System-Level Design (and Modeling for Embedded Systems)

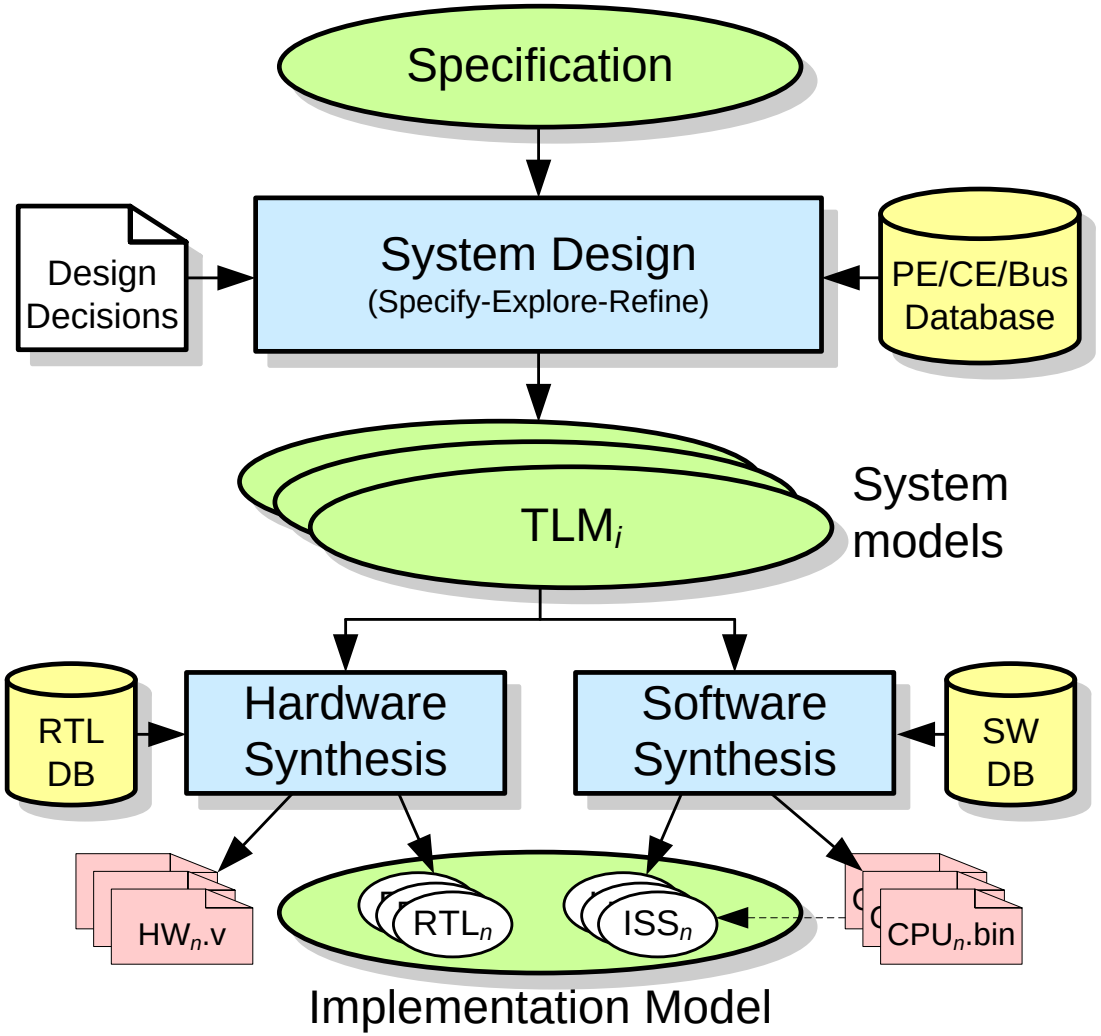
Lecture 5 – Specification and Refinement

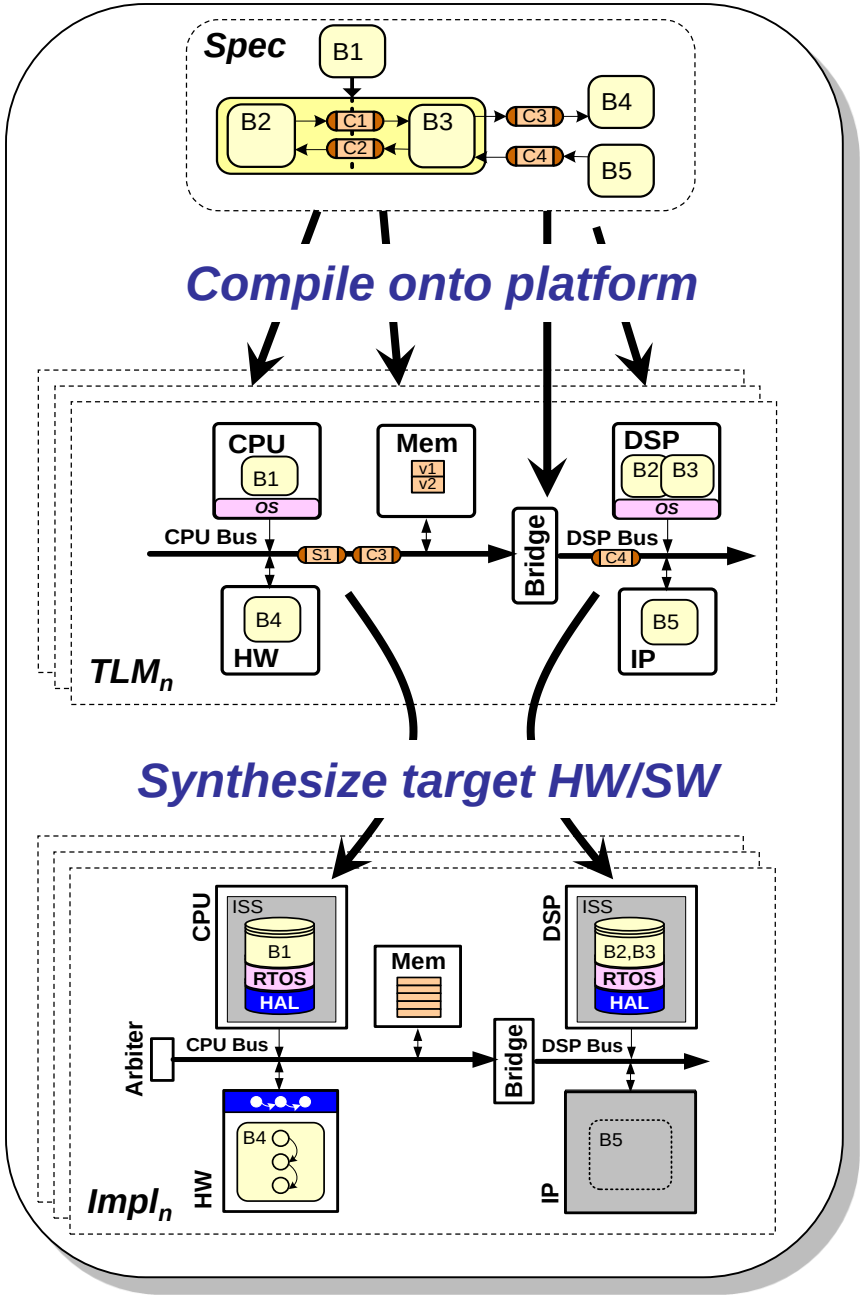
Kim Grüttner kim.gruettner@dlr.de
Henning Schlender henning.schlender@dlr.de
Jörg Walter joerg.walter@offis.de

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

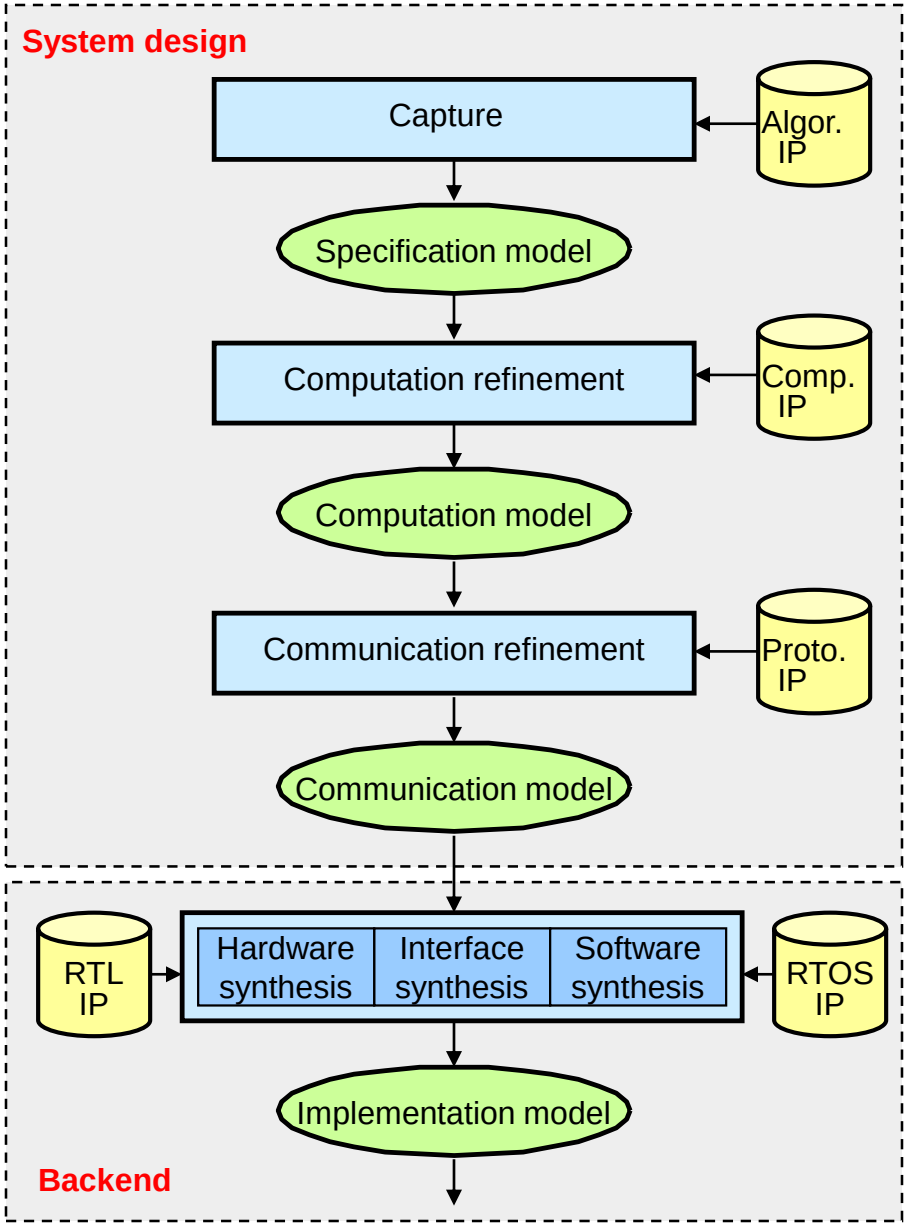
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

