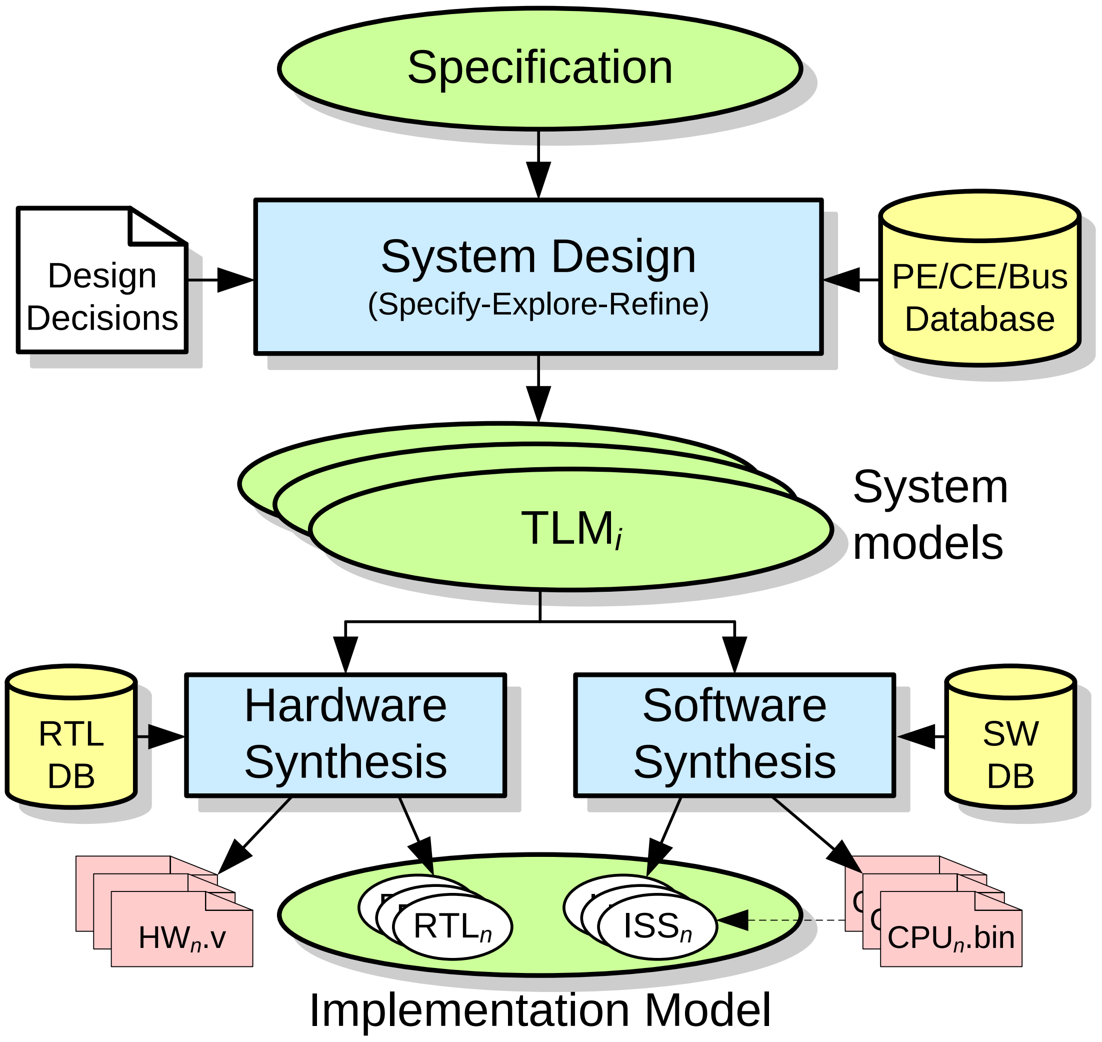
System-Level Design (and Modeling for Embedded Systems)

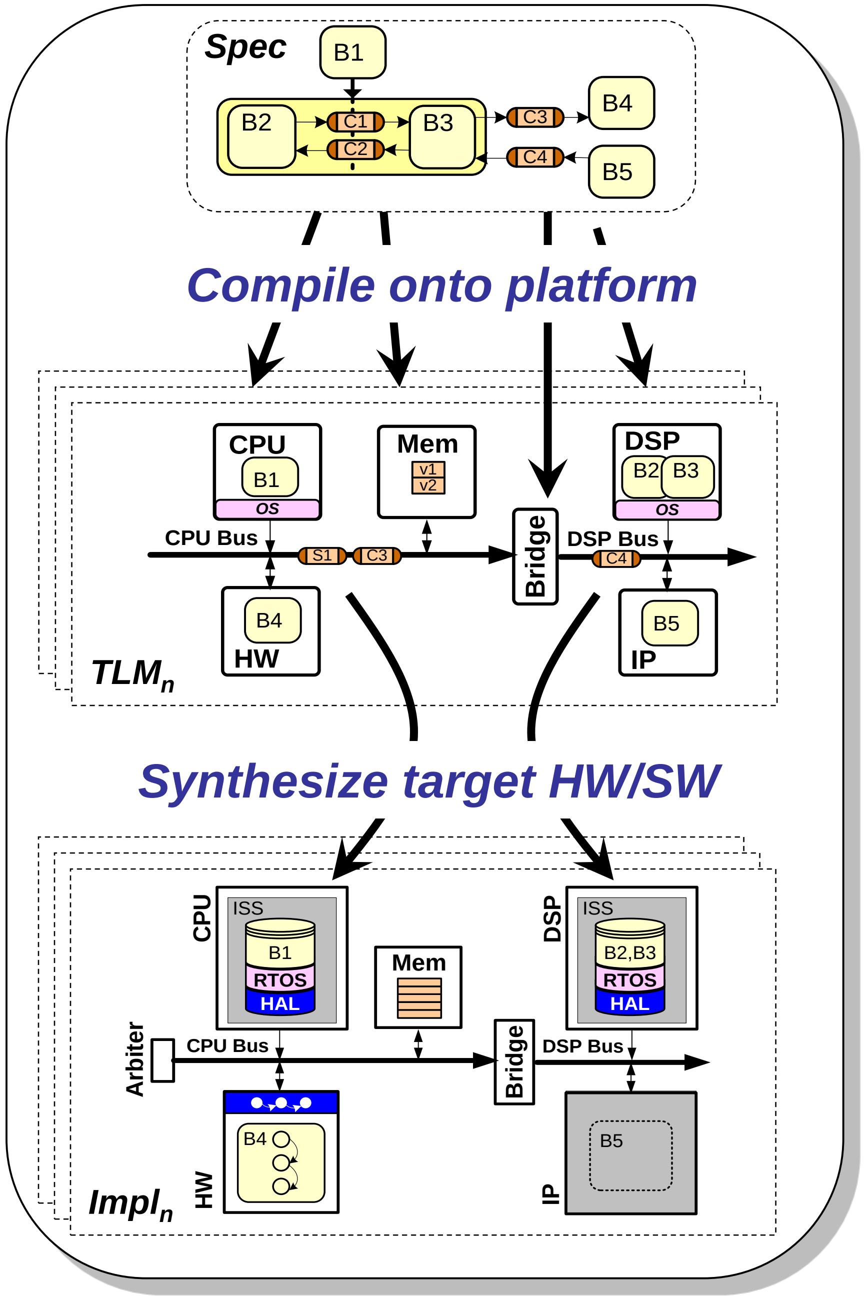
Lecture 5 – Specification and Refinement (Extended Version)

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

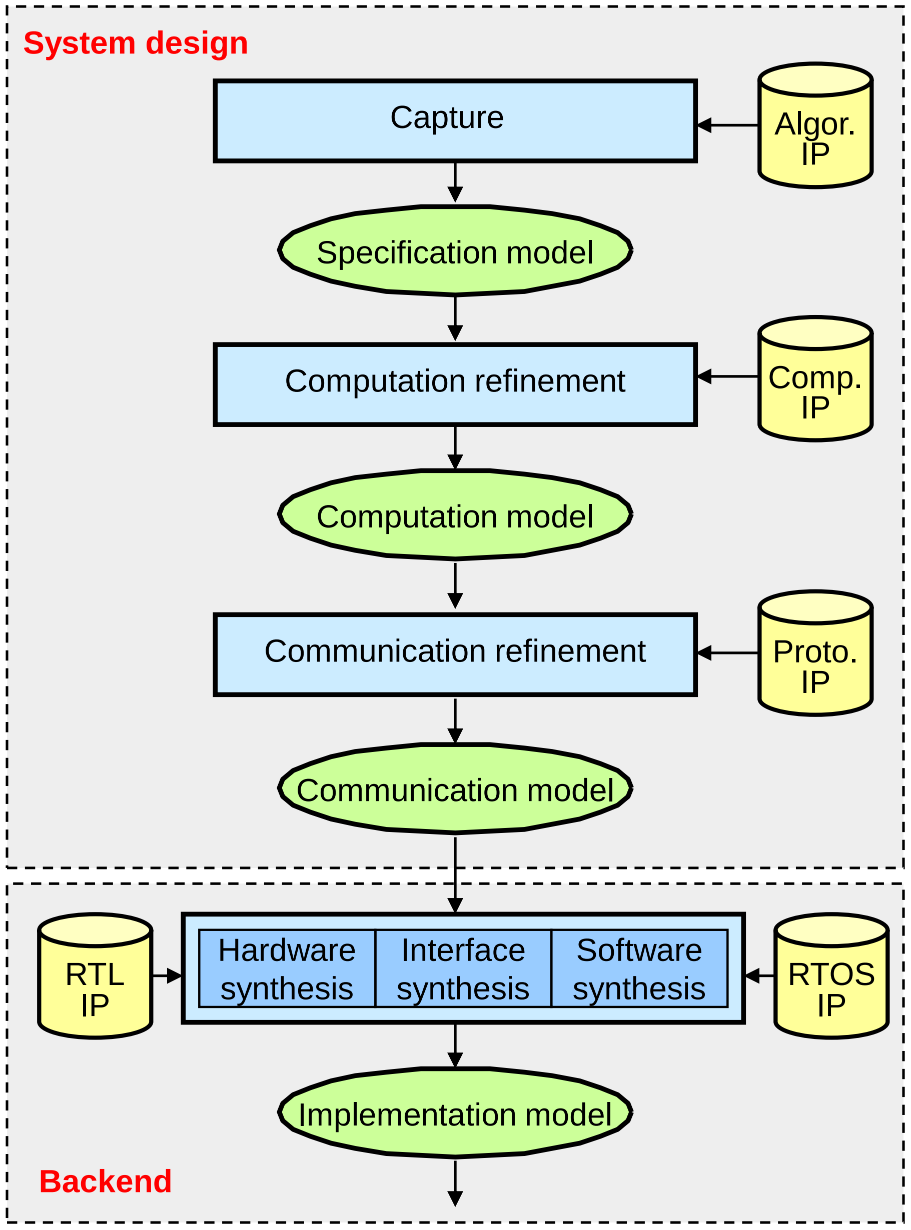
System Evolution and Operation
German Aerospace Center (DLR)
&
Distributed Computation and Communication
OFFIS

