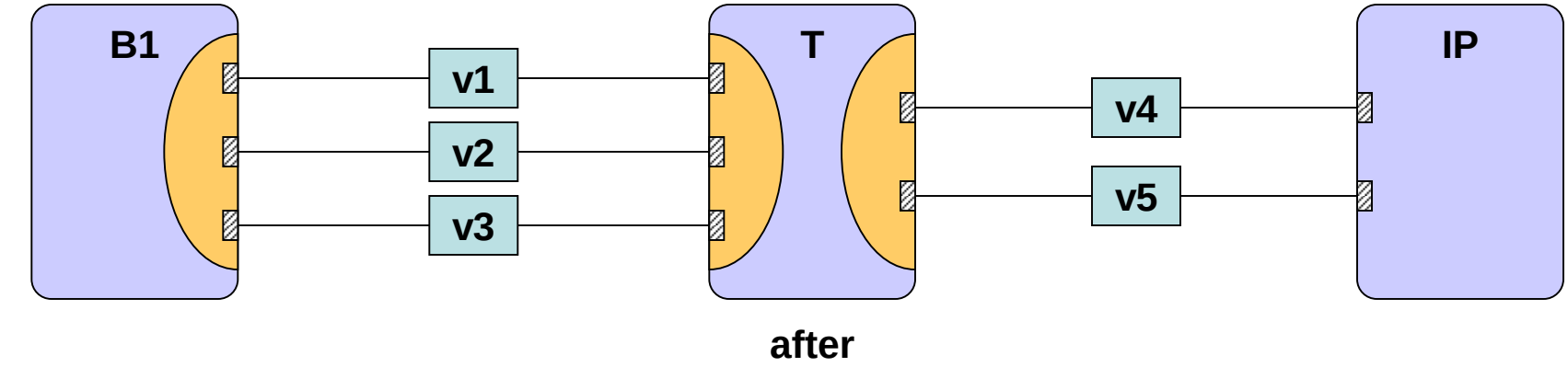
Source: R. Doemer, UC Irvine

Intellectual Property (IP)

- **Incompatible busses: Transducer insertion**



- **Protocol inlining with transducer**



Source: R. Doemer, UC Irvine