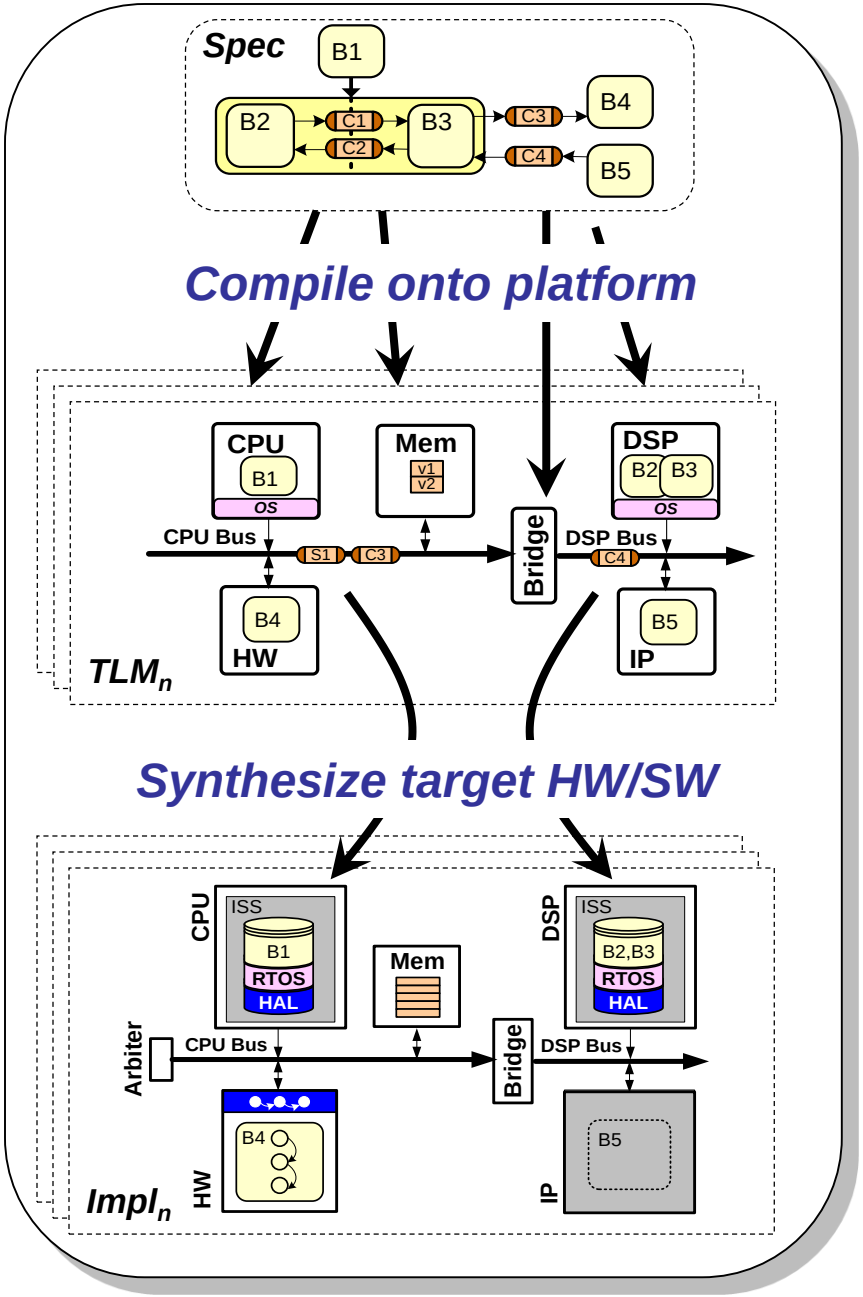
System-On-Chip Design Flow





System-On-Chip Design Flow

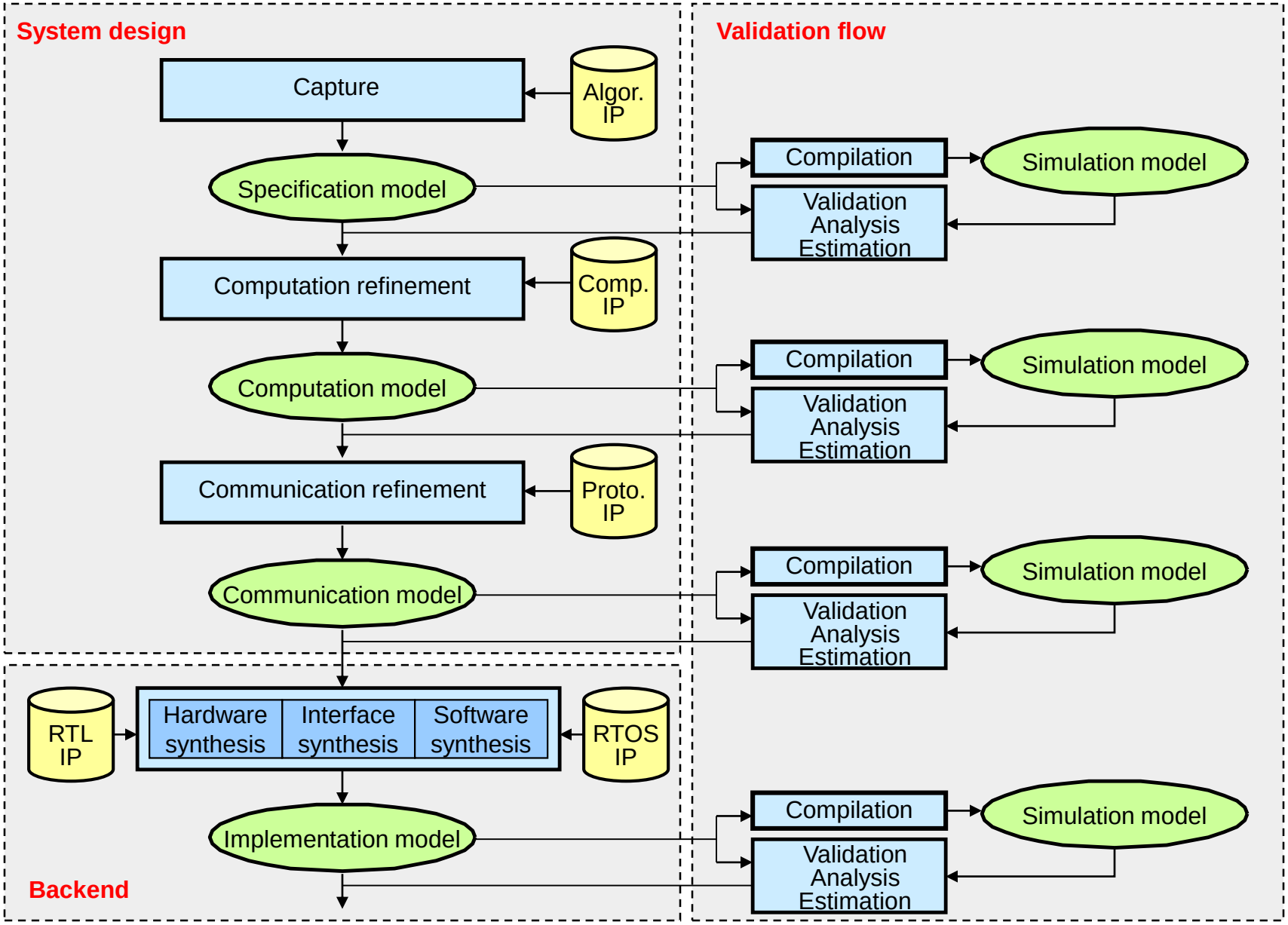




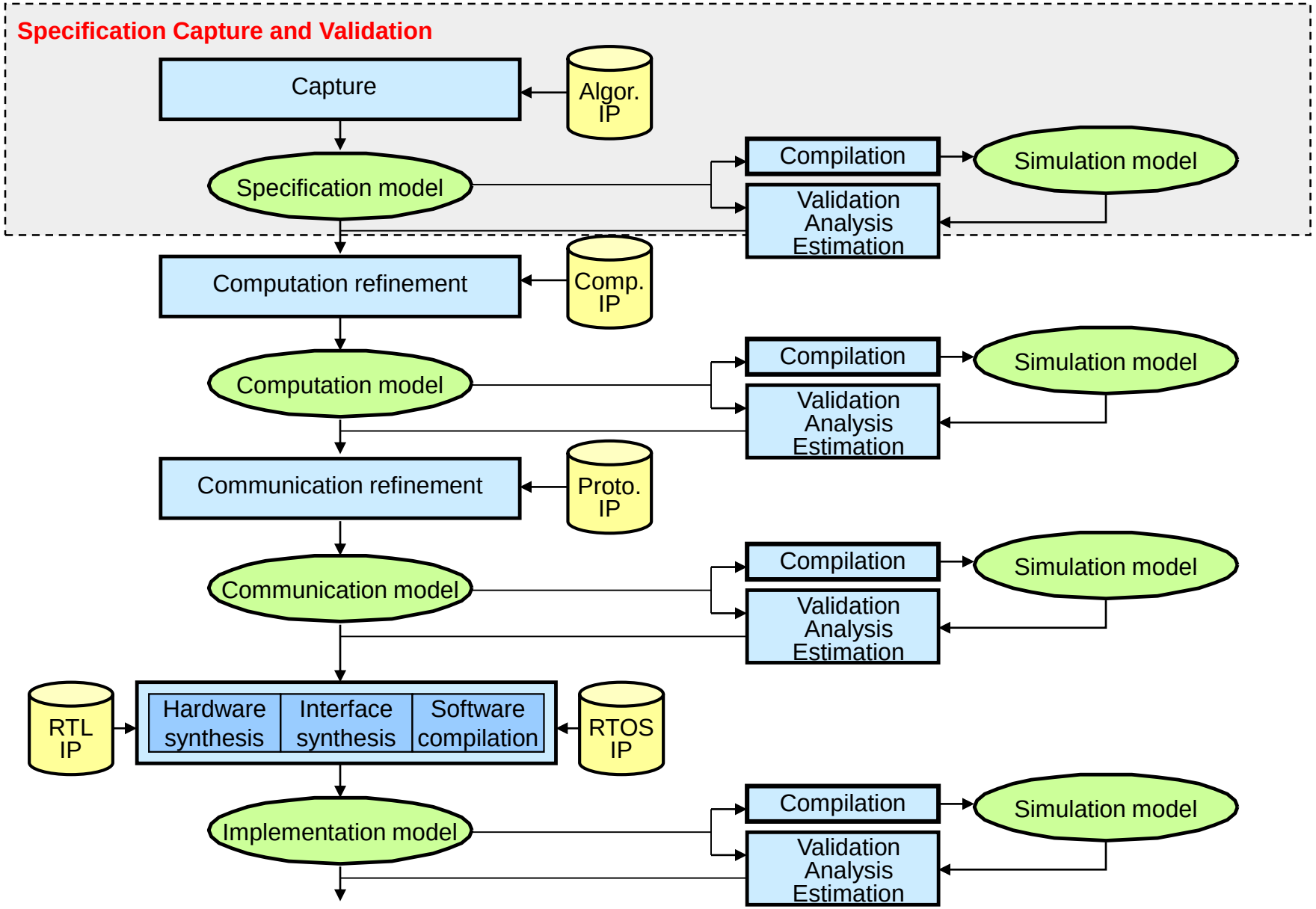
Lecture 5: Outline

- **System specification**
  - Specification modeling
  - System validation
- **System refinement**
  - Computation
  - Communication
  - Implementation
- **SCE design environment**
  - Modeling
  - Refinement
  - Synthesis

Design Methodology



Design Methodology



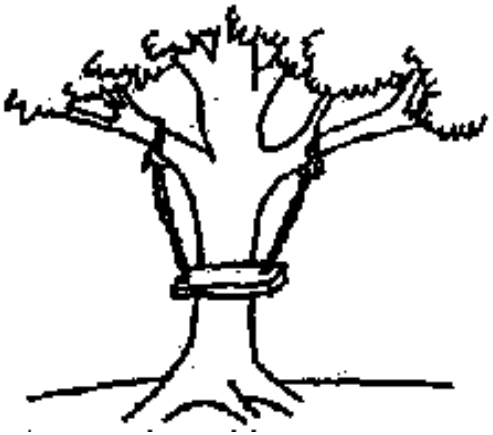
Essential Issues in Specification

- **An Example ...**

Proposed by the project team

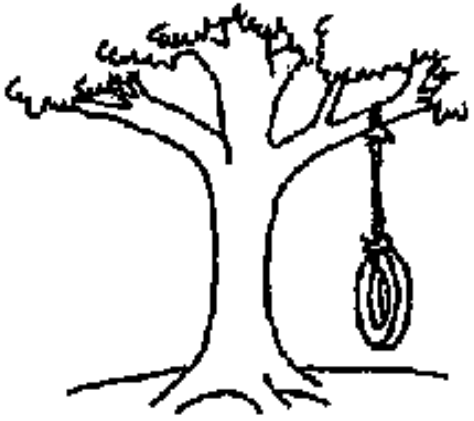
Product specification

Product design by senior analyst



Product after implementation

Product after acceptance by user



What the user wanted

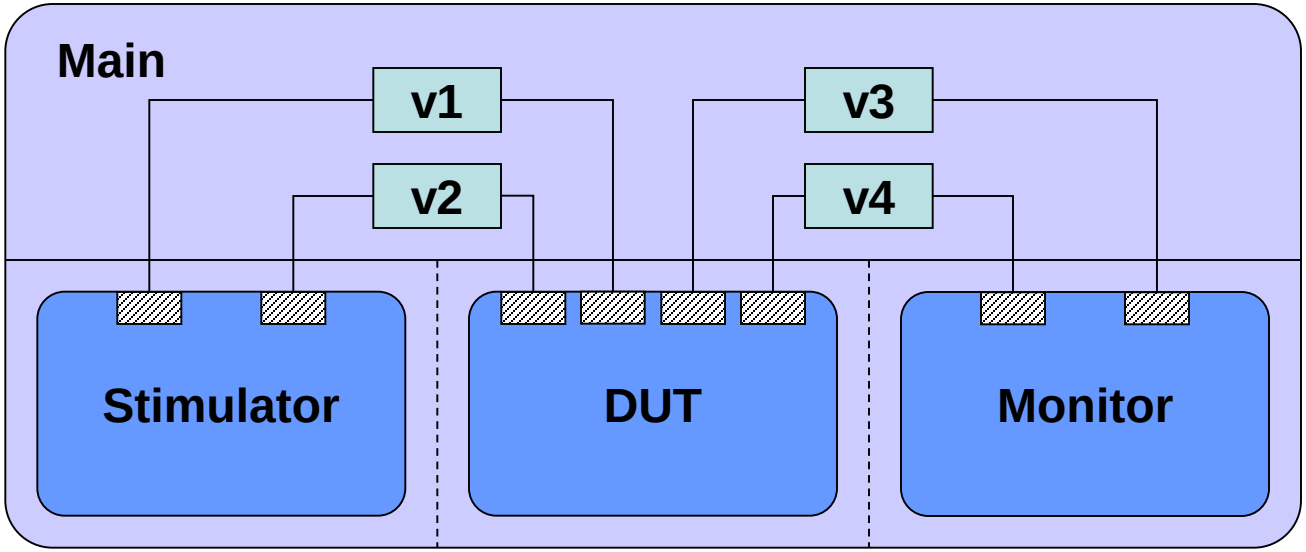
Source: unknown author, Courtesy of: R. Doemer

Specification Model

- **Functional and executable**
  - “golden model” (first functional model in the design flow)
  - all other models will be derived from and compared to this one
- **High abstraction level**
  - no implementation details
  - unrestricted exploration of design space
- **Separation of communication and computation**
  - channels and behaviors
- **Pure functional**
  - no structural information
- **No timing**
  - exception: timing constraints

Specification Model

- **Test bench**
  - Main, Stimulator, Monitor
  - no restrictions in syntax and semantics (no synthesis)
- **Design under test**
  - DUT
  - restricted by syntax and semantic rules (synthesis!)



Source: R. Doemer, UC Irvine

Specification Modeling Guidelines

- **Computation: Behaviors**
  - Hierarchy: explicit concurrency, state transitions, ...
  - Granularity: leaf behaviors = smallest indivisible units
  - Encapsulation: localization, explicit dependencies
  - Concurrency: explicitly specified (par, pipe, fsm, seq)
  - Time: un-timed, partial ordering

- **Communication: Channels**
  - Semantics: abstract communication, synchronization (standard channel library)
  - Dependencies: explicit data dependency, partial ordering, port connectivity

Specification Modeling Guidelines

- **Example rules for SoC Environment (SCE)**
  - Clean behavioral hierarchy
    - hierarchical behaviors:
      no code other than **`seq`**, **`par`**, **`pipe`**, **`fsm`** statements
    - leaf behaviors:
      no SpecC code (pure ANSI-C code only)
  - Clean communication
    - point-to-point communication via standard channels:
      `c_handshake`, **`c_semaphore`**,
      `c_double_handshake`, **`c_queue`** (typed or untyped)
    - ports of plain ANSI C type or interface type, no pointers!
    - port maps to local variables or ports only

- **Detailed rules for SoC Environment**
  - "*SCE Specification Model Reference Manual,*"
    by A. Gerstlauer, R. Doemer, CECS, UC Irvine, April 2005

Specification Modeling Guidelines

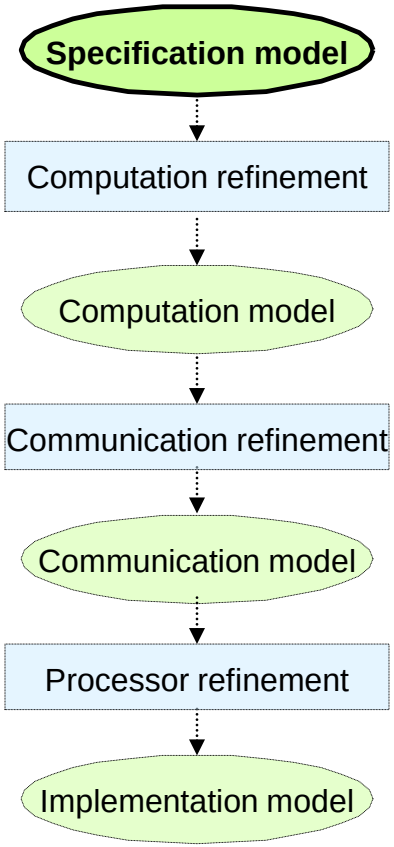
- **C code conversion to SpecC**
  - Functions become behaviors or channels
  - Functional hierarchy becomes behavioral hierarchy
    - Clean behavioral hierarchy required
    - if-then-else structure becomes FSM
    - while/for/do loops become FSM
  - Explicitly specify potential parallelism
    - Data (array) partitioning
  - Explicitly specify communication
    - Avoid global variables
    - Use local variables and ports (signals, wires)
    - Use standard channels
  - Data types
    - Avoid pointers, use arrays instead
    - Use explicit SpecC data types if suitable
    - Floating-point to fixed-point conversion

Lecture 5: Outline

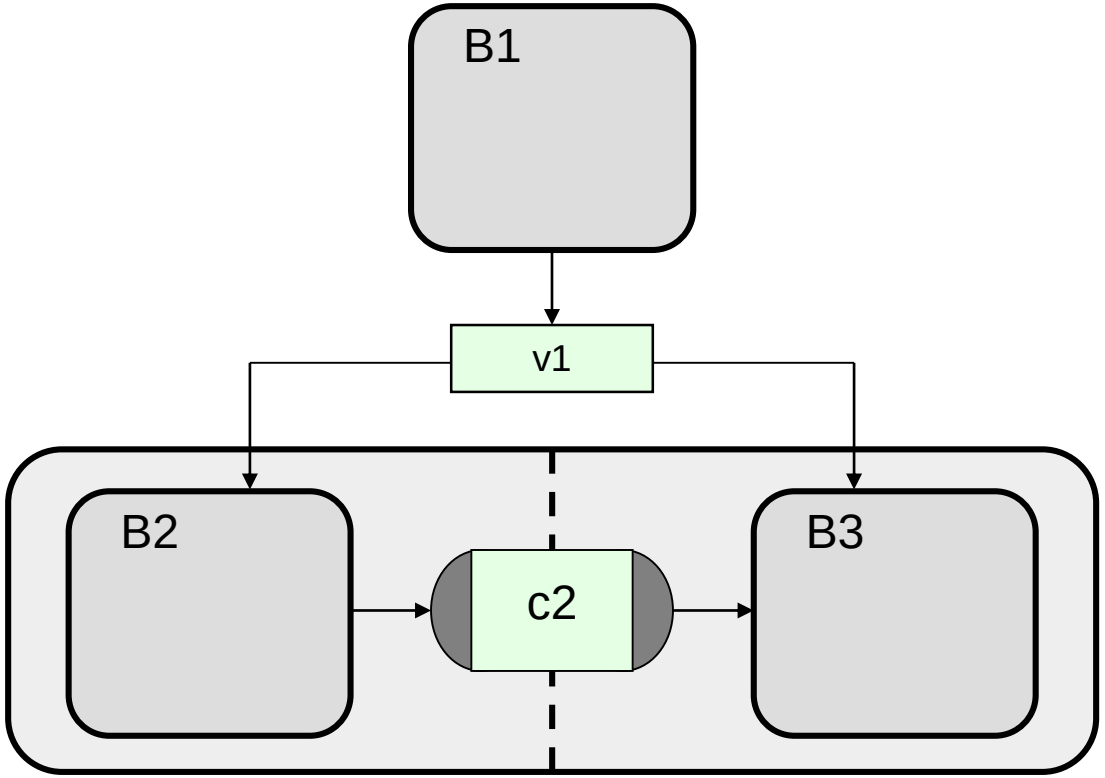
- ✓ **System specification**
  - ✓ Specification modeling
  - ✓ System validation
- **System refinement**
  - Computation
  - Communication
  - Implementation
- **SCE design environment**
  - Modeling
  - Refinement
  - Synthesis