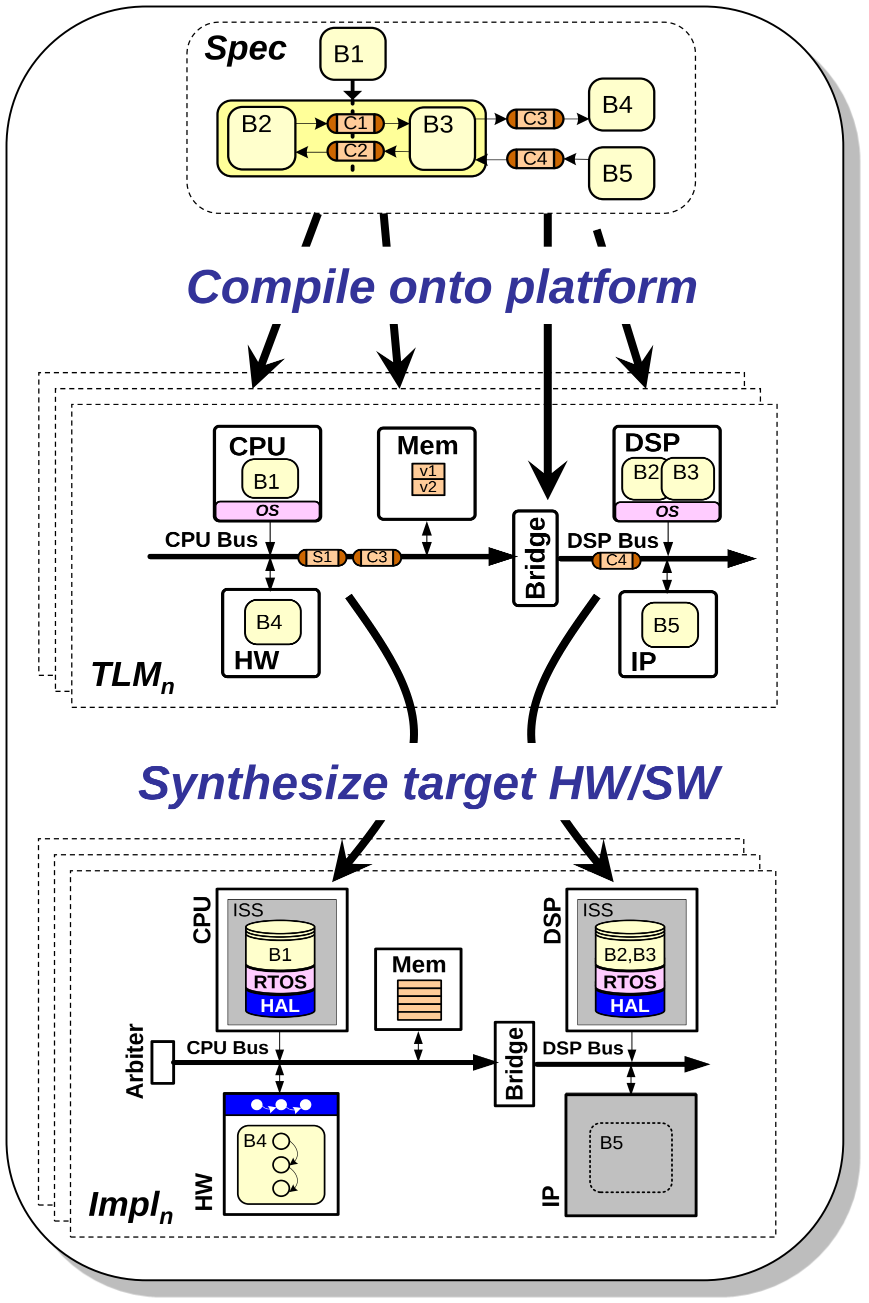
System-On-Chip Design Flow





System-On-Chip Design Flow

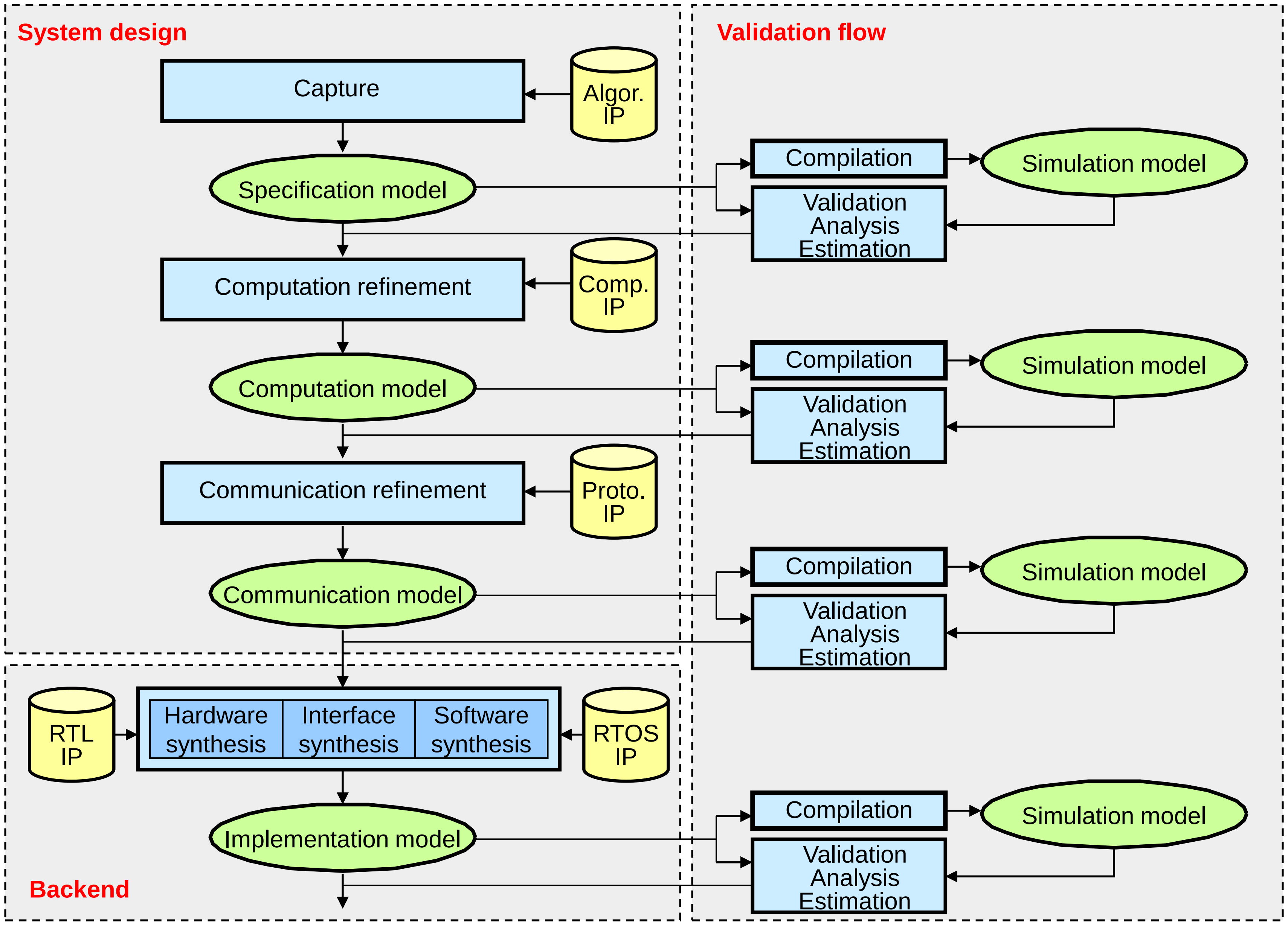




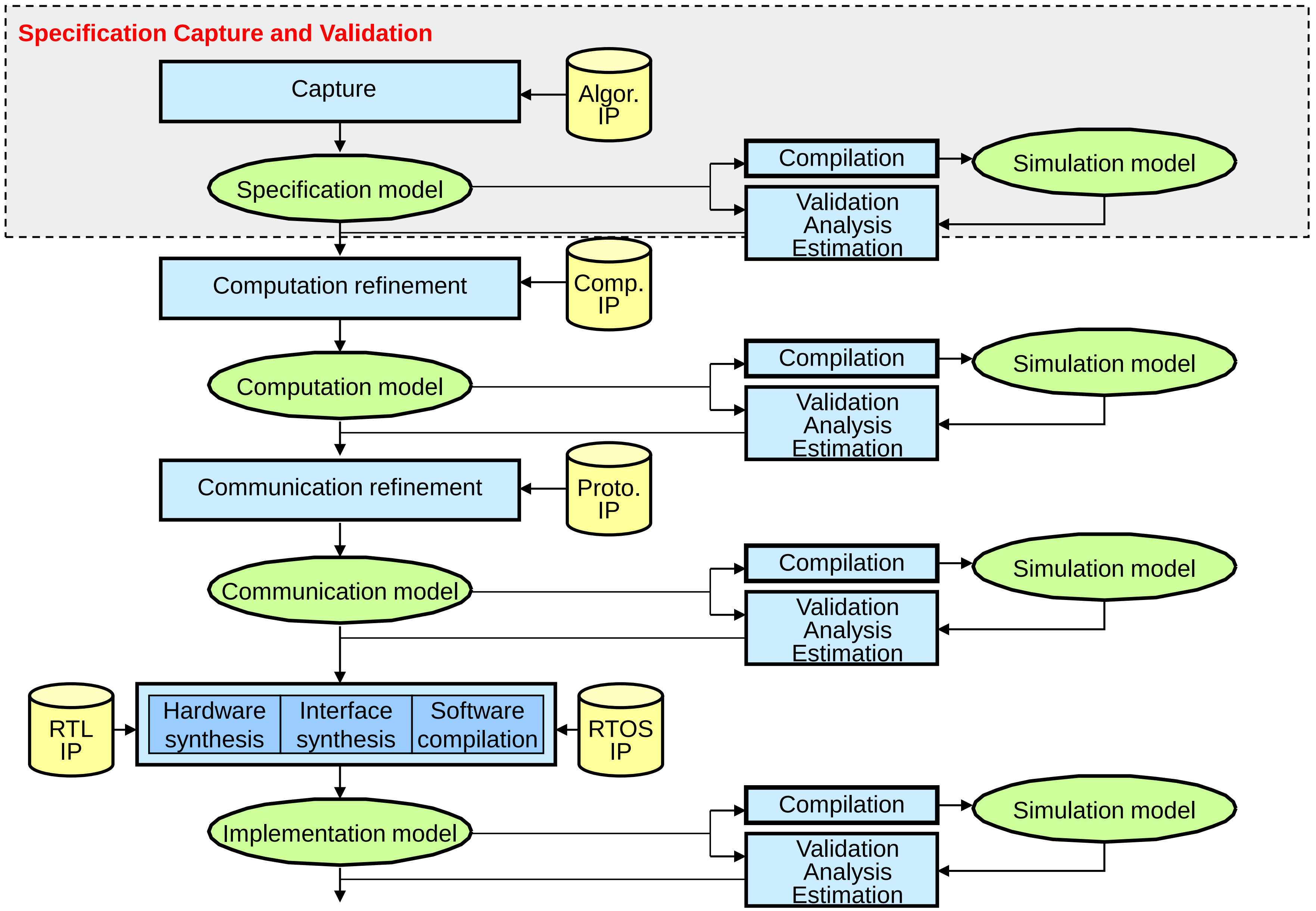
Lecture 5: Outline

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

Design Methodology

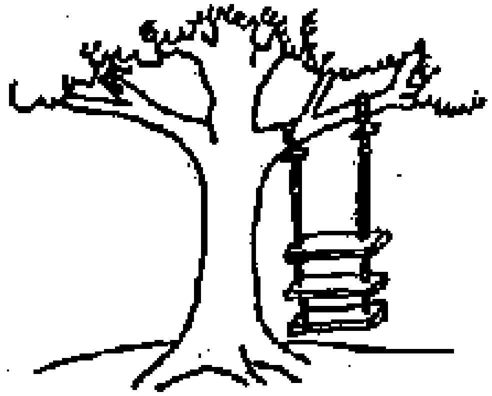


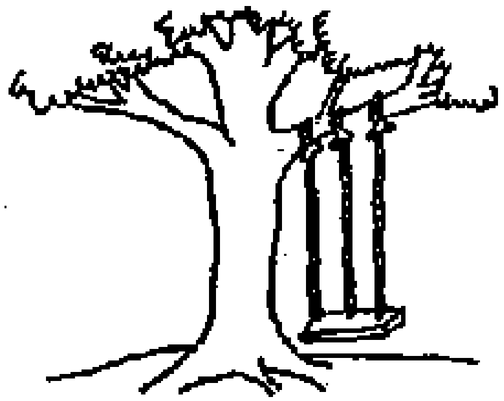
Design Methodology

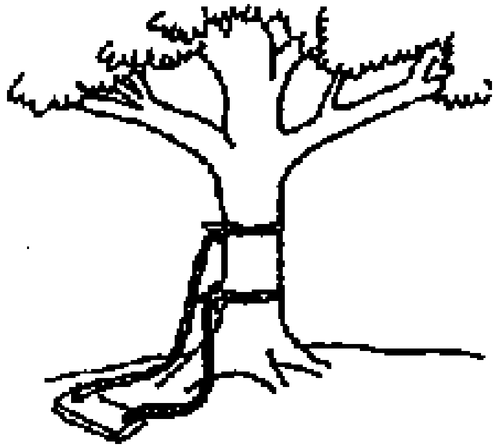


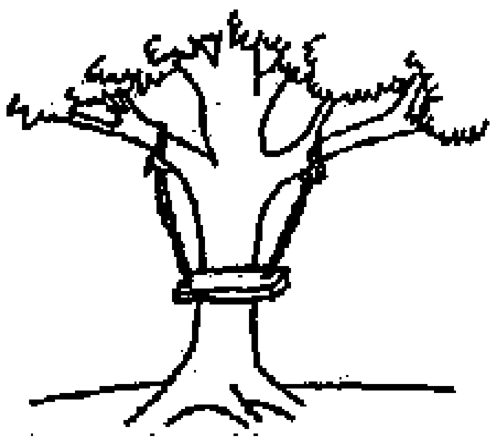
Essential Issues in Specification

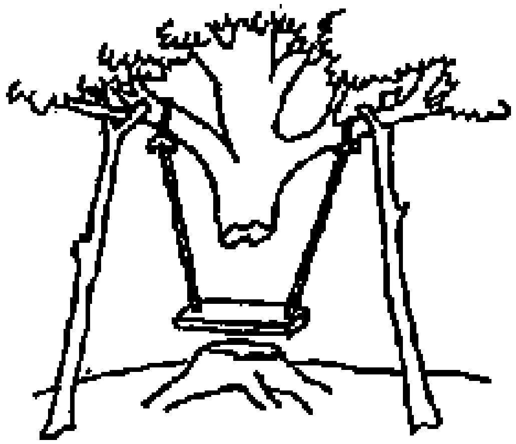
- **An Example ...**

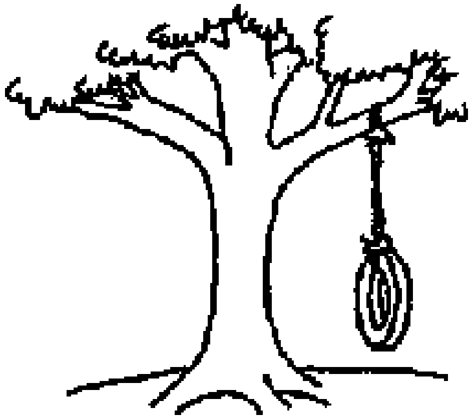

Proposed by the project team


Product specification


Product design by senior analyst


Product after implementation


Product after acceptance by user


What the user wanted

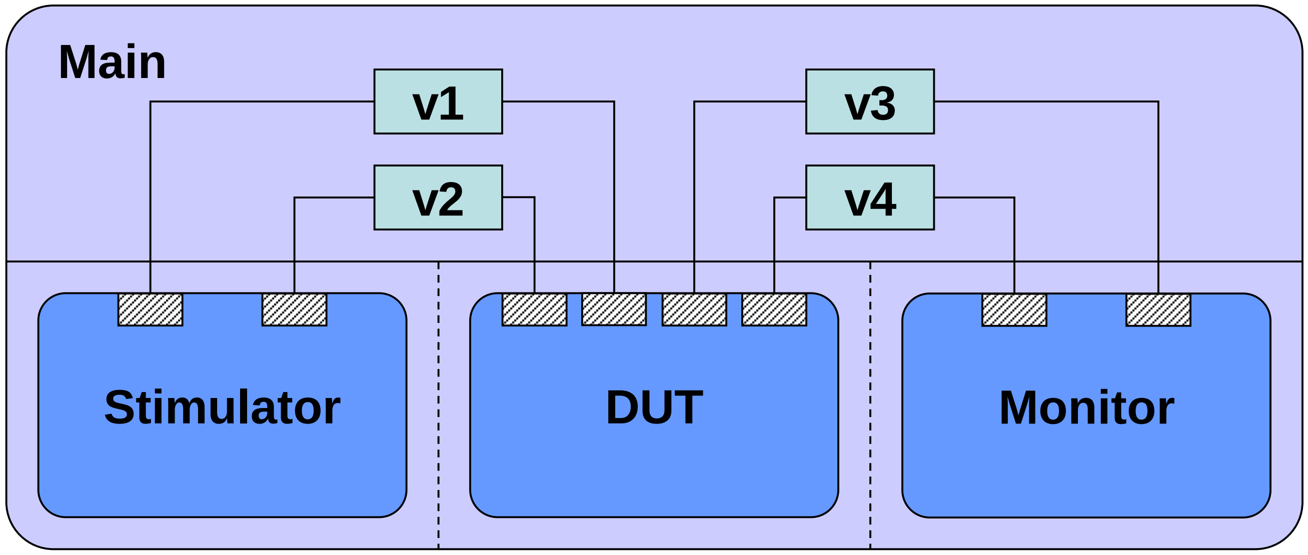
Source: unknown author, Courtesy of: R. Doemer

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

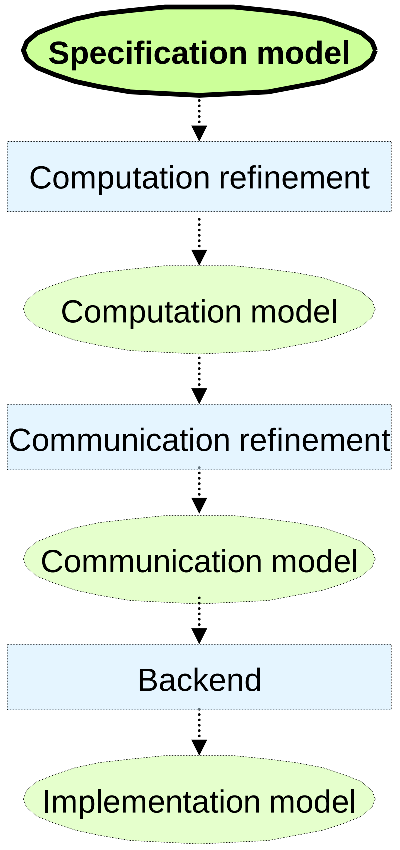
- **C code conversion to SpecC**
  - Functions become behaviors or channels
  - Functional hierarchy becomes behavioral hierarchy
    - Clean behavioral hierarchy required
    - if-then-else structure becomes FSM
    - while/for/do loops become FSM
  - Explicitly specify potential parallelism
    - Data (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