Specification Model

- **High-level, abstract model**
  - Pure system functionality
  - Algorithmic behavior
  - No implementation details
- **No implicit structure / architecture**
  - Behavioral hierarchy
- **Untimed**
  - Executes in zero (logical) time
  - Causal ordering
  - Events only for synchronization



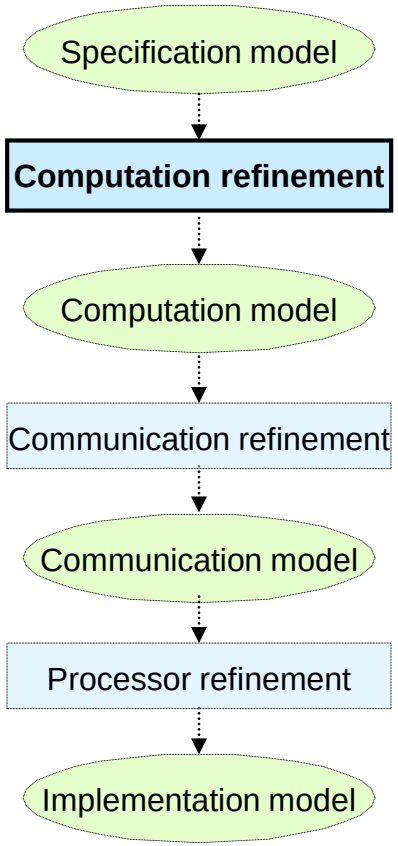
Specification Model Example



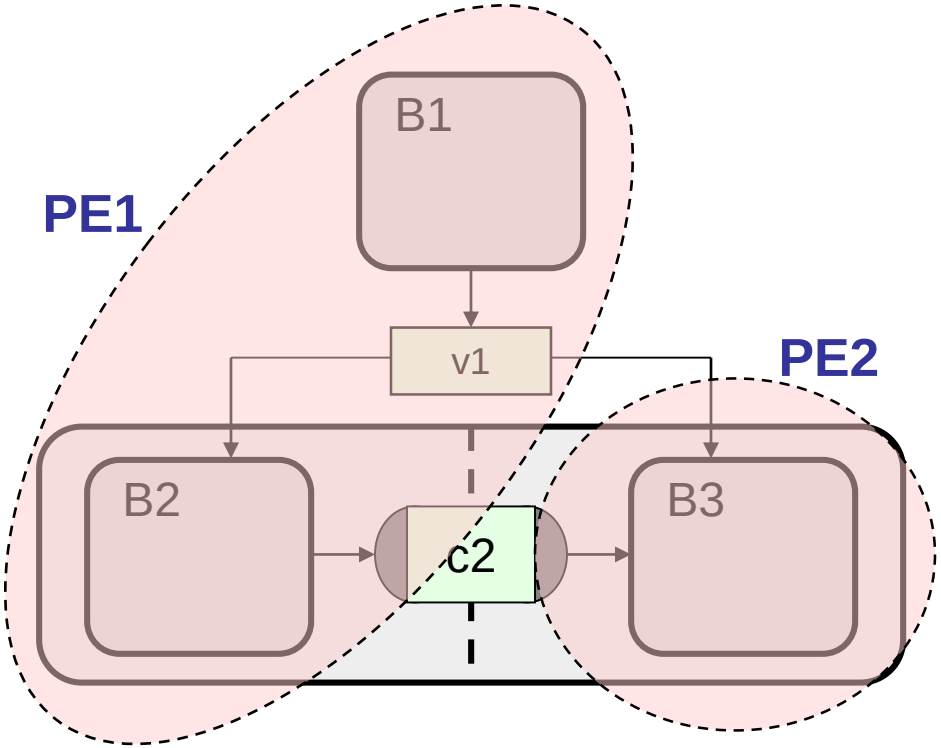
- **Synthesizable specification model**
  - Hierarchical parallel-serial composition
  - Communication through variables and standard channels

Computation Refinement

- **PE allocation / selection**
- **Behavior partitioning**
- **Variable partitioning**
- **Scheduling**



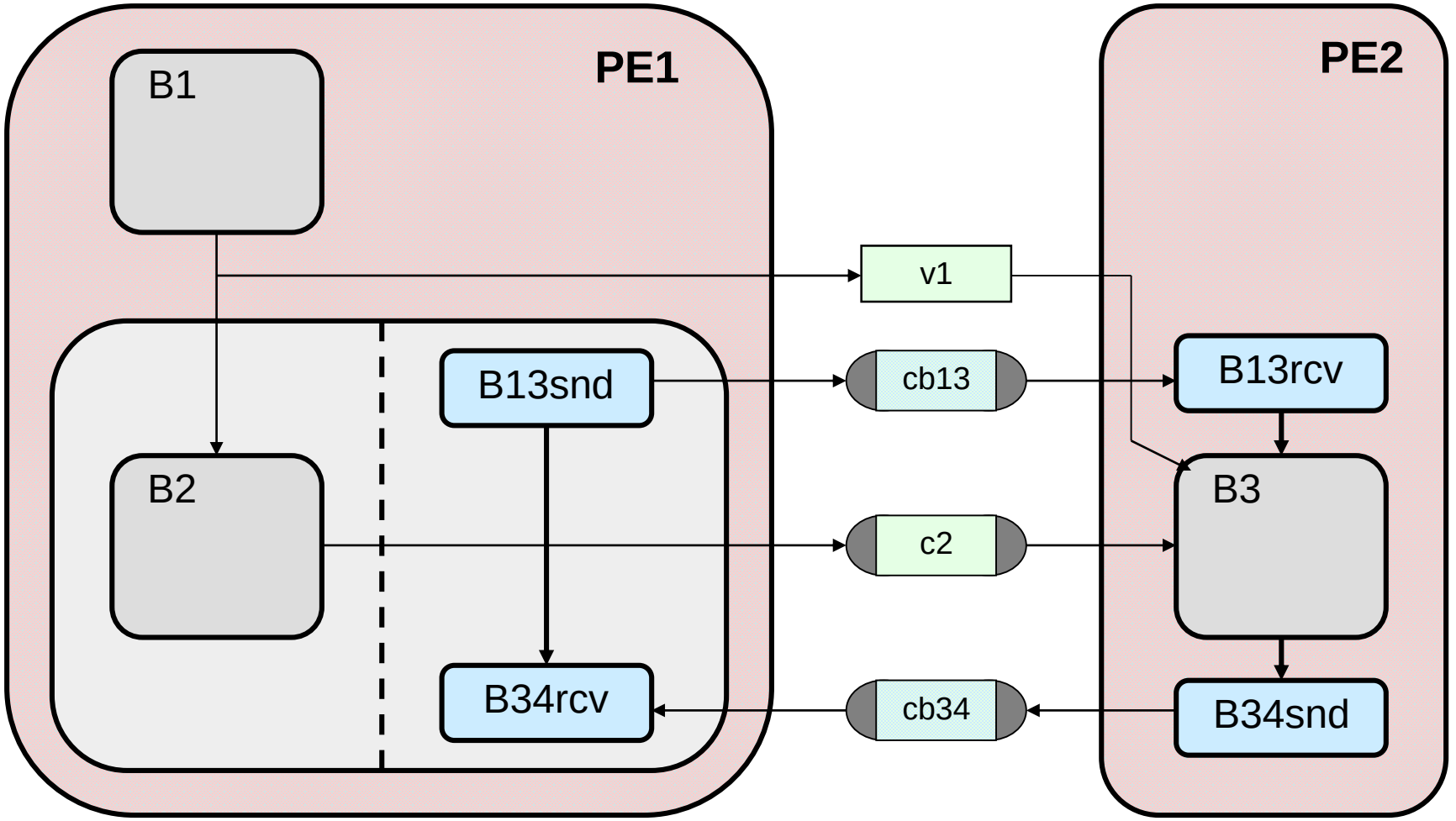
PE Allocation, Behavior Partitioning



- Allocate PEs
- Partition behaviors
- Globalize communication

➢ **Additional level of hierarchy to model PE structure**

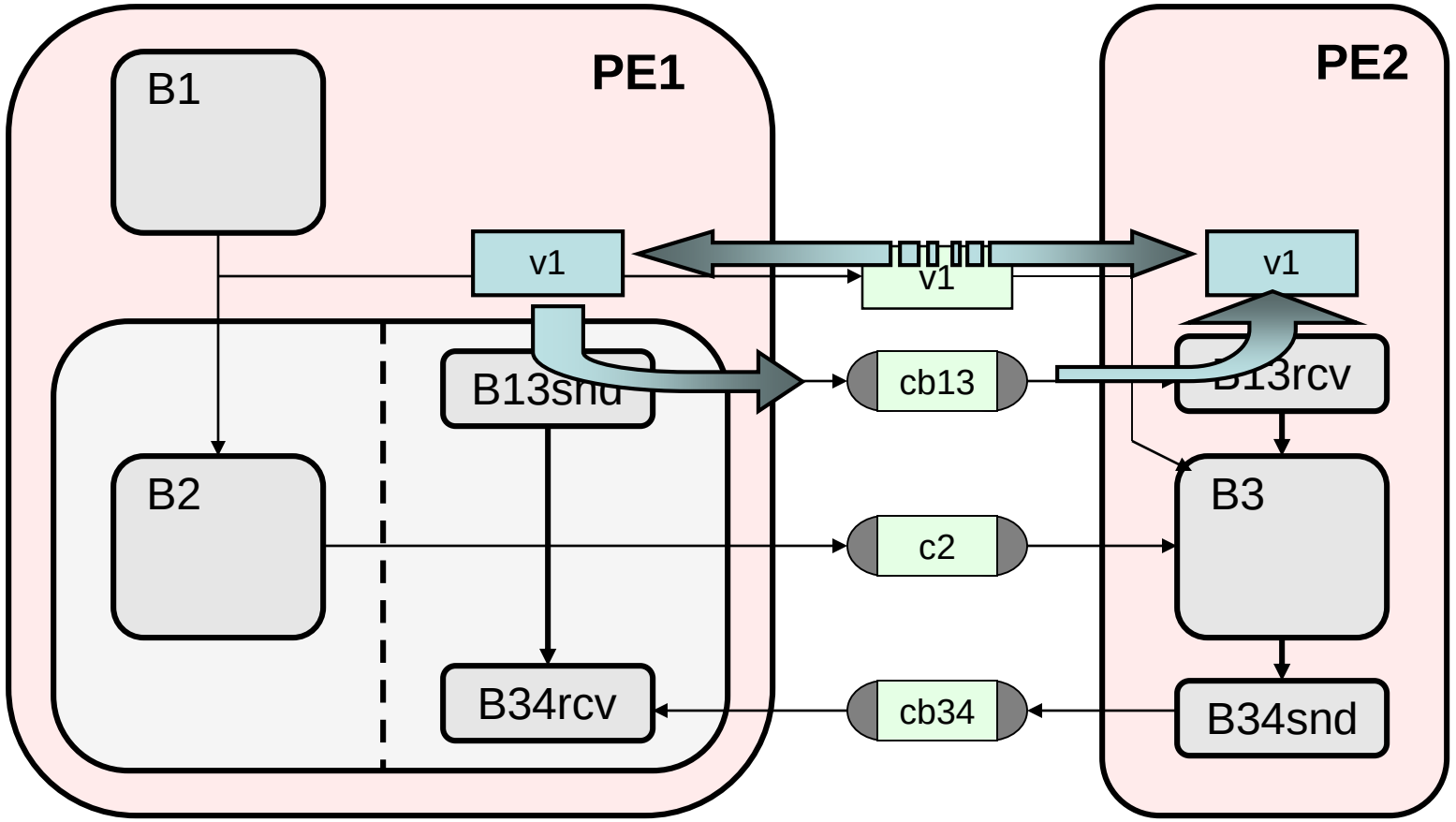
Model after Behavior Partitioning



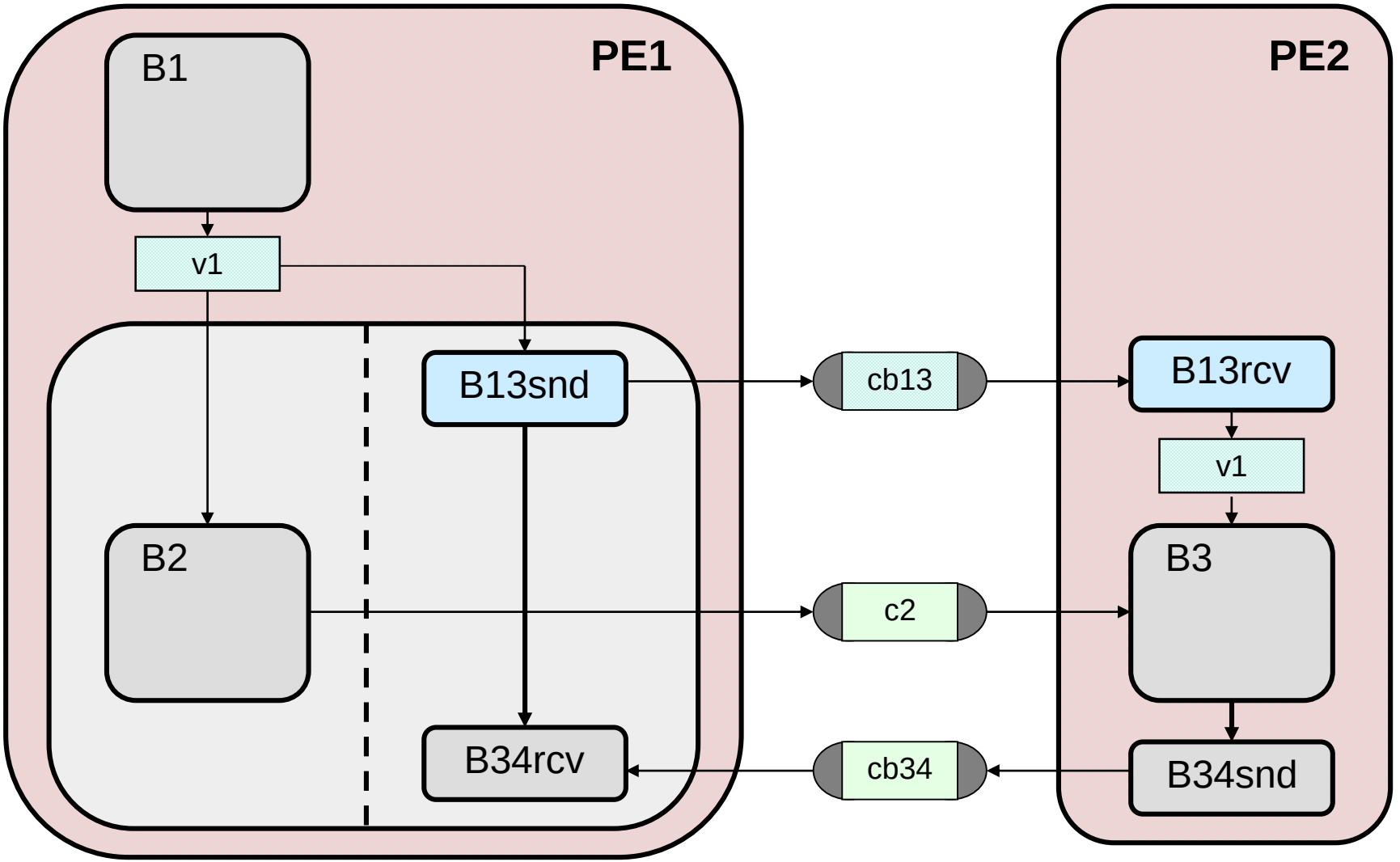
- **Synchronization to preserve execution order/semantics**

Variable Partitioning

- **Shared memory vs. message passing implementation**
  - Map global variables to local memories
  - Communicate data over message-passing channels



Model after Variable Partitioning



- **Keep local variable copies in sync**
  - Communicate updated values at synchronization points
  - Transfer control & data over message-passing channel

Timed Computation

- **Execution time of behaviors**
  - Estimated target delay / timing budget
- **Granularity**
  - Behavior / function / basic-block level