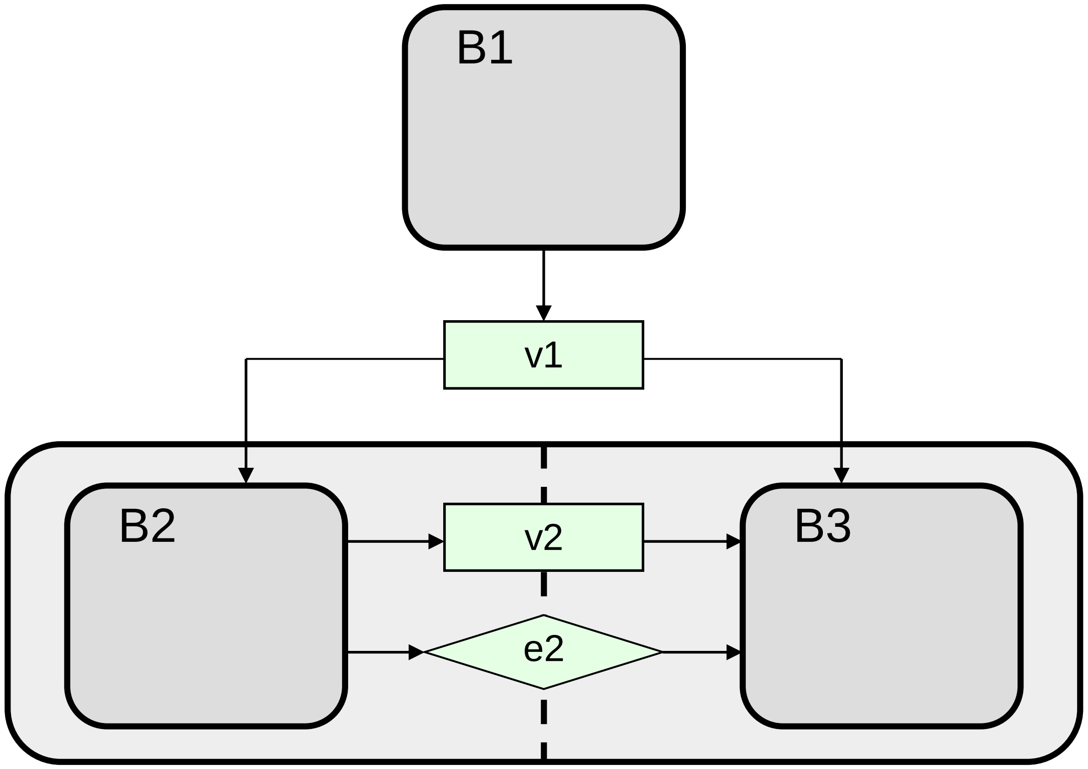
- ✓ **System specification**
  - ✓ Specification modeling
  - ✓ System validation
- **System refinement**
  - Computation
  - Communication
  - Implementation
- **SCE design environment**
  - Modeling
  - Refinement
  - Synthesis

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

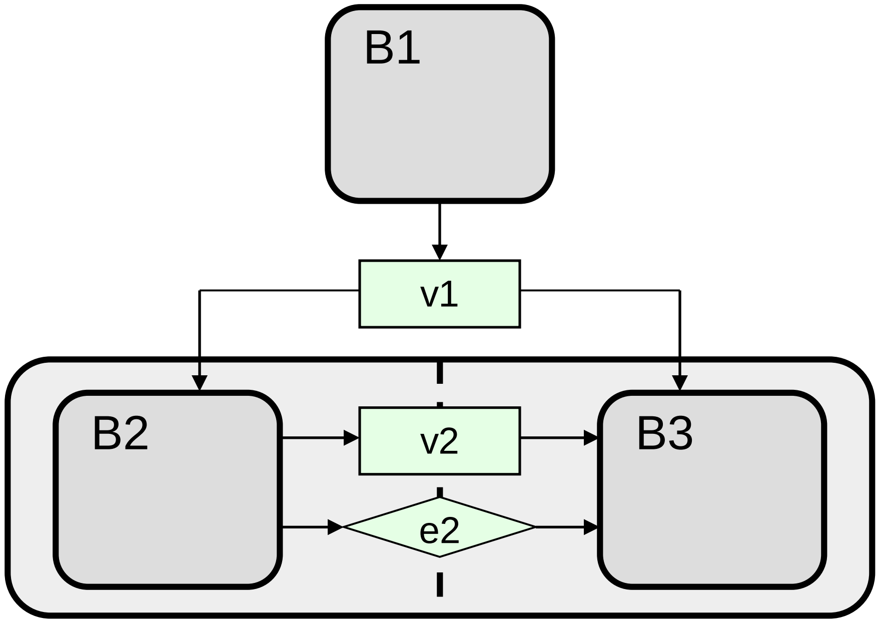


Specification Model Example



- **Simple, typical specification model**
  - Hierarchical parallel-serial composition
  - Communication through ports and variables, events

SpecC Specification Model Example (2)

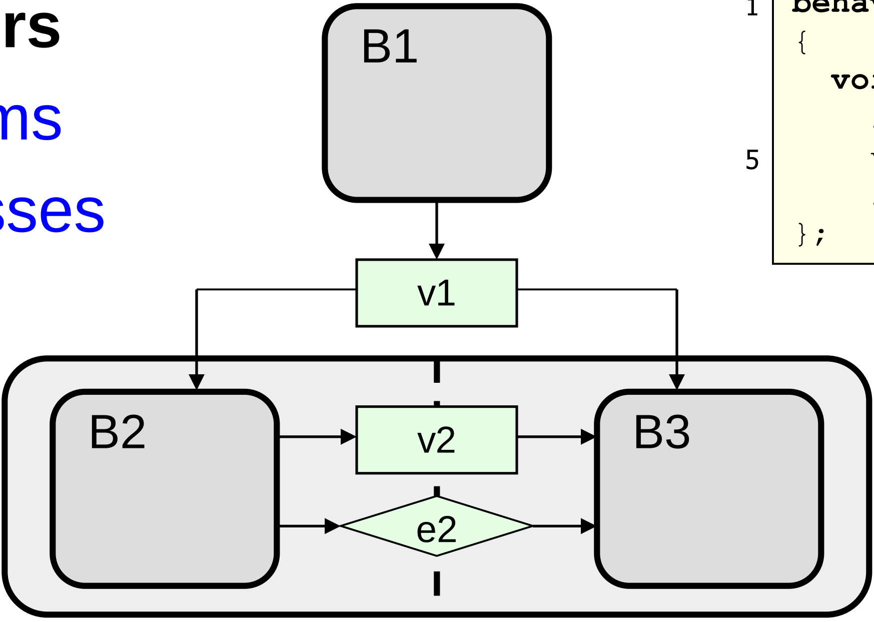


SpecC design hierarchy:

```
behavior B2B3( in int v1 )
{
  int   v2;                // variables
  event e2;

  B2 b2( v1, v2, e2 );   // children
  B3 b3( v1, v2, e2 );

  void main(void) {
    par {
     b2.main();
     b3.main();
    }
  }
};

behavior Design()
{
  int v1;                  // variables

  B1   b1   ( v1 );        // children
  B2B3 b2b3( v1 );

  void main(void) {
     b1.main();
     b2b3.main();
  }
};
```

SpecC Specification Model Example (3)

- **Leaf behaviors**
  - C algorithms
  - Port accesses



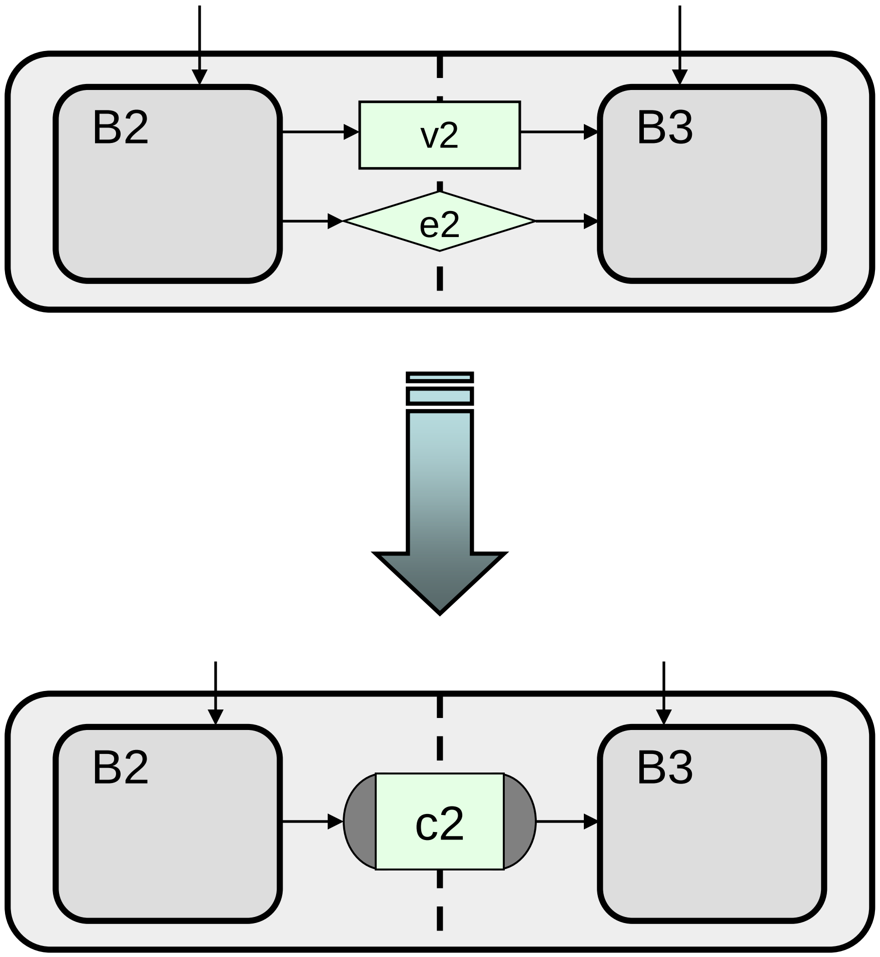
```
behavior B1( out int v1 )
{
  void main(void) {
    …
    v1 = …     // write v1
    …
};
```

```
behavior B2( in  int   v1,
             out int   v2,
             out event e2 )
{
   void main(void) {      // read v1
      …
      v2 = f2( v1, … ); // produce v2
      notify( e2 );      // synchronize
      …
    }
};
```

```
behavior B3( in int    v1,
             in int    v2,
             in event e2 )
{
    void main(void) {     // read v1
       …
       wait( e2 );         // wait for sync
       f3( v1, v2, … ); // consume v2
       …
    }
};
```

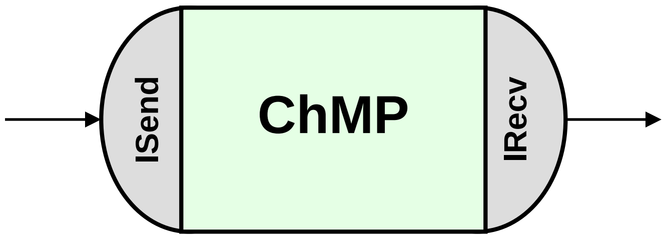
Communication

- **Message-passing**
  - Abstract communication and synchronization
  - Encapsulate in channel



Message-Passing Channel

- Blocking, unbuffered message-passing



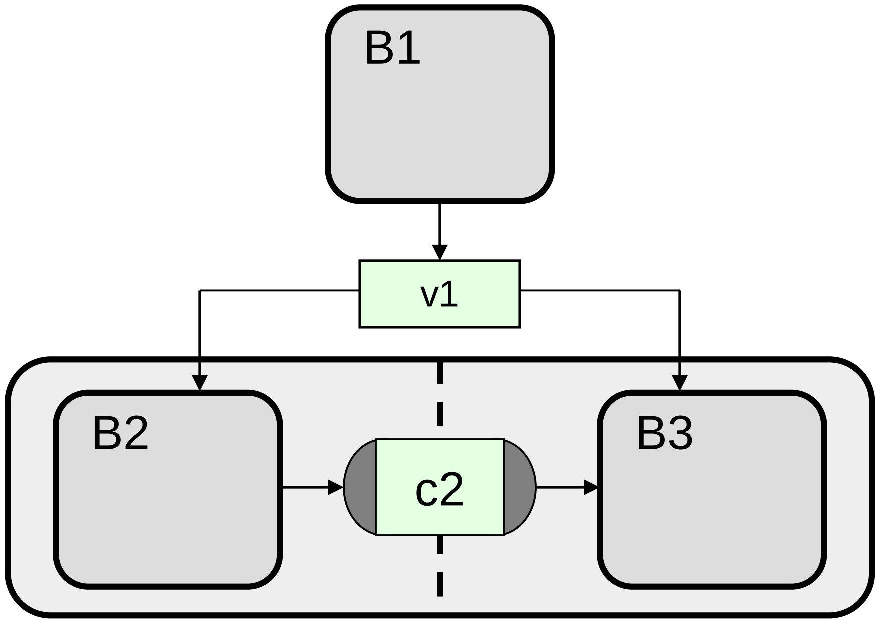
SpecC Channel interfaces:

```
interface ISend {
  void send( void *d, int size );
};

interface IRecv {
  void recv( void *d, int size );
};
```

SpecC Simulation model:

```
channel ChMP() implements ISend, IRecv
{
  void  *buf = 0;
  event eReady, eAck;

  void send( void *d, int size ) {
    buf = malloc( size );
    memcpy( buf, d, size );
    notify( eReady );
    wait( eAck );
    free( buf );
    buf = 0;
  }

  void recv( void *d, int size ) {
    if( !buf ) wait( eReady );
    memcpy( d, buf, size );
    notify( eAck );
  }
};
```

SpecC Message-Passing Specification Model



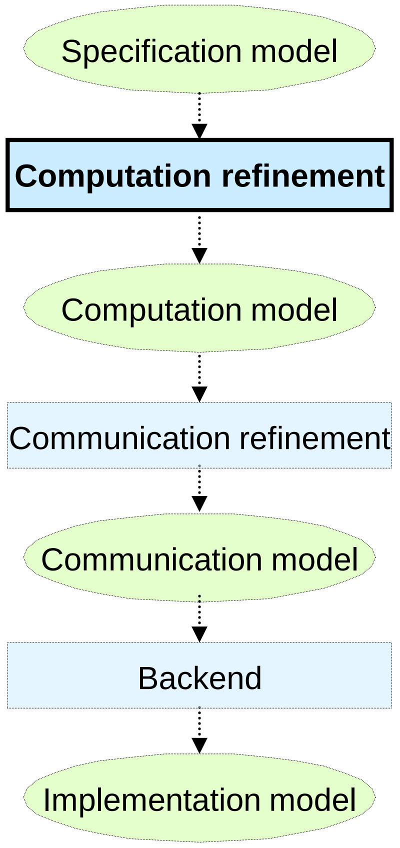
```
behavior B2B3( in int v1 )
{
  ChMP c2;

  B2   b2( v1, c2 );

  B3   b3( v1, c2 );

  void main(void) {
    par {
      b2.main();
      b3.main();
    }
  }
};
```

```
behavior B2( in int v1, ISend c2 )
{
  void main(void) {
    …
    v2 = f2( v1, … );
    c2.send( &v2, sizeof(v2) );
    …
  }
};
```

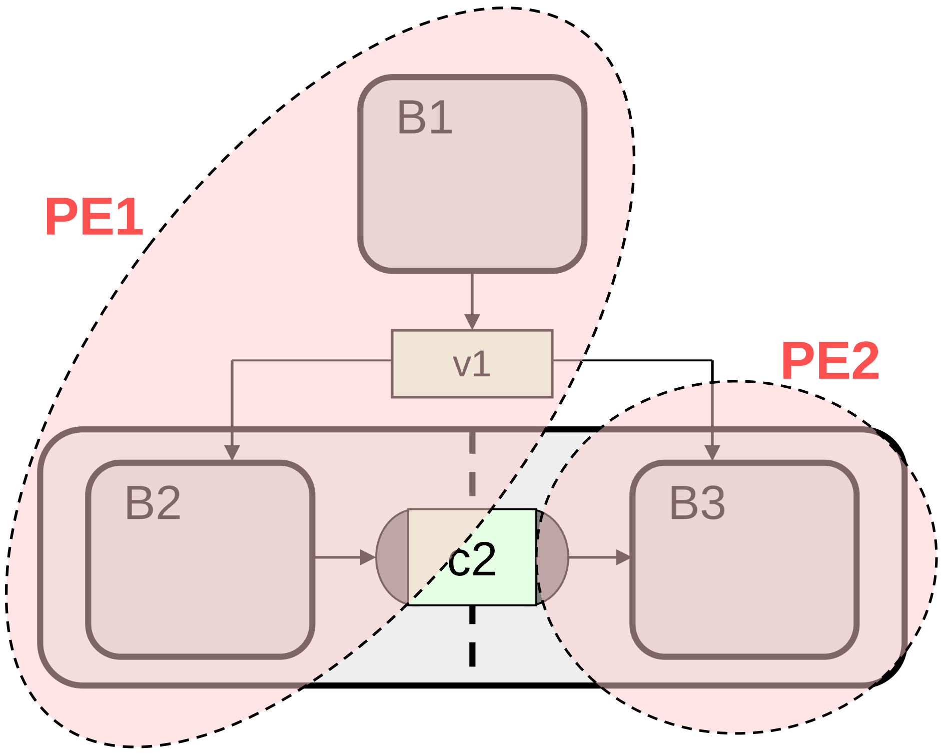
```
behavior B3( in int v1, IRecv c2 )
{
  void main(void) {
    …
    c2.recv( &v2, sizeof(v2) );
    f3( v1, v2, …);
    …
  }
};
```

Computation refinement

- Component allocation / selection
- Behavior partitioning
- Variable partitioning
- Scheduling