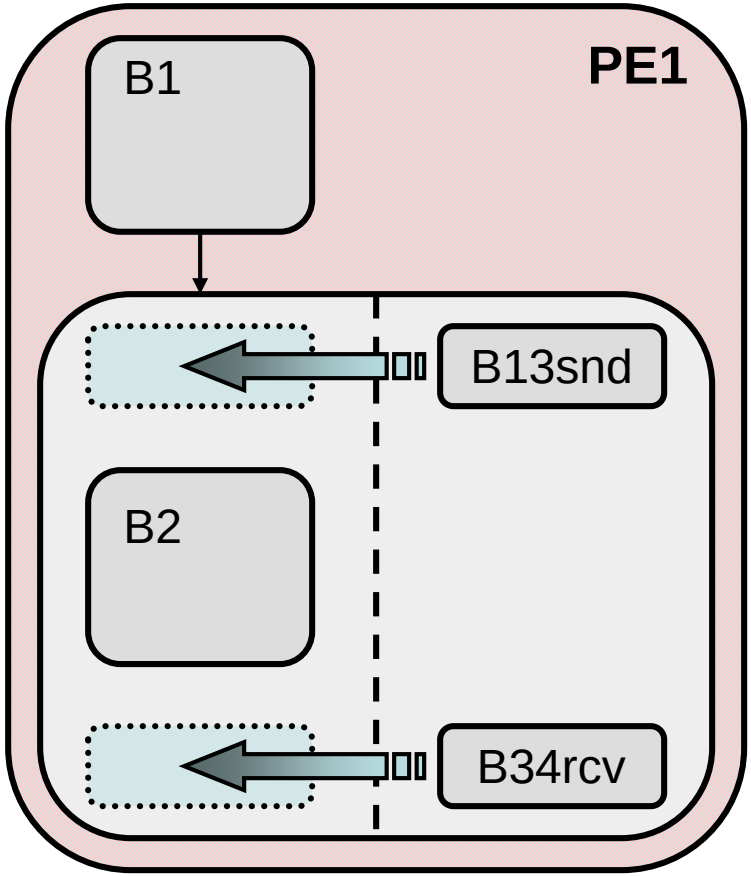
➢ **Annotate behaviors**

- Simulation feedback
- Synthesis constraints

```
1  behavior B2( in int v1, ISend c2 )
   {
     void main(void) {
       ...
5      waitfor( B2_DELAY1 );
       c2.send( ... );
       ...
10     waitfor( B2_DELAY2 );
     }
   };
```

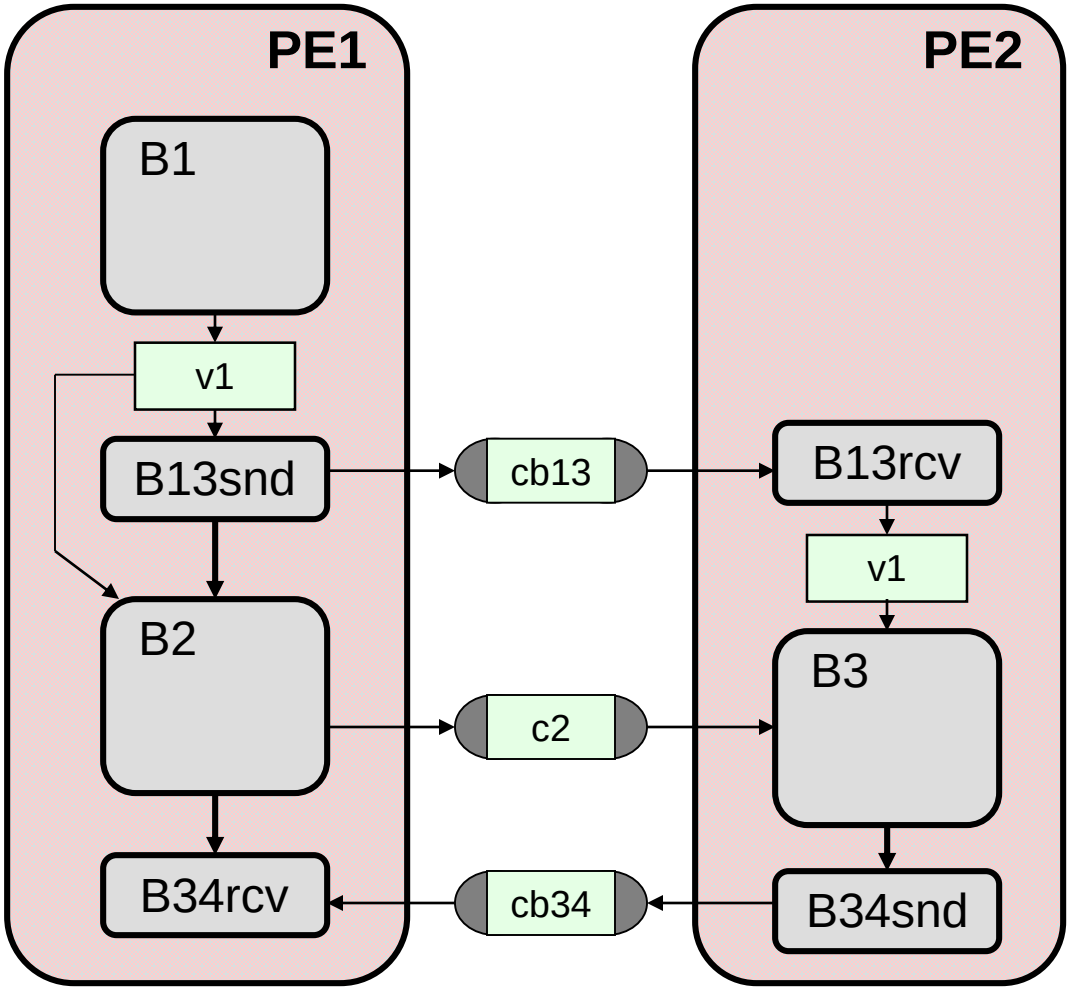
Scheduling

➢ Serialize behavior execution on components



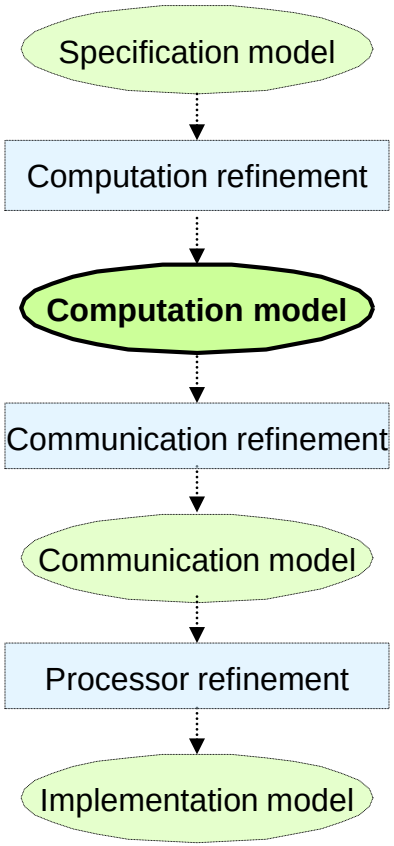
- Static scheduling
  - Fixed behavior execution order
  - Flattened behavior hierarchy
- Dynamic scheduling
  - Pool of tasks
  - Scheduler, abstracted OS

Computation Model Example



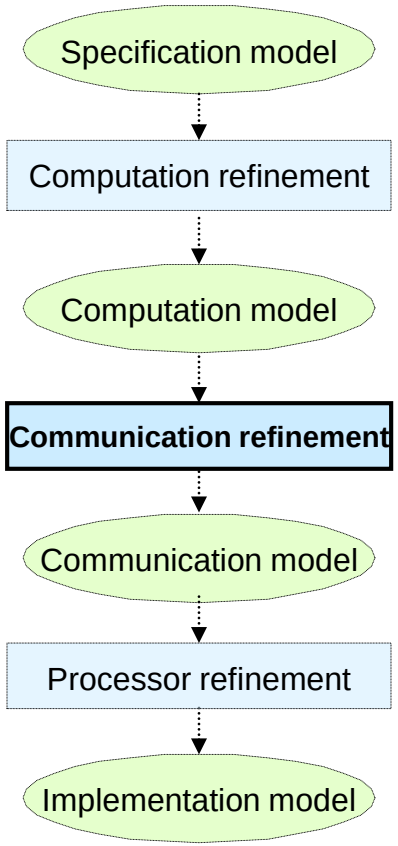
Computation Model

- **Component structure/architecture**
  - Top level of behavior hierarchy
- **Behavioral/functional component view**
  - Behaviors grouped under top-level component behaviors
  - Sequential behavior execution
- **Timed**
  - Estimated execution delays

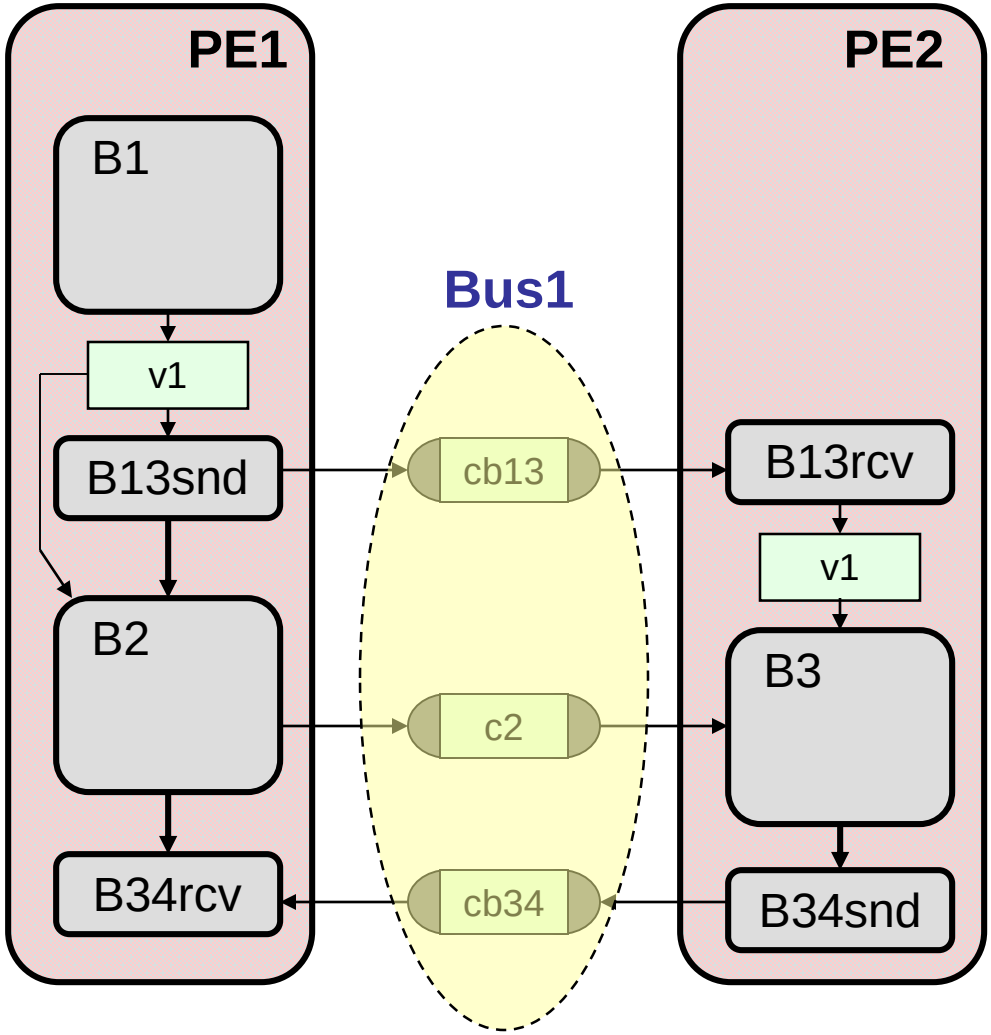


Communication Refinement

- **Network allocation / protocol selection**
- **Channel partitioning**
- **Protocol stack insertion**
- **Inlining**



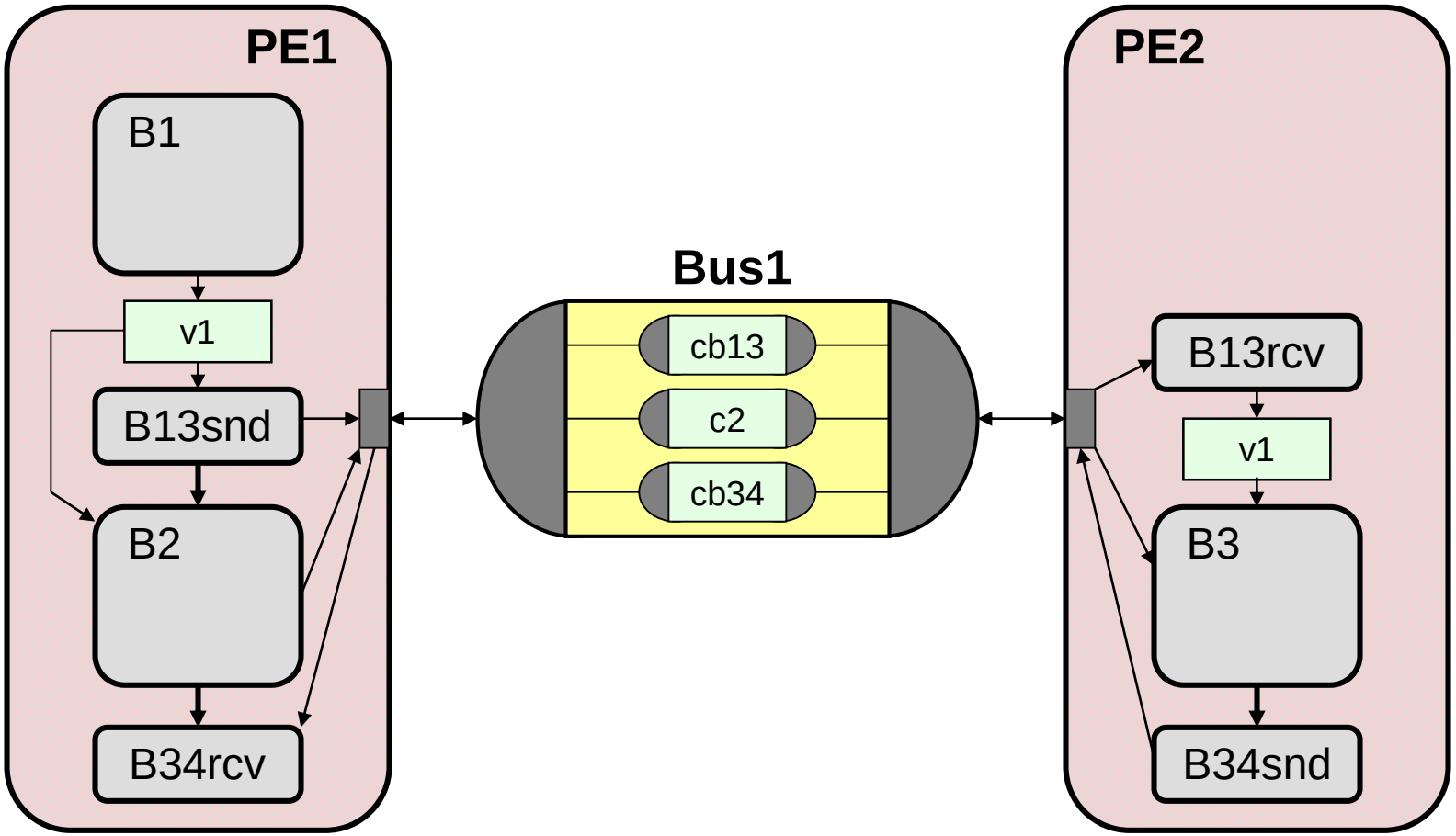
Network Allocation / Channel Partitioning



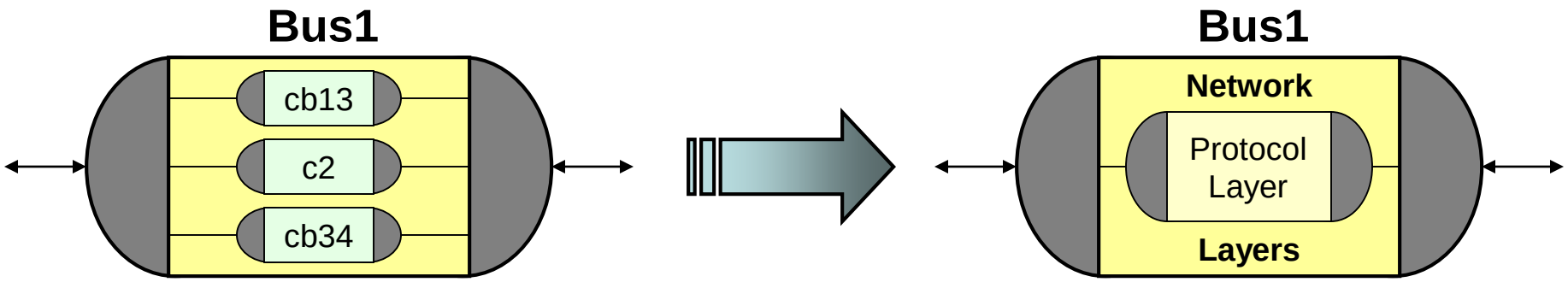
- Allocate busses
- Partition channels
- Update communication