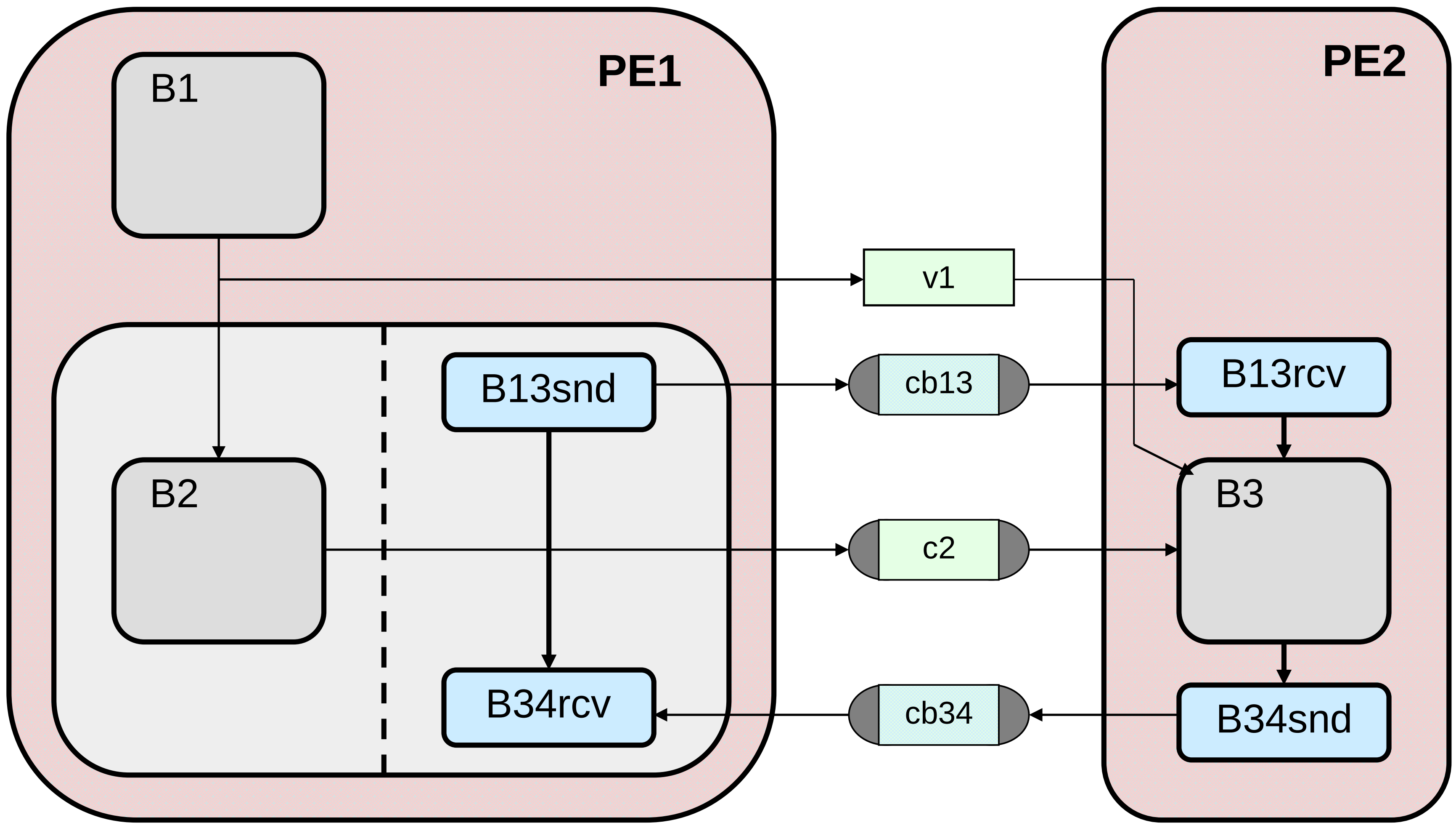
Allocation, Behavior Partitioning



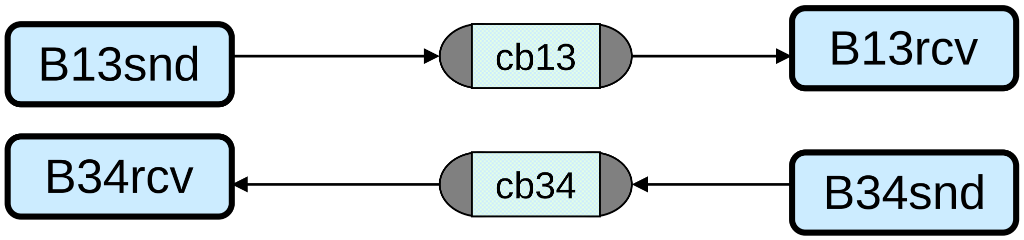
- Allocate PEs
- Partition behaviors
- Globalize communication

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

Model after Behavior Partitioning



Synchronization



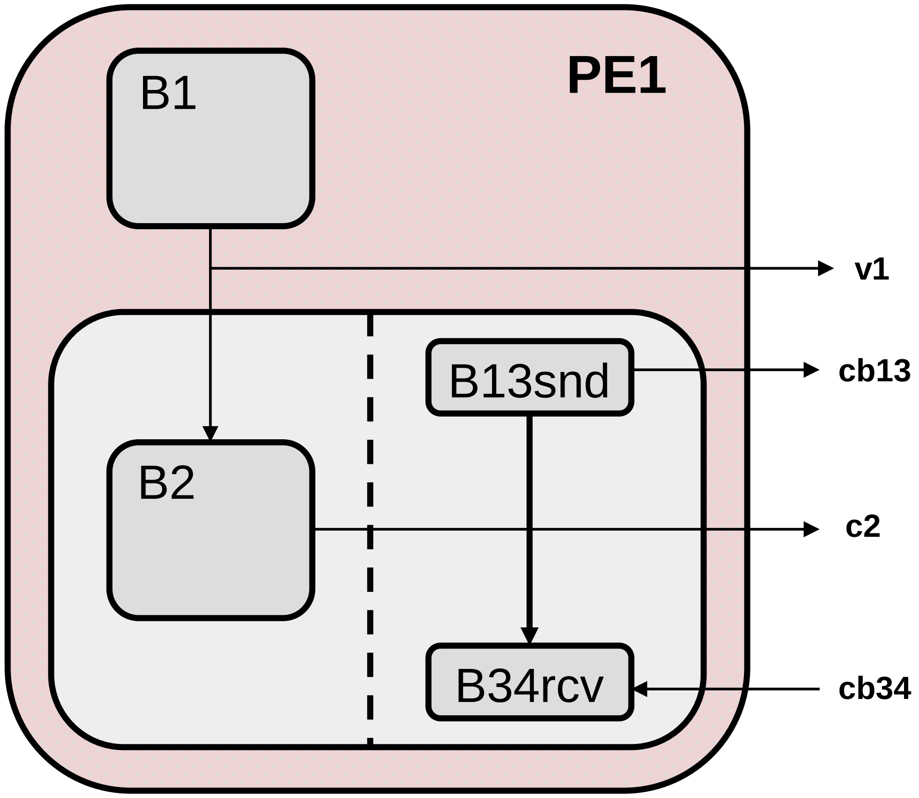
- **For each component-crossing transition**
  - Synchronization behavior pair
  - Synchronize over blocking message-passing channel

```
behavior BSnd( ISend ch )
{
  void main(void)  {
    ch.send( 0, 0 );
  }
};
```

```
behavior BRcv( IRecv ch )
{
  void main(void)  {
    ch.recv( 0, 0 );
  }
};
```

➢ **Preserve execution semantics**

SpecC Model after Behavior Partitioning (PE1)



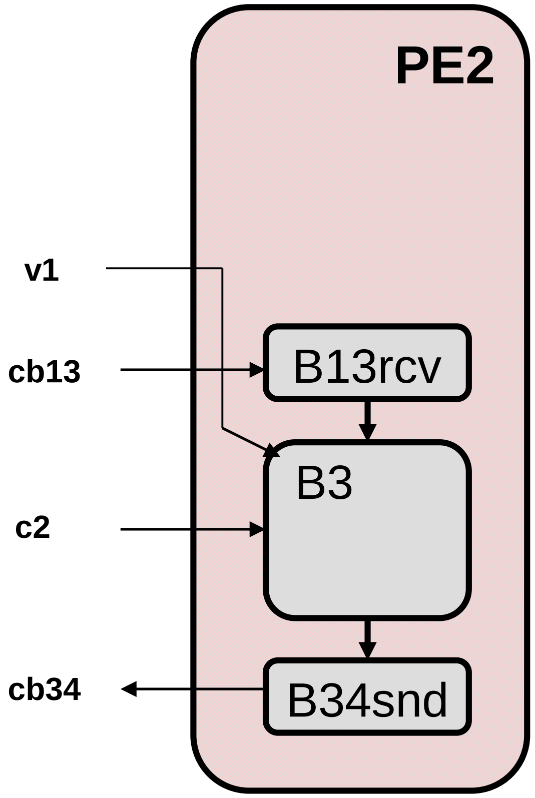
```
behavior PE1( int v1,
              ISend cb13,
              ISend c2,
              IRecv cb34 )
{
  B1   b1   ( v1 );
  B2B3 b2b3( v1, cb13, c2, cb34 );

  void main(void) {
    b1.main();
    b2b3.main();
  }
};
```

```
behavior B2B3( in int v1,
               ISend cb13,
               ISend c2,
               IRecv cb34 )
{
  B2     b2    ( v1, c2 );
  B3Stub b3stub( cb13, cb34 );

  void main(void) {
    par {
      b2.main();
      b3stub.main();
    }
  }
};
```

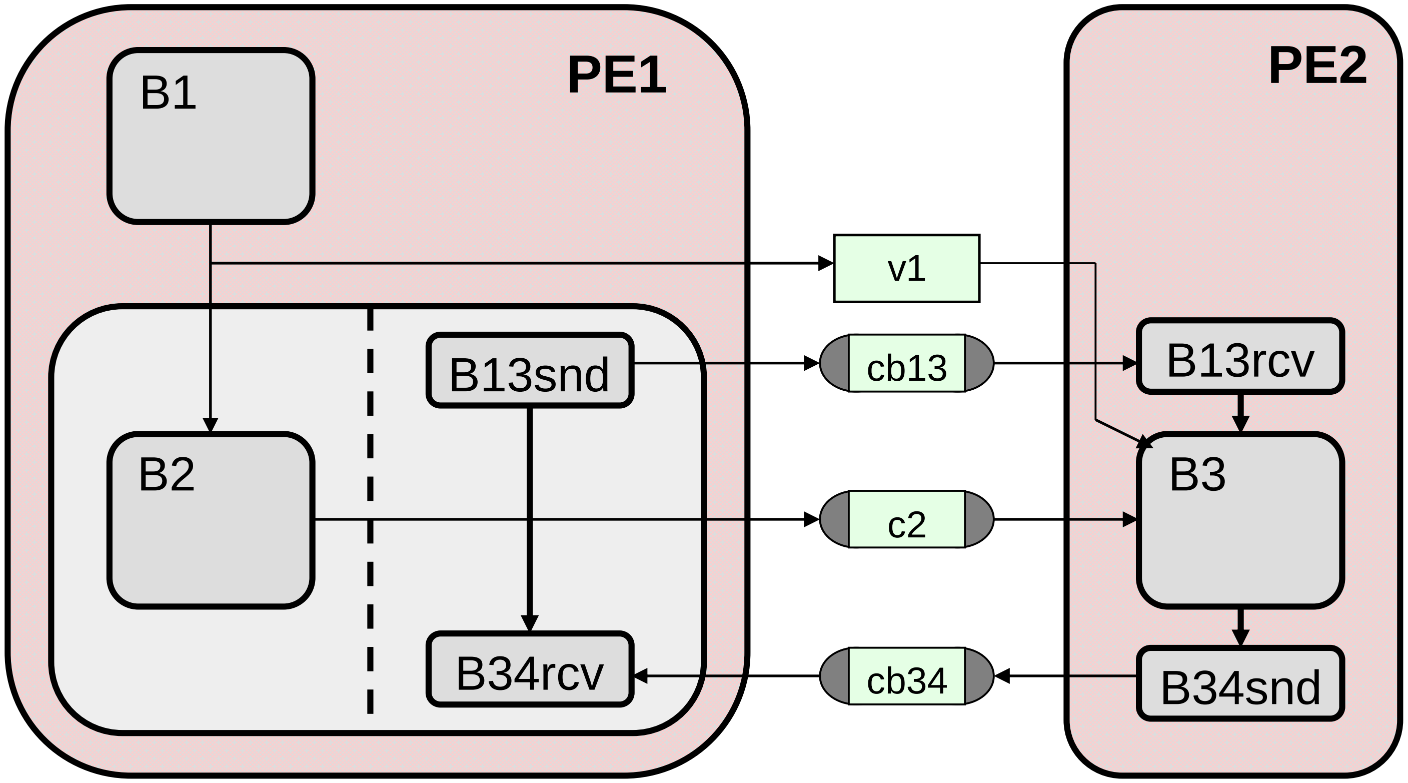
```
behavior B3Stub( ISend cb13, IRecv cb34 )
{
  BSnd b13snd( cb13 );
  BRcv b34rcv( cb34 );

  void main(void) {
    b13snd.main();
    b34rcv.main();
  }
};
```