➢ **Additional level of hierarchy to model bus structure**

Model after Channel Partitioning

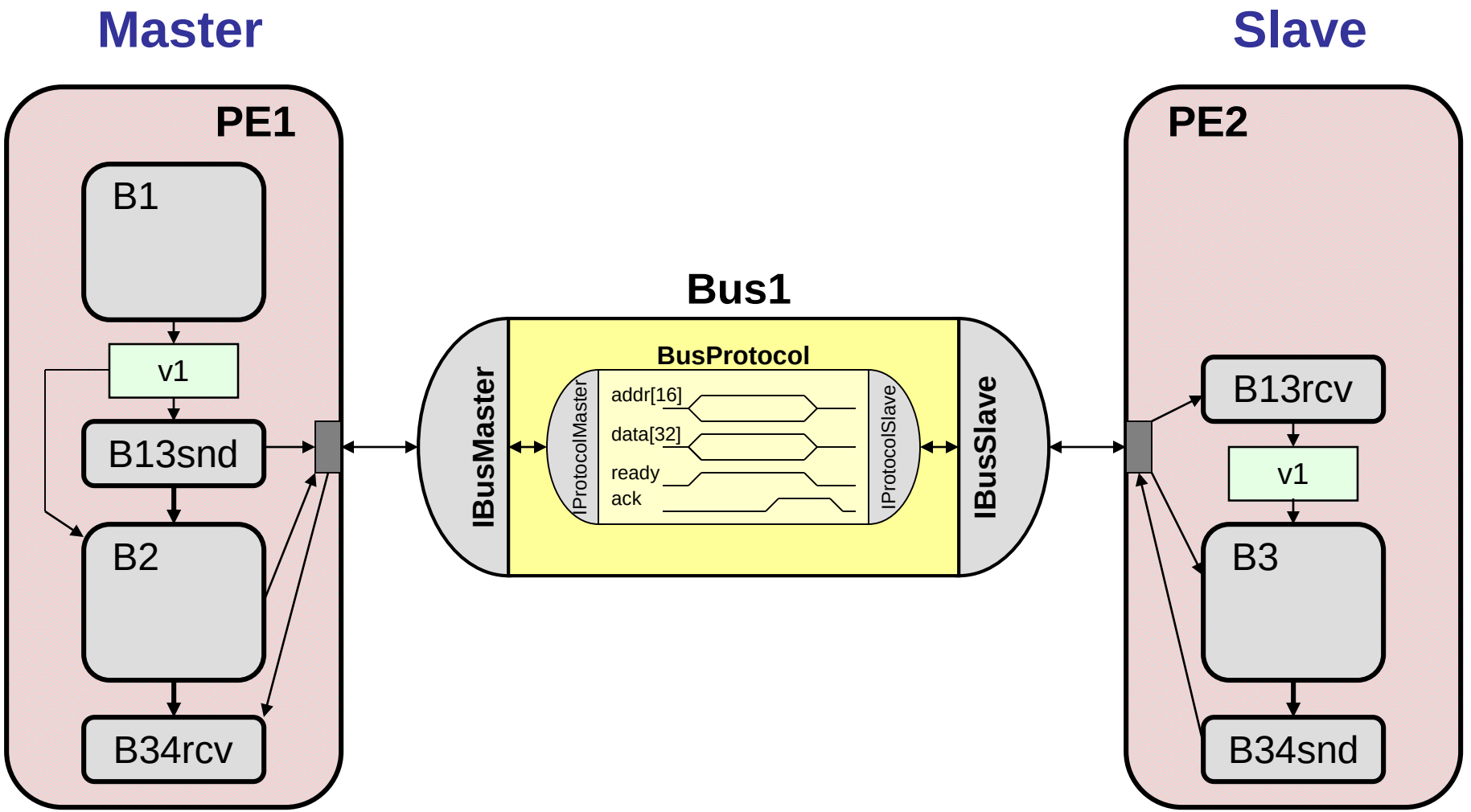


Protocol Insertion

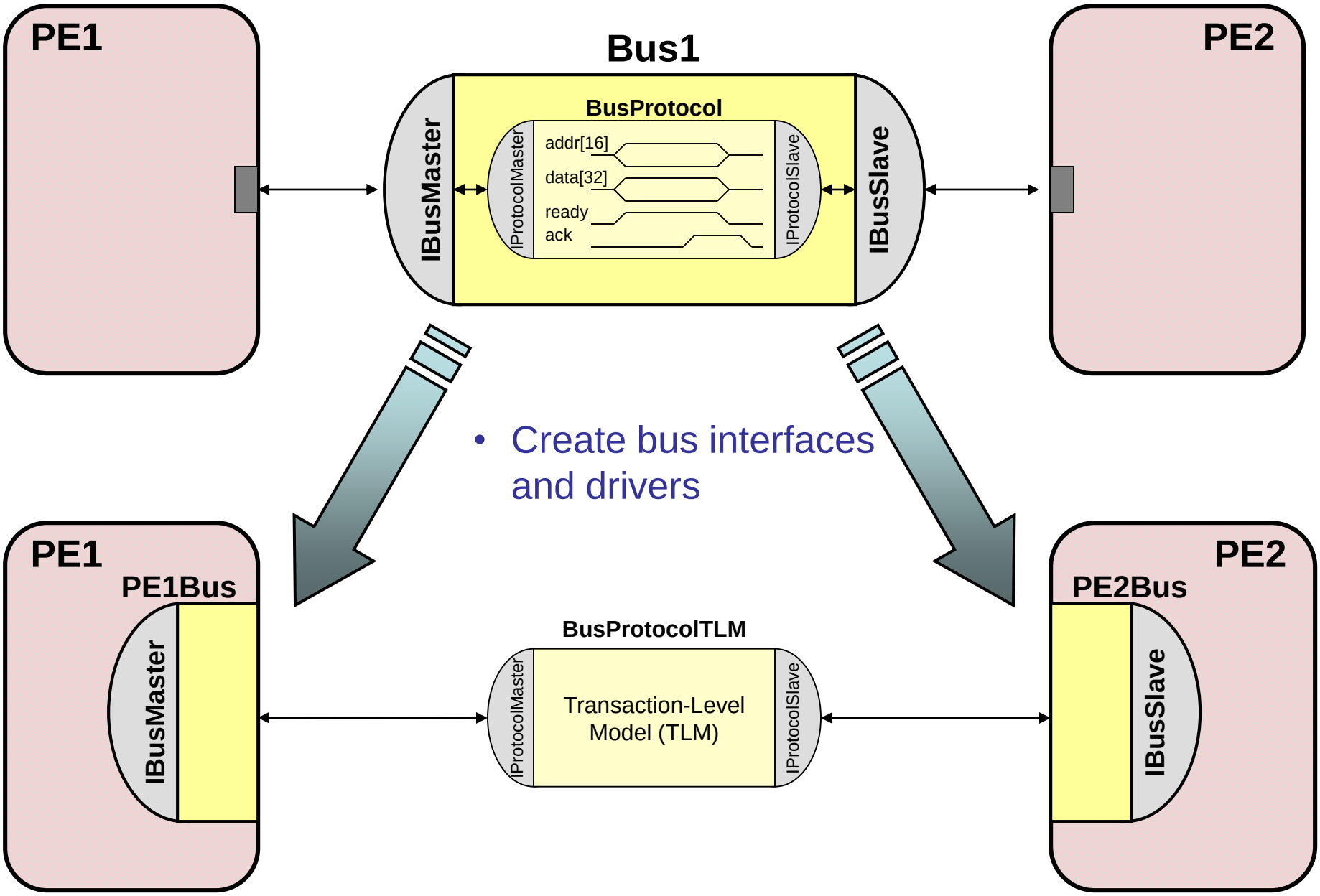


- **Insert protocol layer**
  - Bus protocol channel from database
- **Create network layers**
  - Implement message-passing over bus protocol
- **Replace bus channel**
  - Hierarchical combination of complete protocol stack

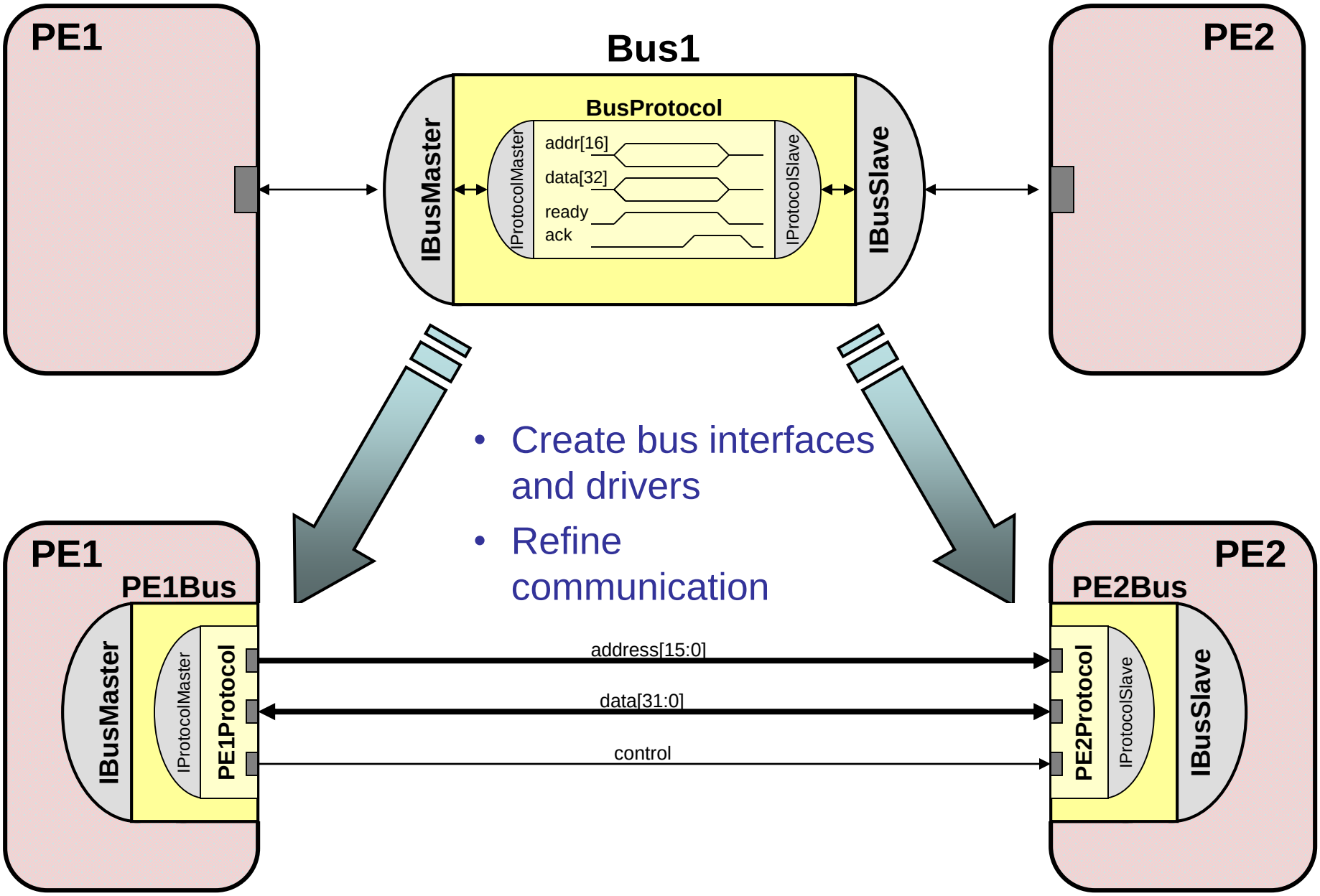
Model after Protocol Insertion



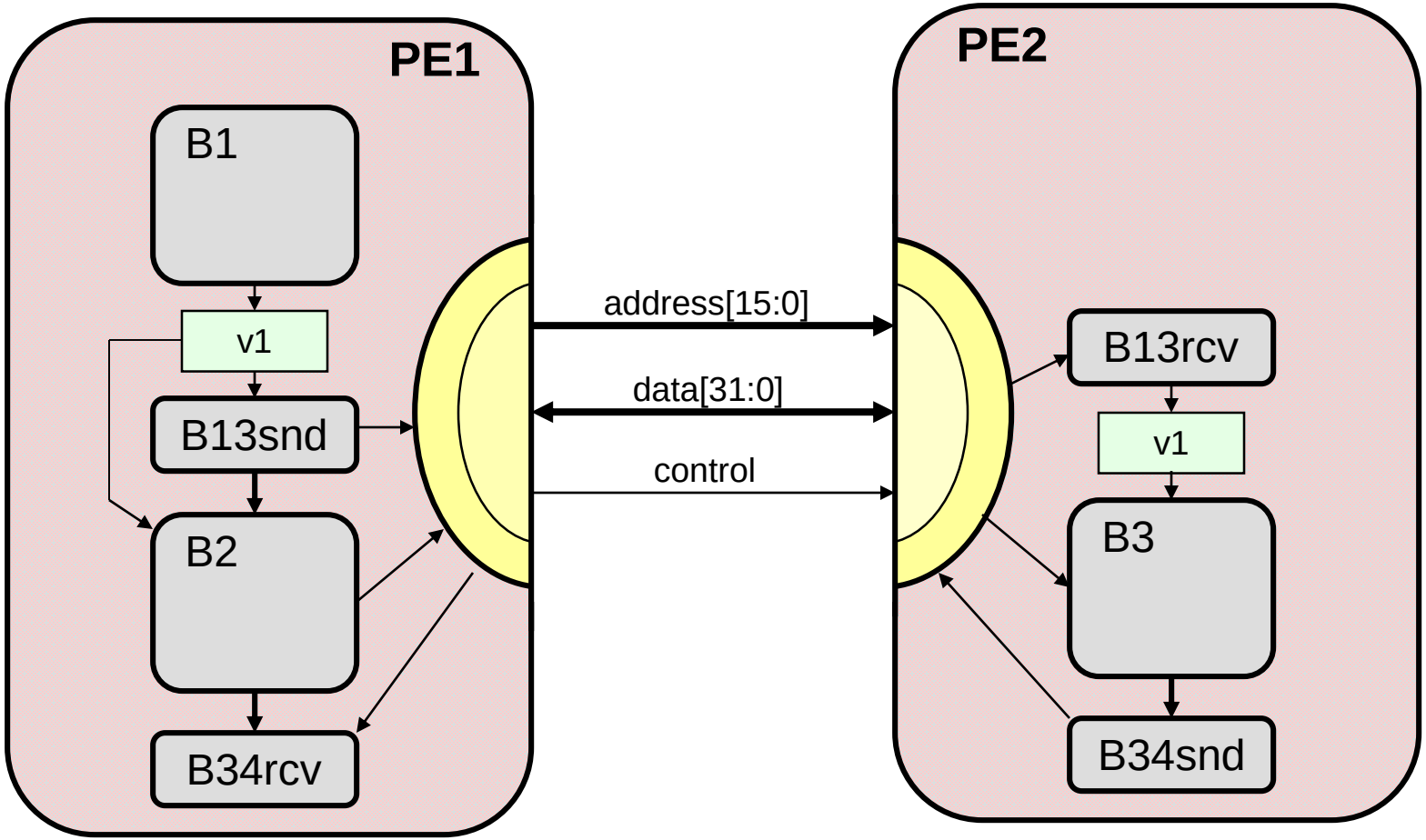
Inlining


PE1
Bus1
PE2
BusProtocol
IBusMaster
IProtocolMaster
addr[16]
data[32]
ready
ack
IProtocolSlave
IBusSlave
• Create bus interfaces and drivers
PE1
PE1Bus
IBusMaster
BusProtocolTLM
IProtocolMaster
Transaction-Level Model (TLM)
IProtocolSlave
PE2
PE2Bus
IBusSlave

Inlining

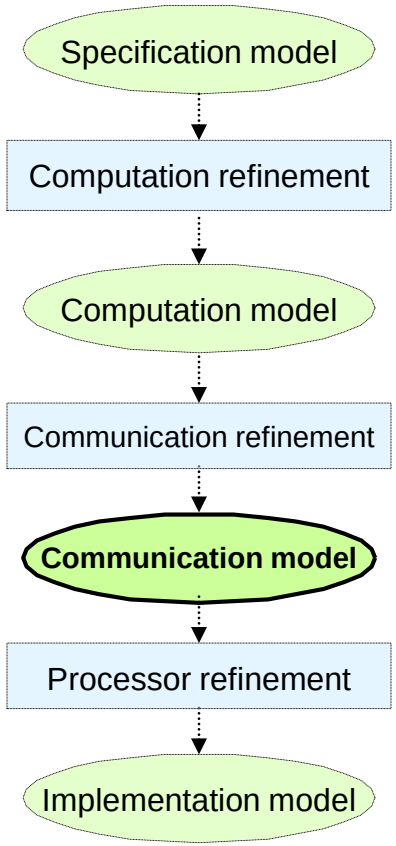

PE1
Bus1
PE2
BusProtocol
IBusMaster
IProtocolMaster
addr[16]
data[32]
ready
ack
IProtocolSlave
IBusSlave
• Create bus interfaces and drivers
• Refine communication
PE1
PE1Bus
IBusMaster
IProtocolMaster
PE1Protocol
address[15:0]
data[31:0]
control
PE2
PE2Bus
PE2Protocol
IProtocolSlave
IBusSlave

Communication Model Example



Communication Model

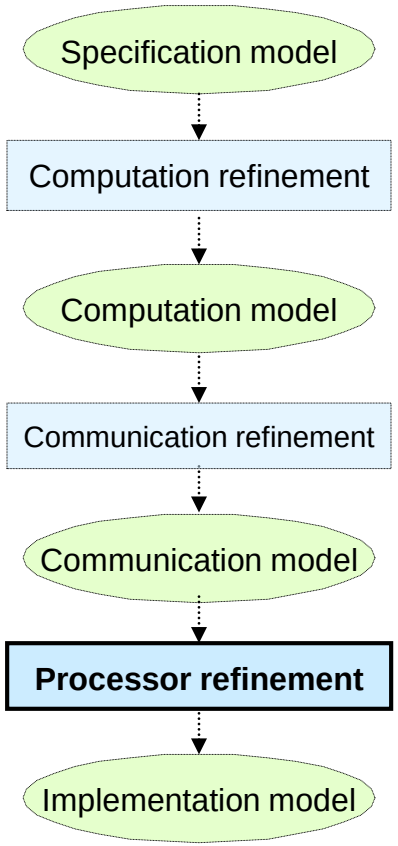
- **Component & bus structure/architecture**
  - Top level of hierarchy
- **Bus-functional component models**
  - Timing-accurate bus protocols
  - Behavioral component description
- **Timed**
  - Estimated component delays
  - Timing-accurate communication



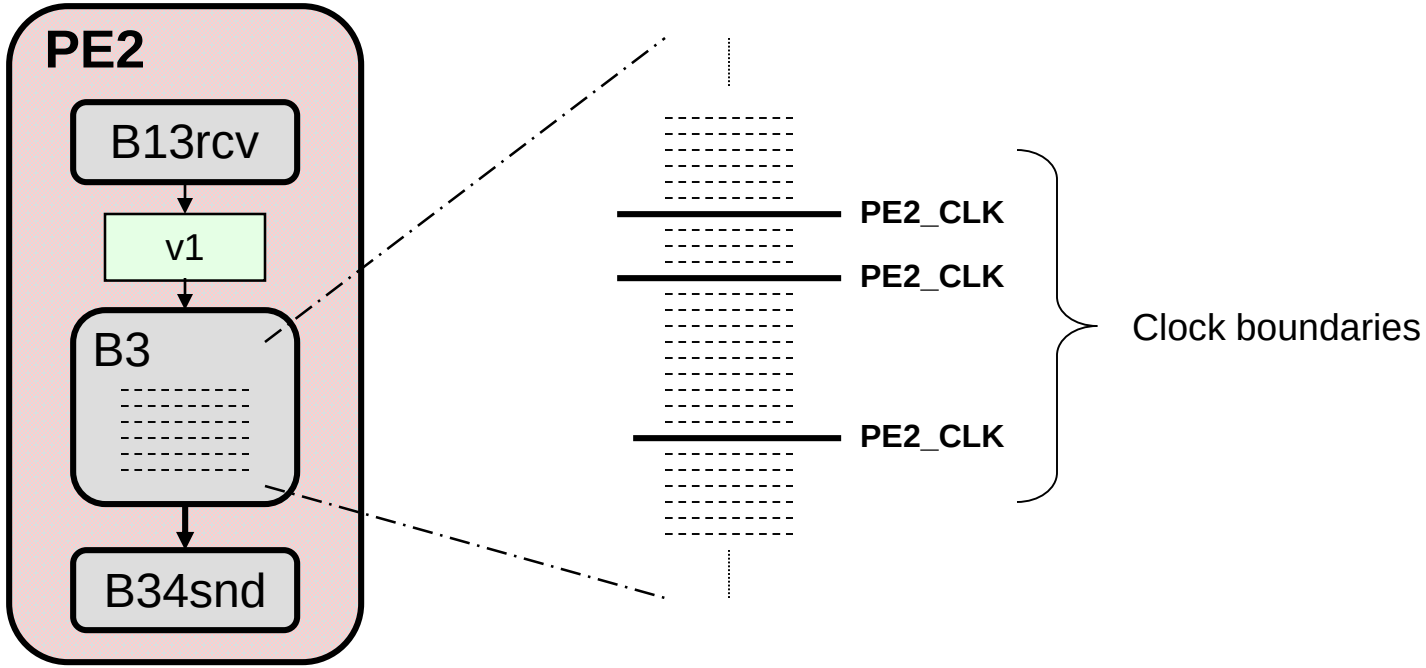
- ➢ **Transaction-level model (TLM)**
- ➢ **Pin-accurate model (PAM)**
  - ➢ Bus cycle-accurate model (BCAM)

Processor Refinement

- **Cycle-accurate implementation of PEs**
  - Hardware synthesis down to RTL
  - Software synthesis down to IS
  - Interface synthesis down to RTL/IS

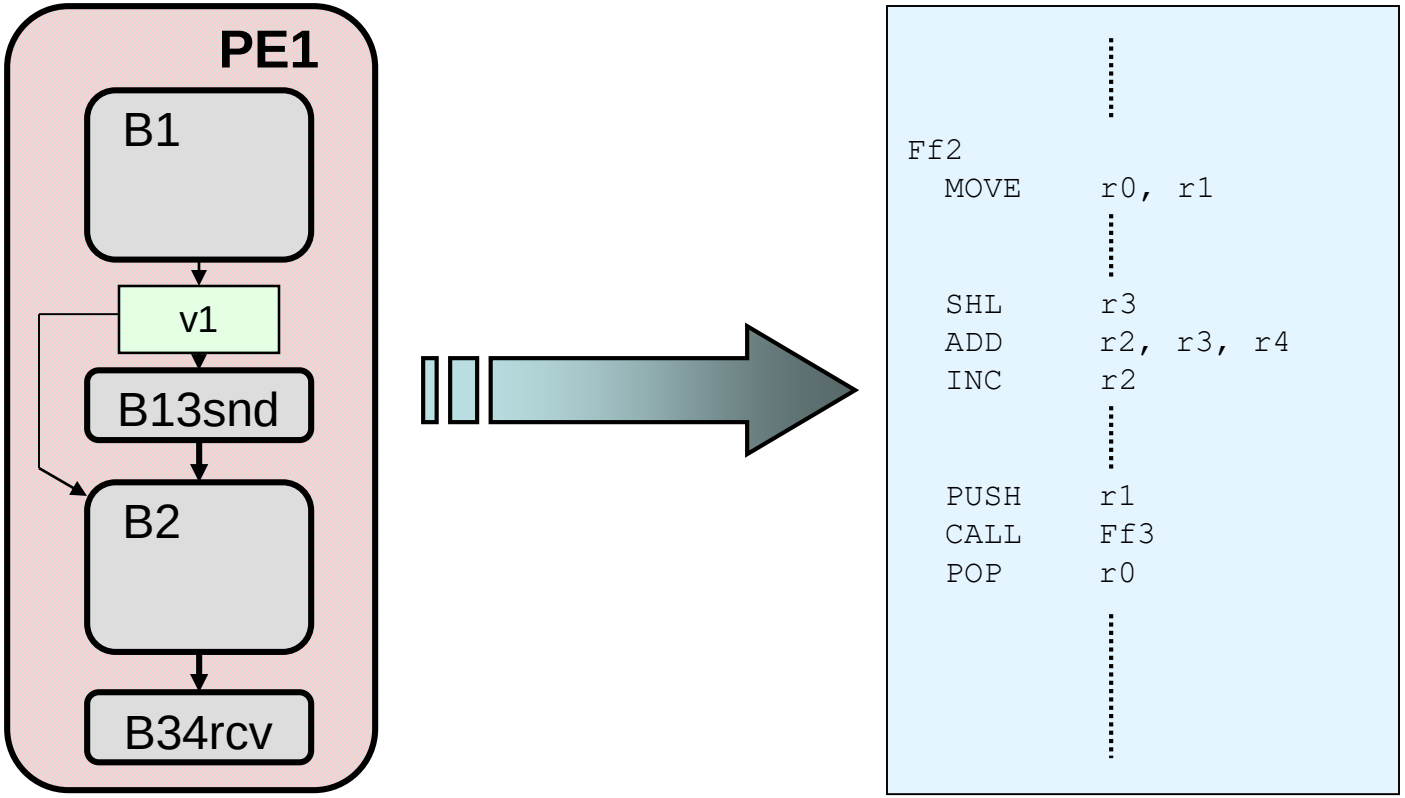


Hardware Synthesis



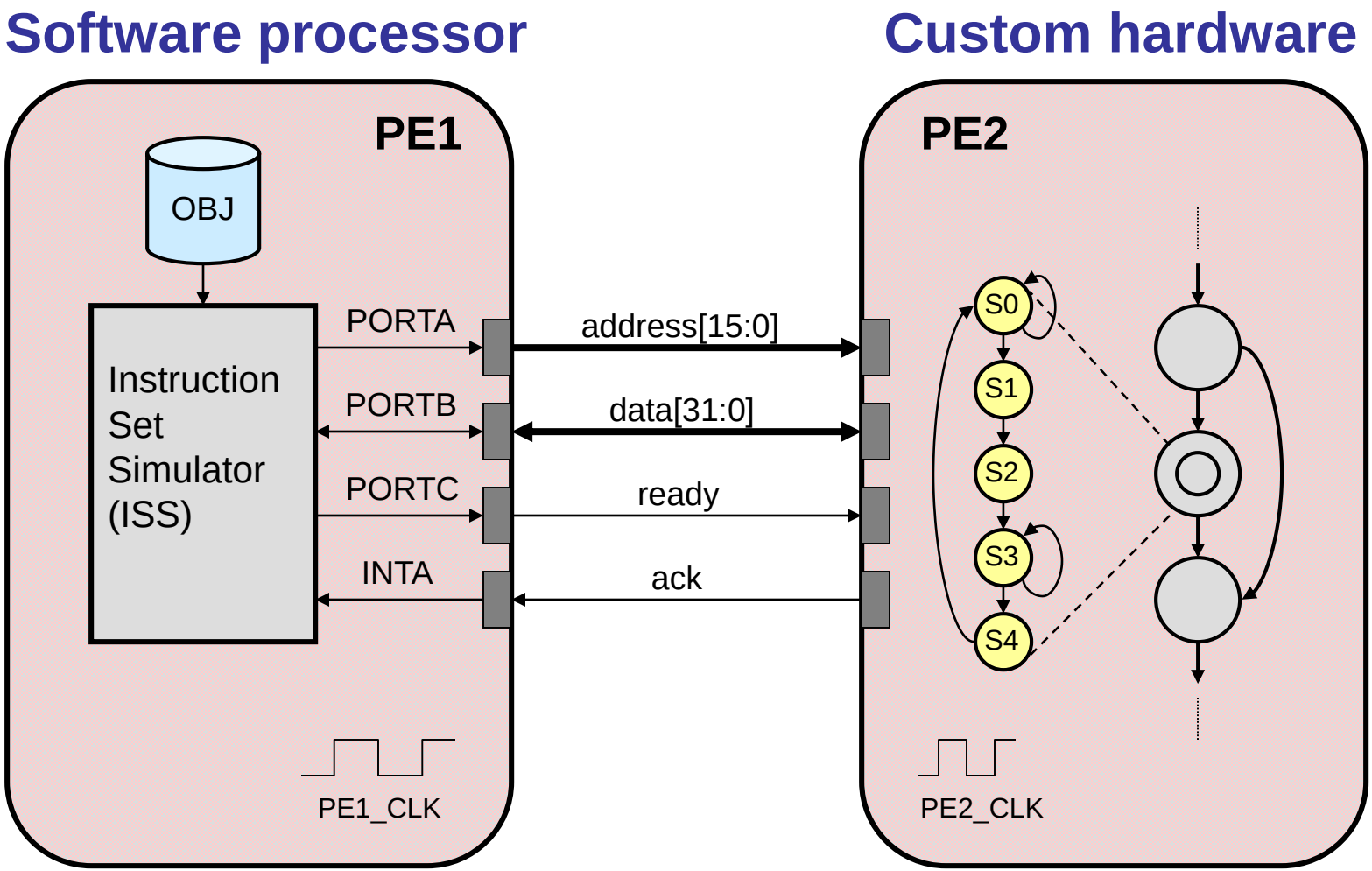
- **Schedule operations into clock cycles**
  - Define clock boundaries in leaf behavior C code
  - Create FSMD model from scheduled C code
    - Controller + datapath

Software Synthesis



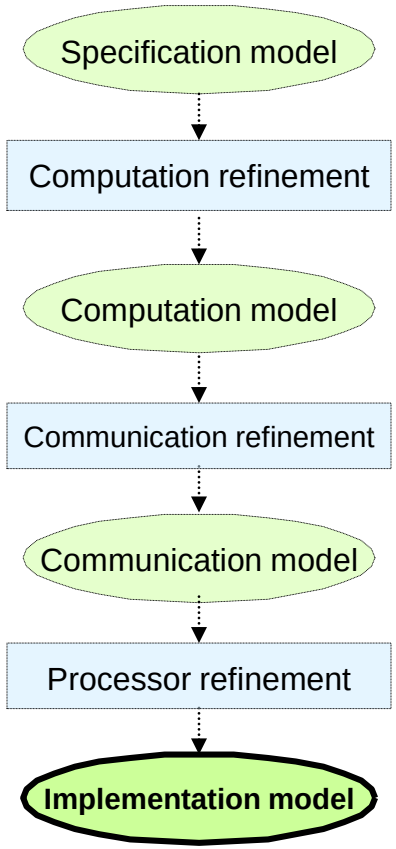
- **Implement behavior on processor instruction-set**
  - Code generation
  - Compilation

Implementation Model



Implementation Model

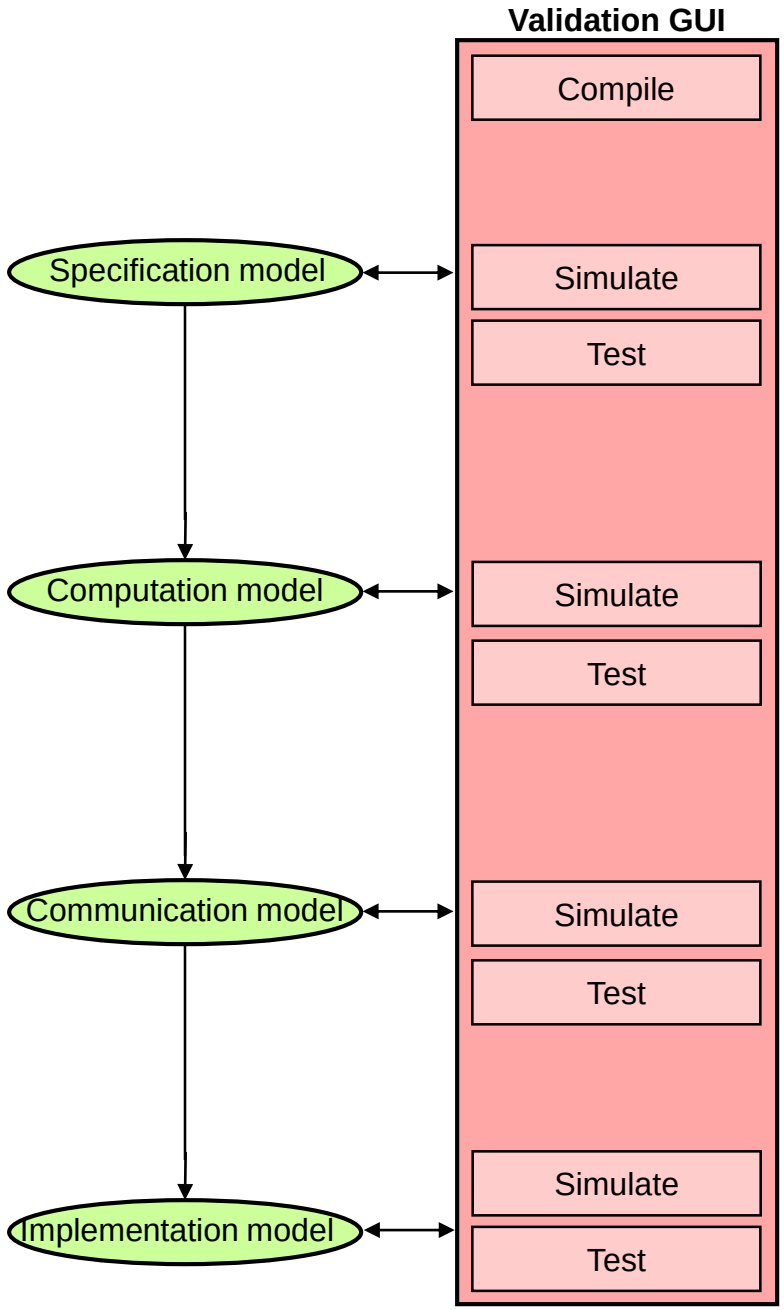
- **Cycle-accurate system description**
  - RTL description of hardware
    - Behavioral/structural FSMD view
  - Object code for processors
    - Instruction-set co-simulation
  - Clocked bus communication
    - Bus interface timing based on PE clock