SpecC Model after Behavior Partitioning (PE2)

```
behavior PE2( in int v1,
              IRecv cb13,
              IRecv c2,
              ISend cb34)
{
  BRcv b13rcv( cb13 );
  B3   b3    ( v1, c2 );
  BSnd b34snd( cb34 );

  void main(void)
  {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```



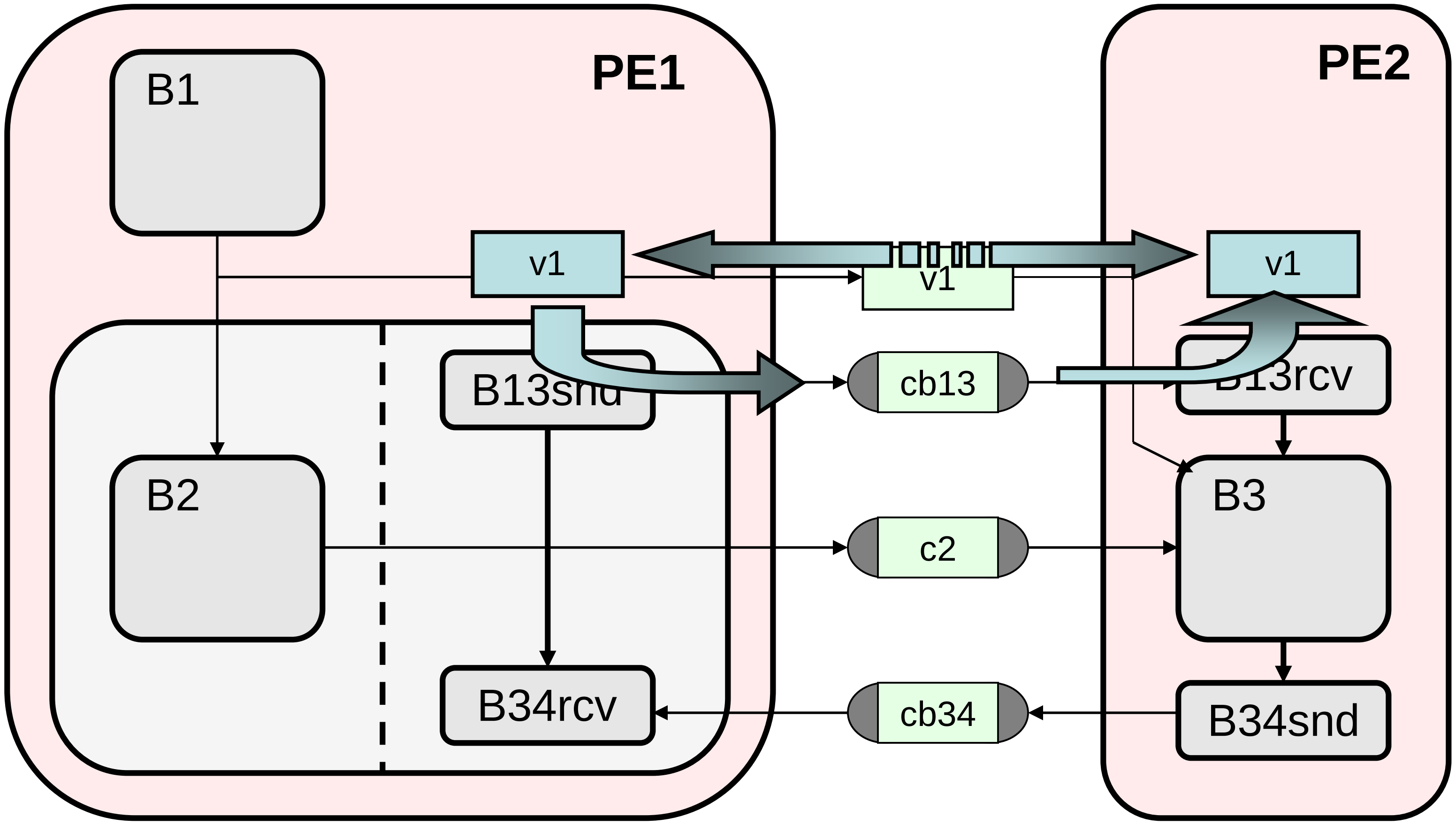
SpecC Model after Behavior Partitioning (Top)



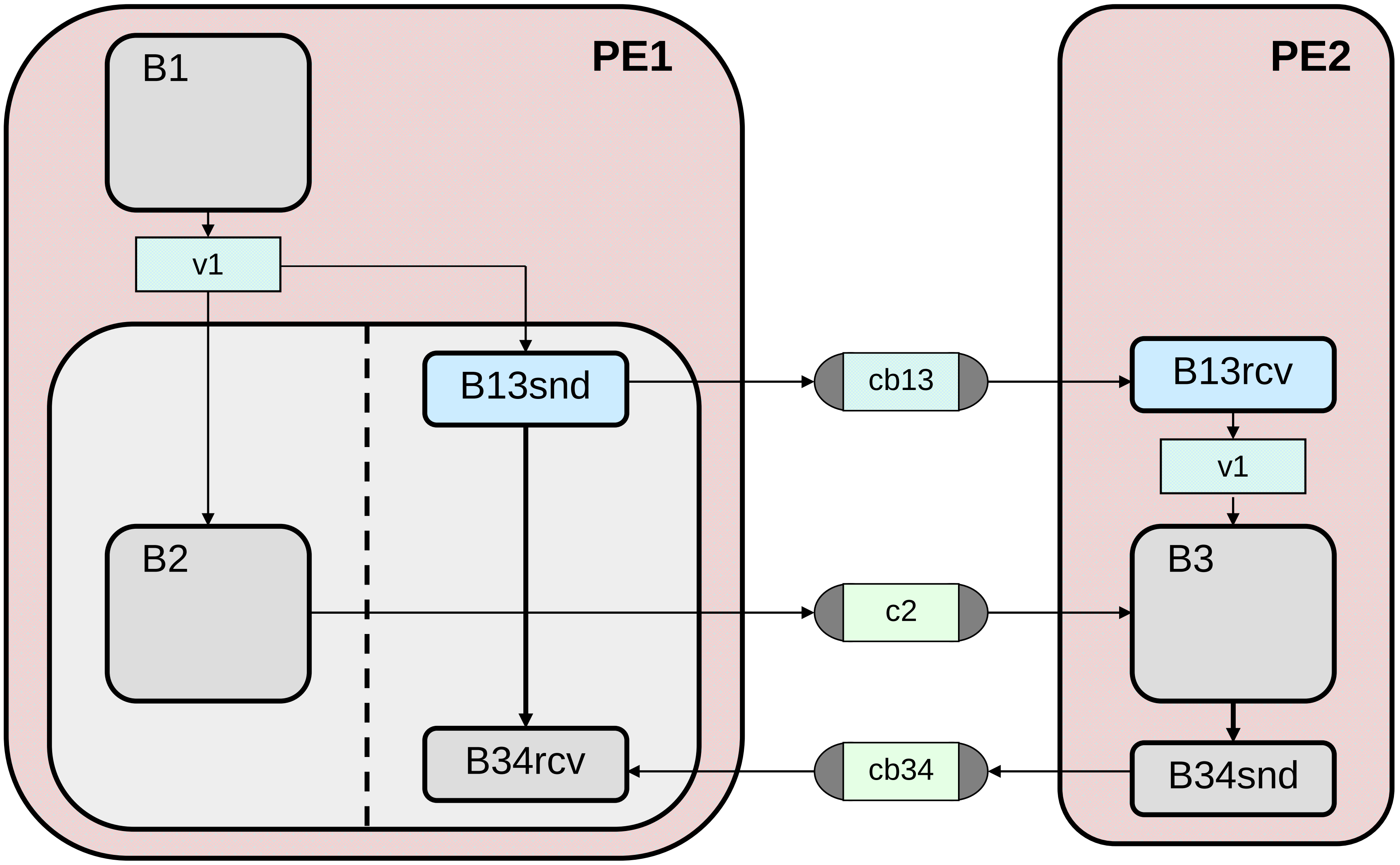
```
behavior Design() {
  int v1;
  ChMP cb13, c2, cb34;