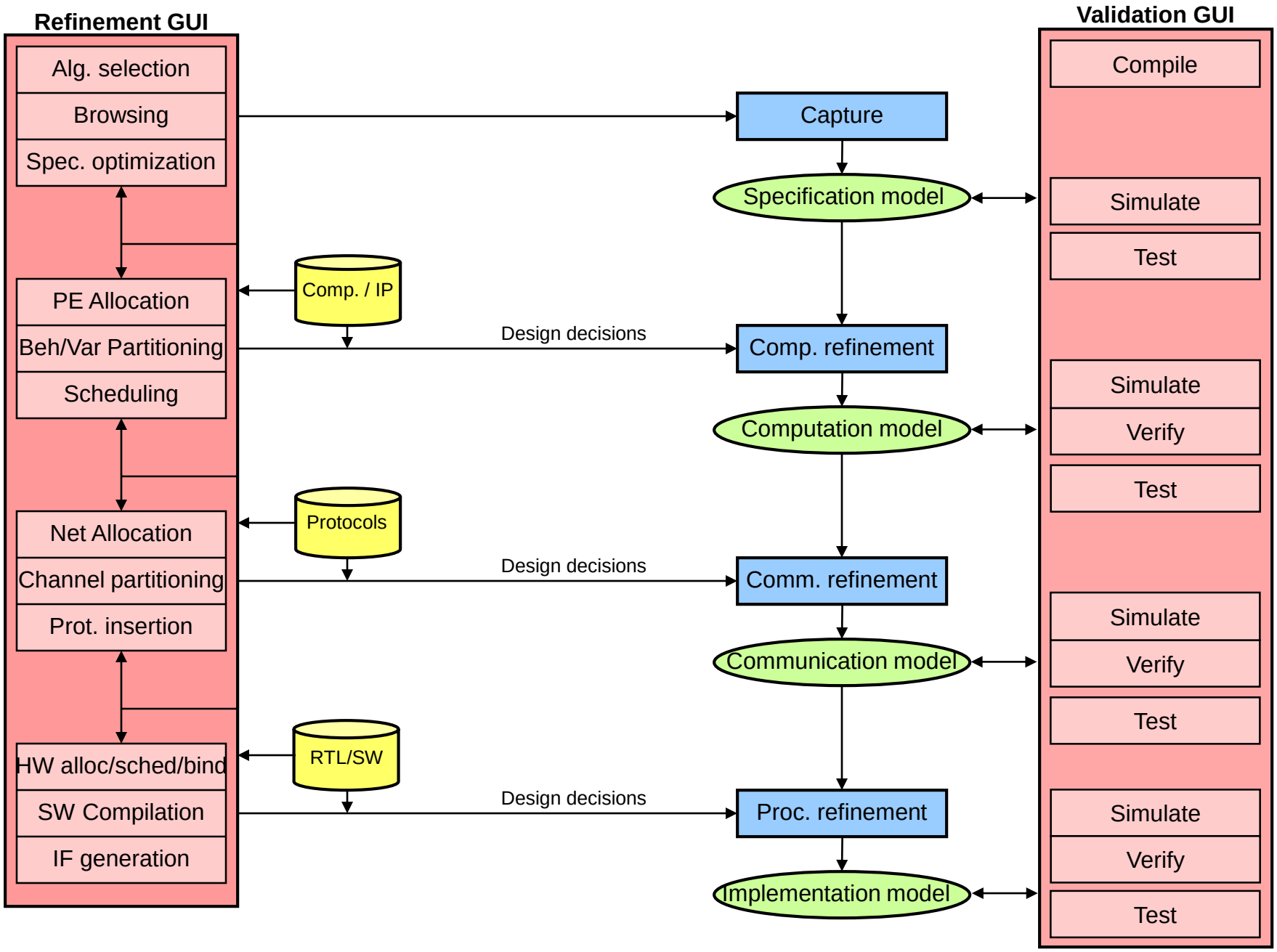
Lecture 5: Outline

- ✓ **System specification**
  - ✓ Specification modeling
  - ✓ System validation
- ✓ **System refinement**
  - ✓ Computation
  - ✓ Communication
  - ✓ Implementation
- **SCE design environment**
  - Modeling
  - Refinement
  - Synthesis

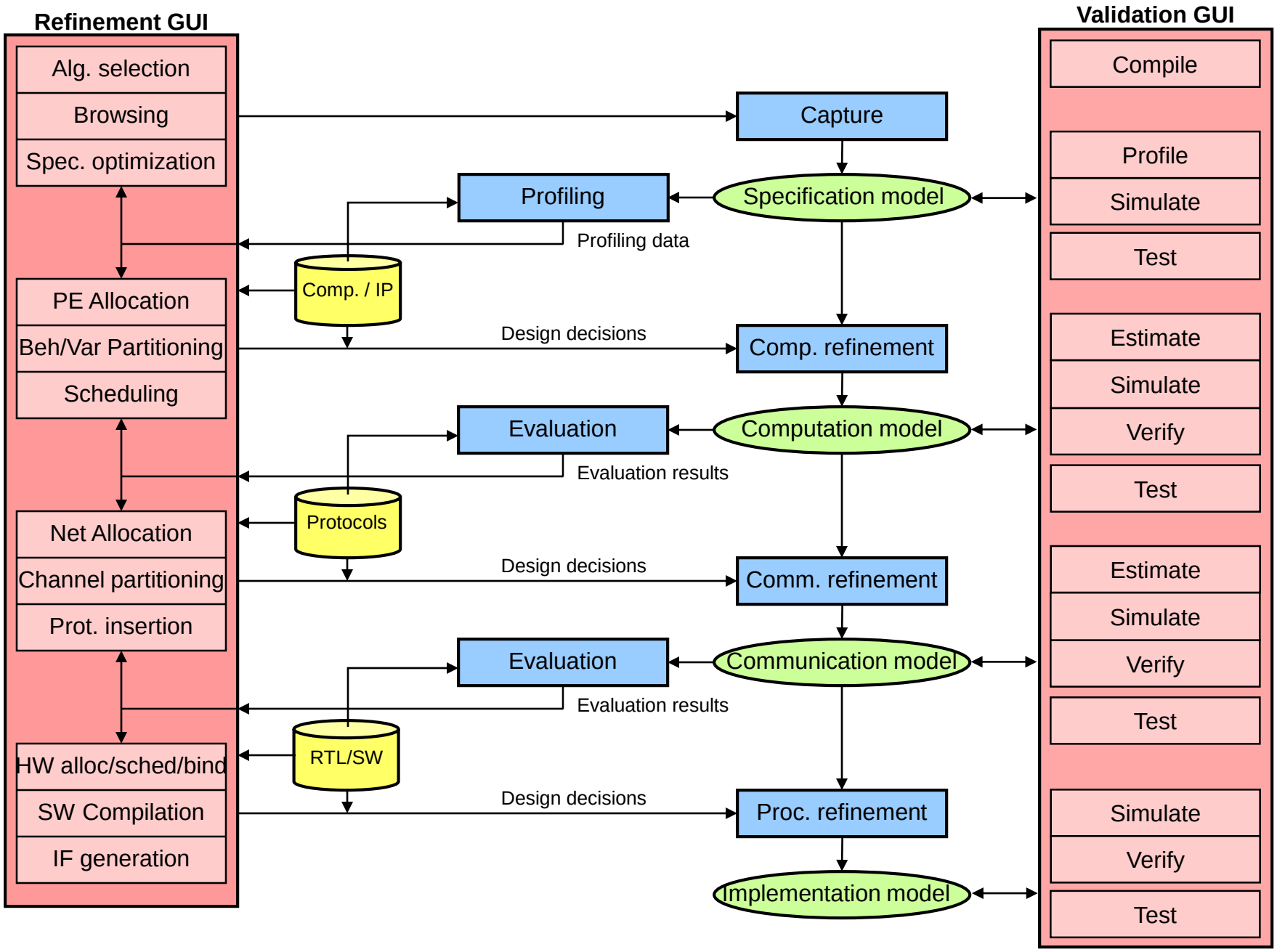
Design Environment (1): Modeling



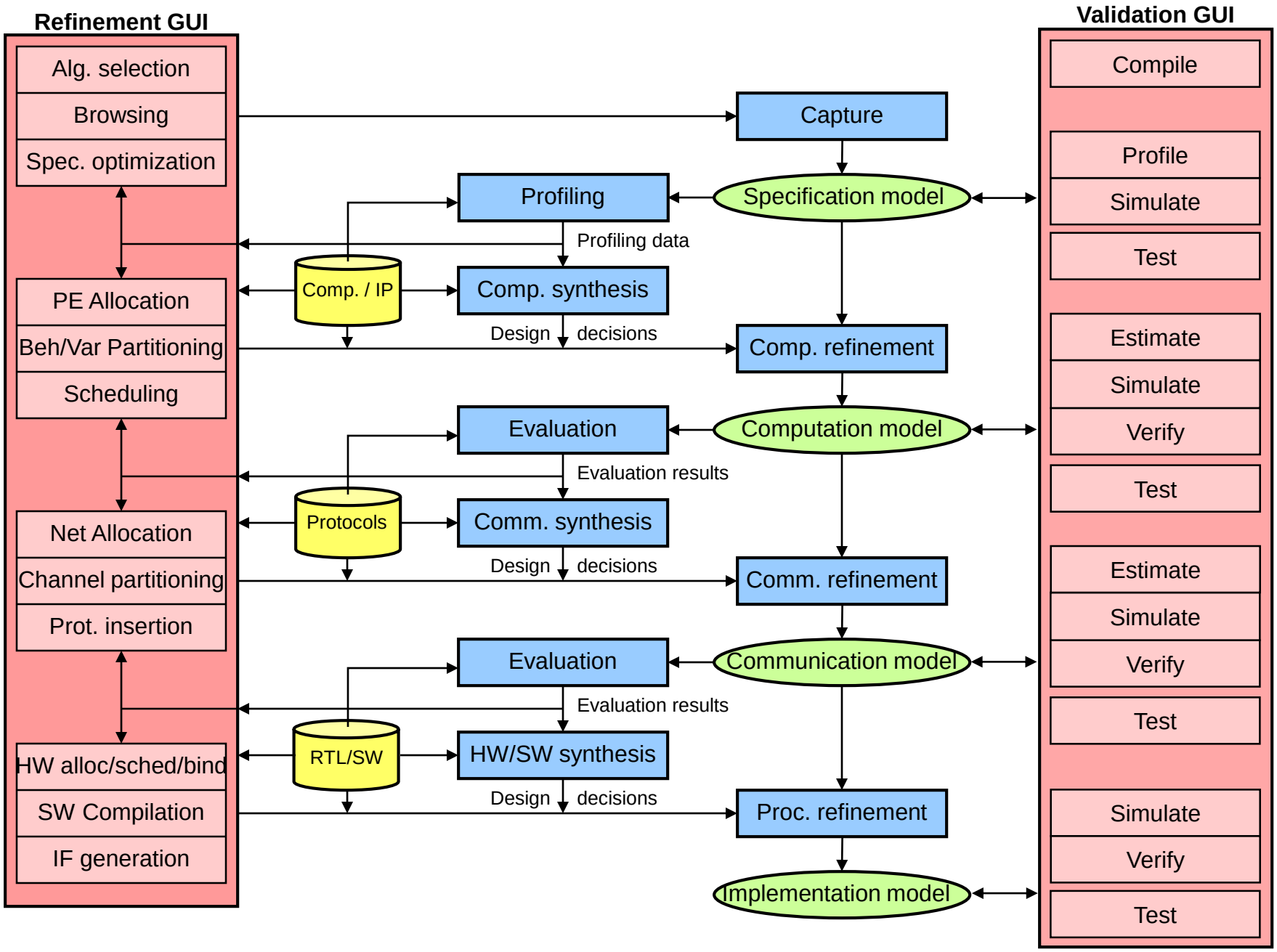
Design Environment (2): Refinement



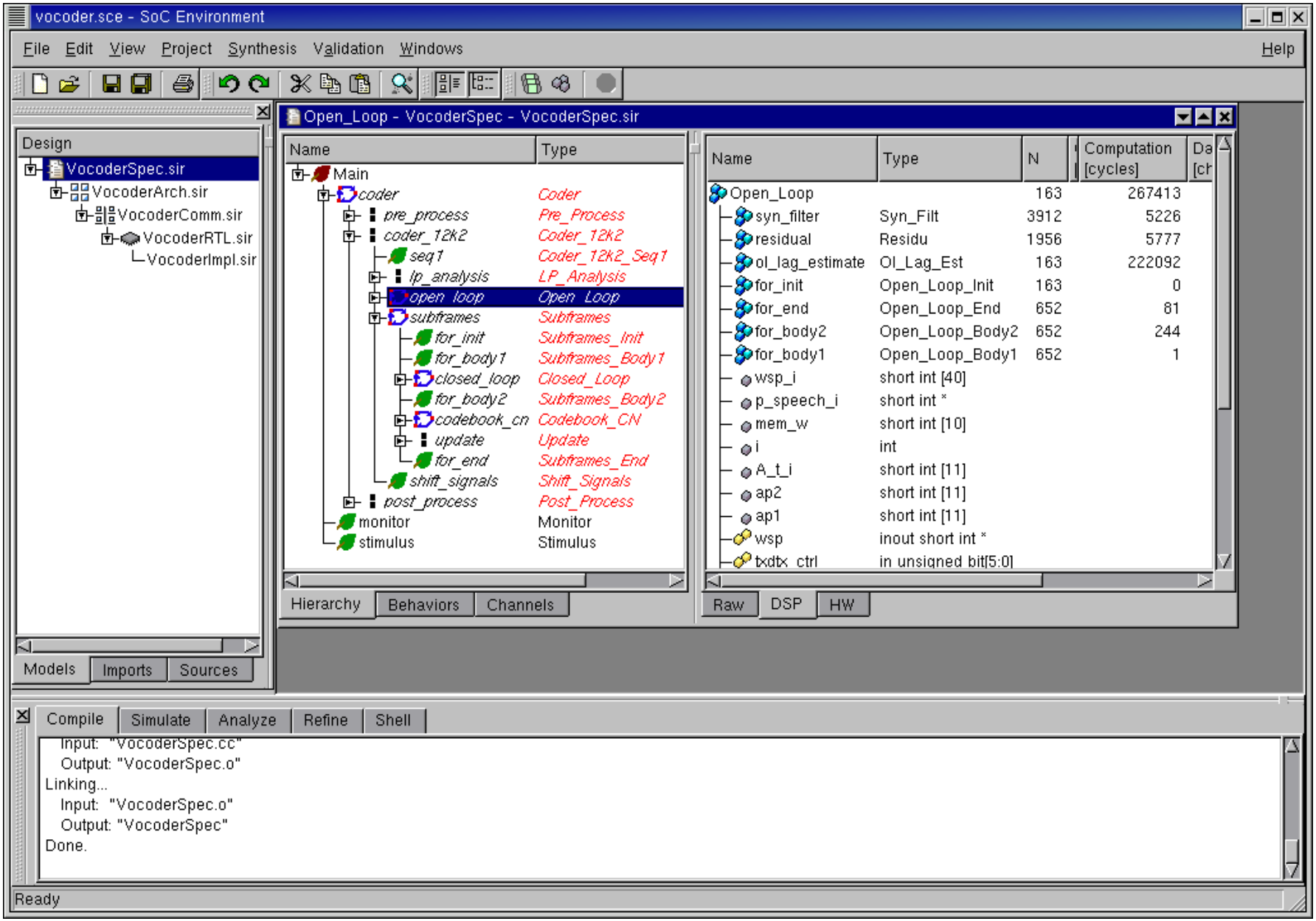
Design Environment (3): Exploration



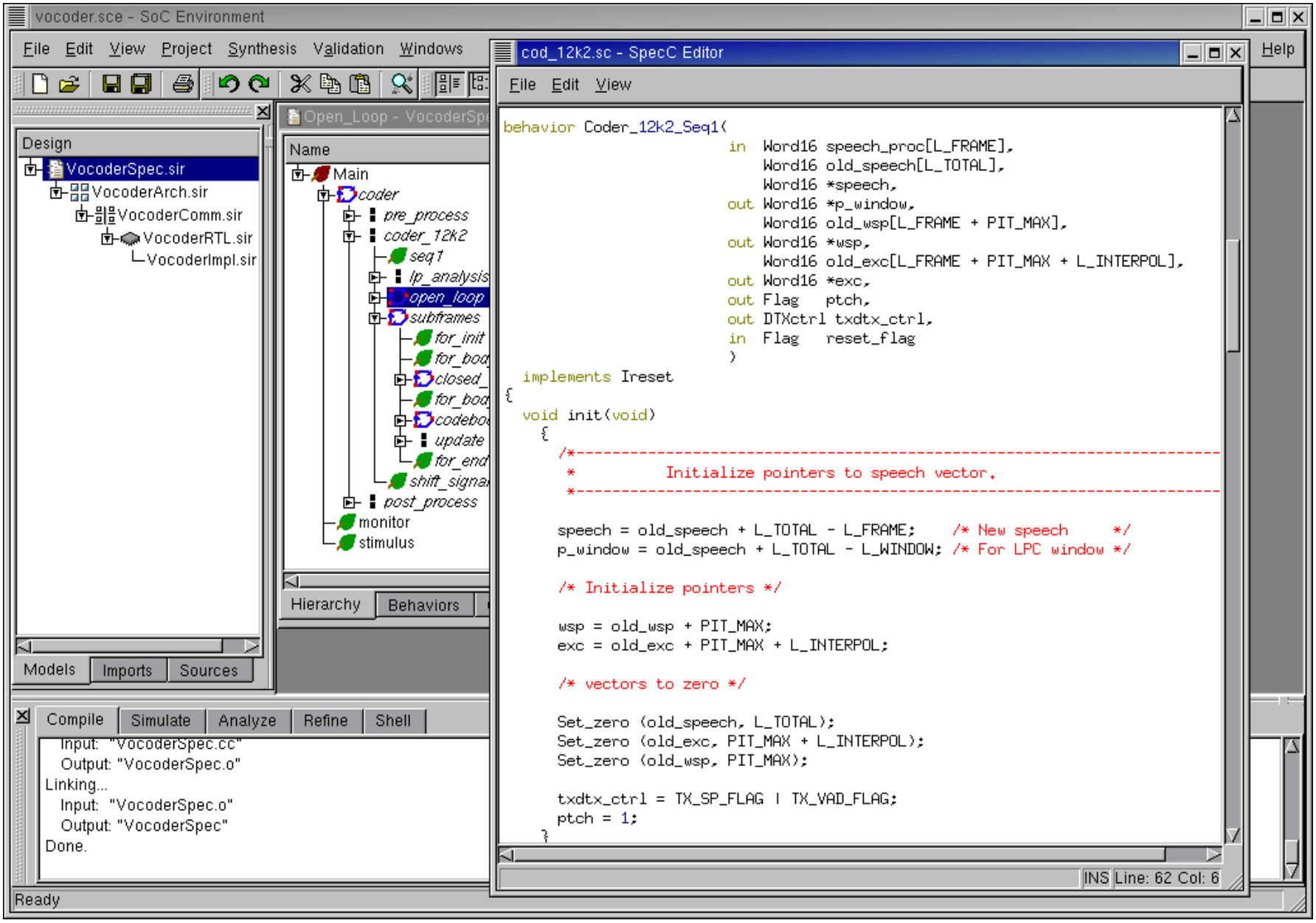
Design Environment (4): Synthesis



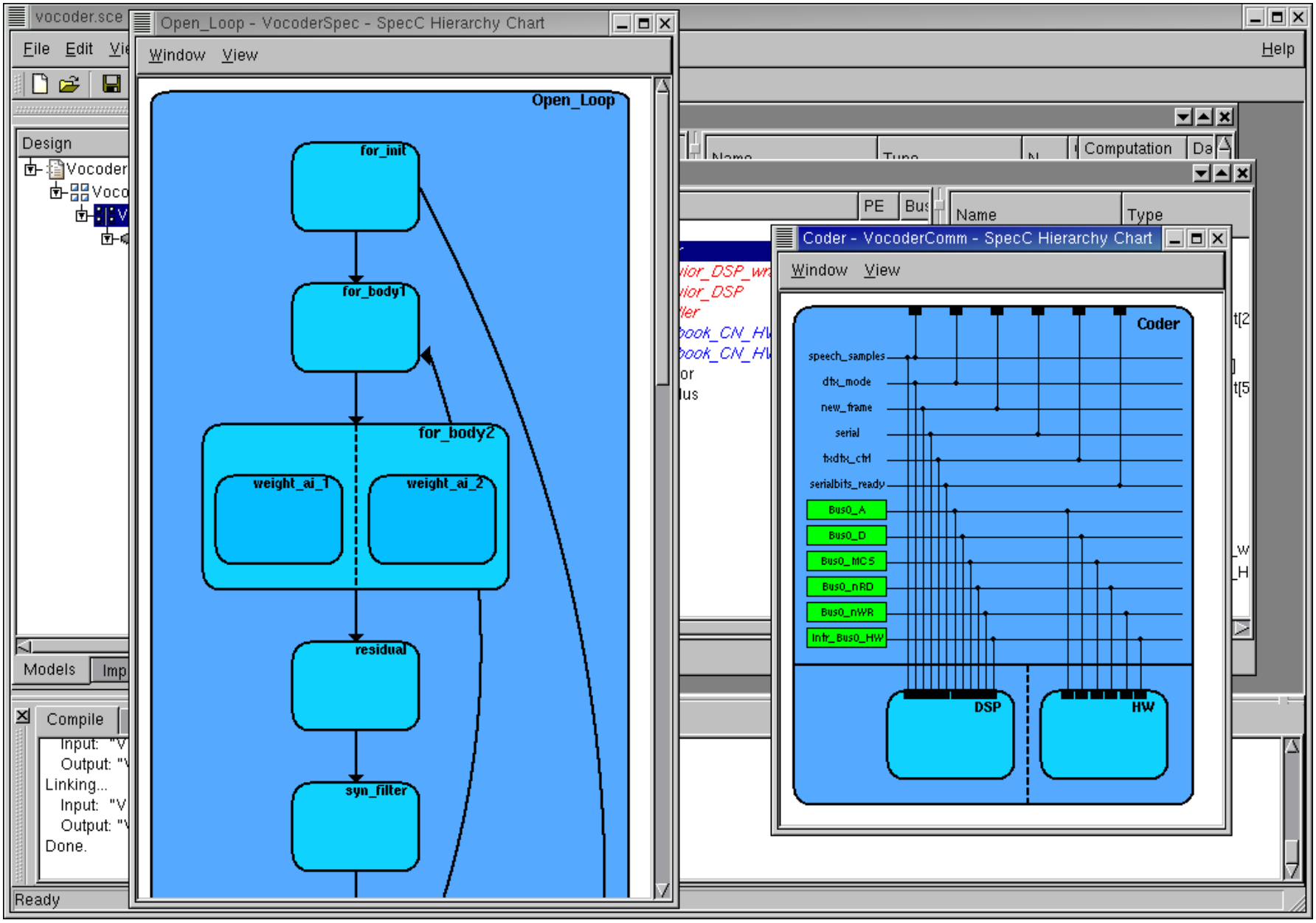
System-on-Chip Environment (SCE) - Main Window



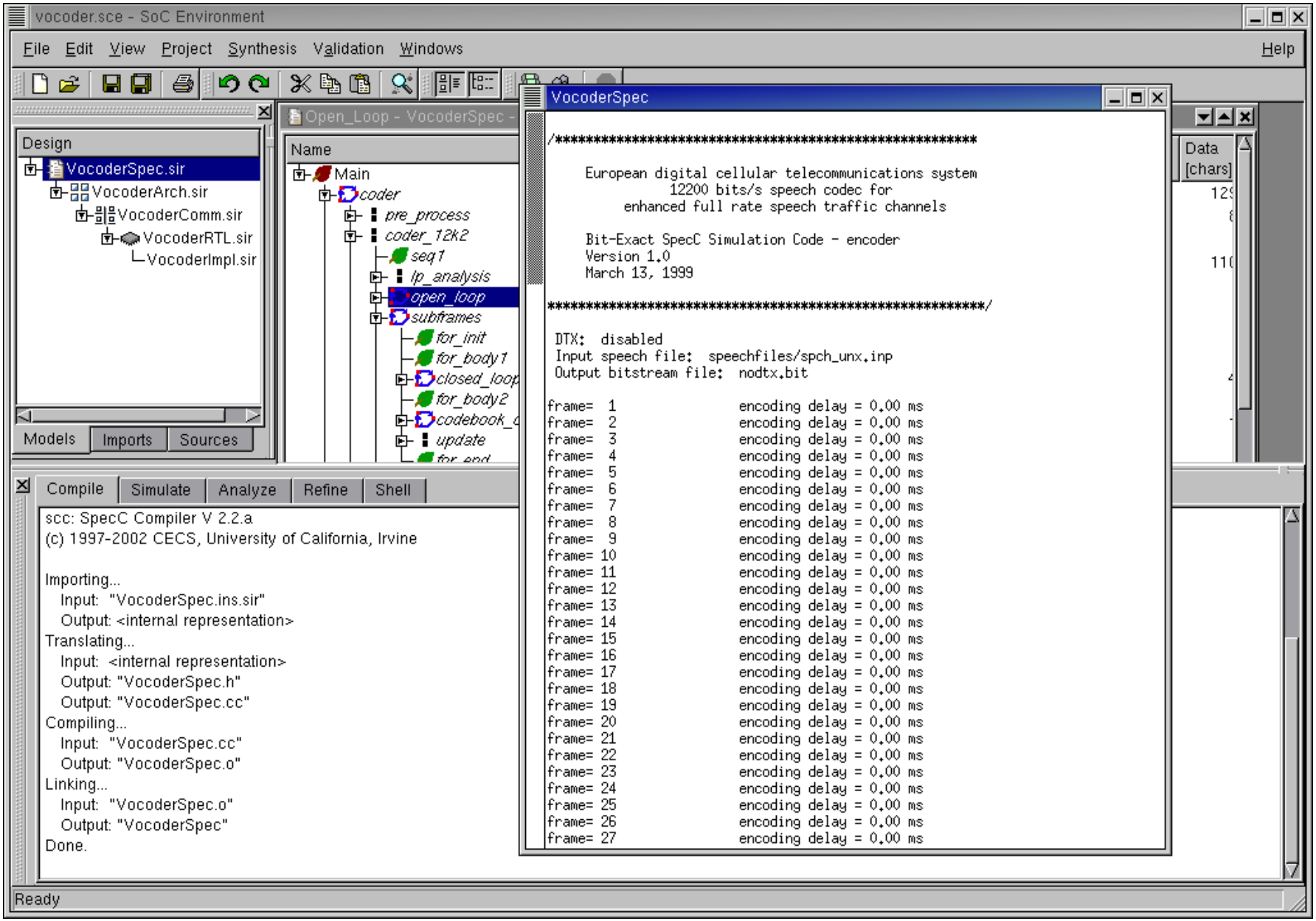
SCE Source Editor



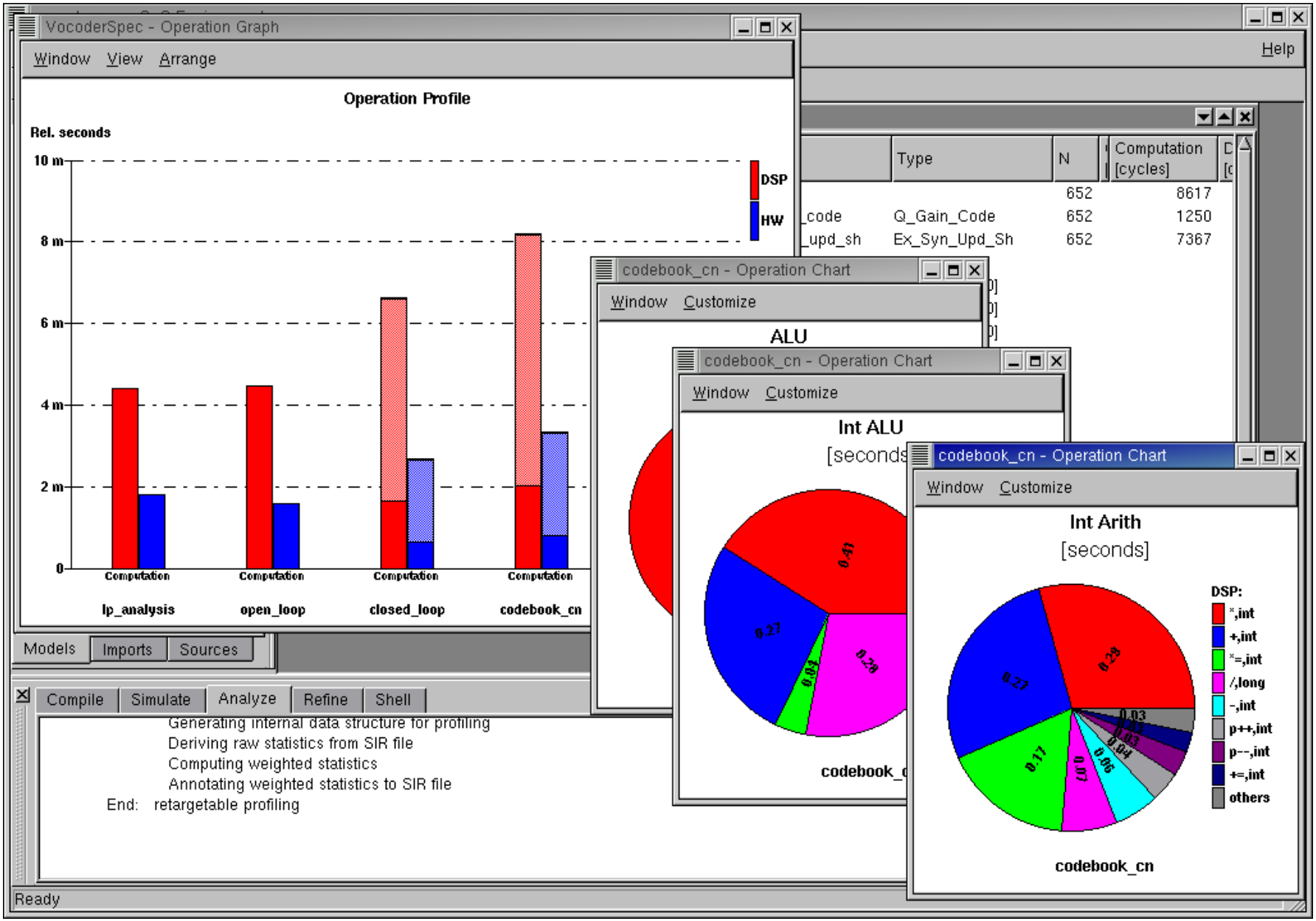
SCE Hierarchy Displays



SCE Compiler and Simulator



SCE Profiling and Analysis



Lecture 5: Summary

- **Design methodology**
  - Four levels of abstraction
    - Specification model: untimed, functional
    - Computation model: estimated, structural
    - Communication model: timed, bus-functional
    - Implementation model: cycle-accurate, RTL/IS
  - Three refinement steps
    - Computation refinement
    - Communication refinement
    - Processor refinement
      - » HW / SW / interface synthesis
  - Well-defined, formal models & transformations
    - Automatic, gradual refinement
    - Executable models, test bench re-use
    - Simple verification