  PE1 pe1( v1, cb13, c2, cb34 );
  PE2 pe2( v1, cb13, c2, cb34 );

  void main(void) {
    par { pe1.main(); pe2.main(); }
  }
};
```

Variable Partitioning

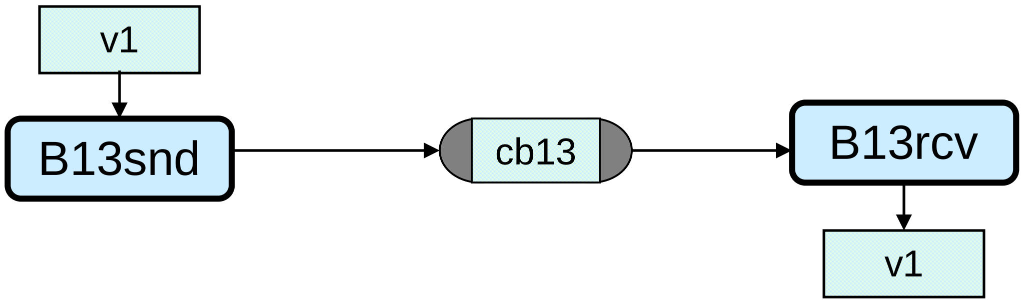
- Shared memory vs. message passing implementation
  - Map global variables to local memories
  - Communicate data over message-passing channels



Message-Passing Model



SpecC Message-Passing Communication



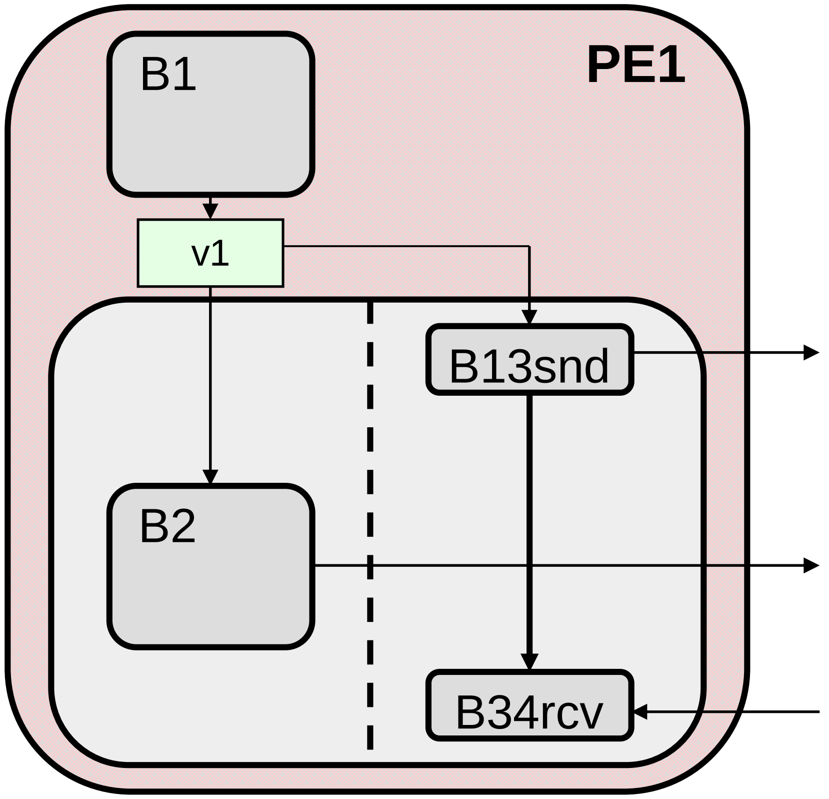
- **Keep local variable copies in sync**
  - Communicate updated values at synchronization points
  - Transfer control & data over message-passing channel

```
behavior B13Snd( in int v1, ISend ch )
{
  void main(void) {
    ch.send( &v1, sizeof(v1) );

  }
};
```

```
behavior B13Rcv( IRecv ch, out int v1 )
{
  void main(void) {
    ch.recv( &v1, sizeof(v1) );

  }
};
```

➢ **Preserve shared semantics of variables**

SpecC Message-Passing Model (PE1)



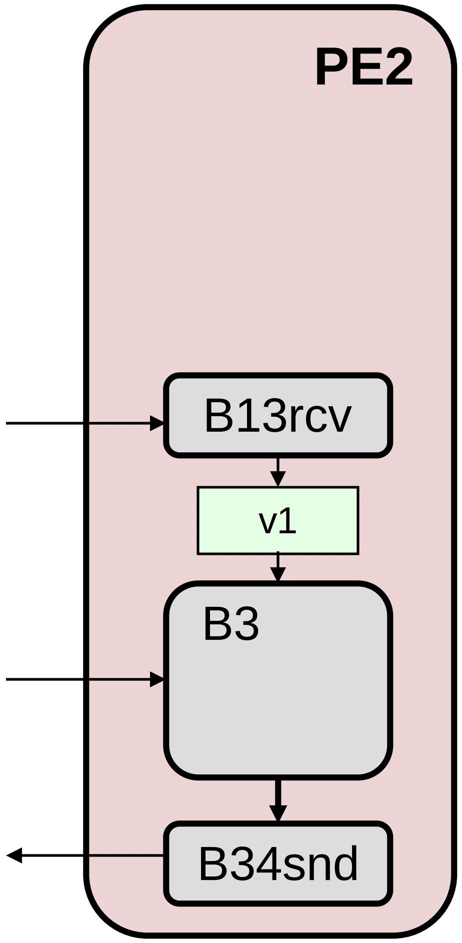
```
behavior PE1( ISend cb13,
              ISend c2,
              IRecv cb34)
{
  int v1;

  B1   b1  ( v1 );
  B2B3 b2b3( v1, cb13, c2, cb34 );

  void main(void) {
    b1.main();
    b2b3.main();
  }
};
```

```
behavior B2B3( in int v1,
               ISend cb13,
               ISend c2,
               IRecv cb34)
{
  B2     b2     ( v1, c2 );
  B3stub b3stub( v1 , cb13, cb34 );

  void main(void) {
    par {
      b2.main();
      b3stub.main();
    }
  }
};
```

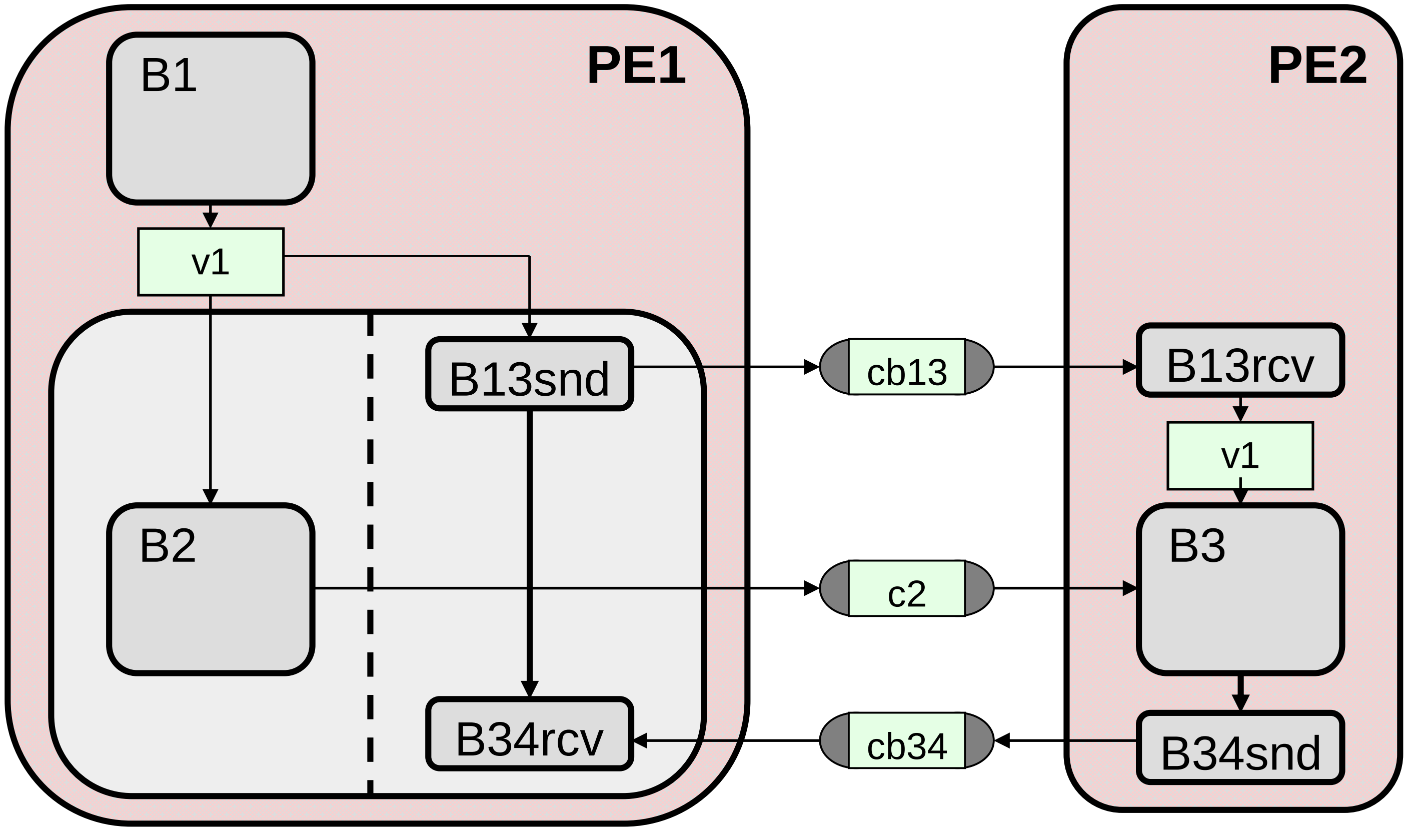
```
behavior B3stub( in int v1 , ISend cb13,
                             IRecv cb34 ) {
  B13Snd b13snd( v1, cb13 );
  BRcv   b34rcv( cb34 );

  void main(void) {
    b13snd.main();
    b34rcv.main();
  }
};
```

SpecC Message-Passing Model (PE2)

```
behavior PE2( IRecv cb13,
              IRecv c2,
              ISend cb34 )
{
  int v1;

  B13Rcv b13rcv( cb13 ,v1 );
  B3     b3     ( v1, c2 );
  BSnd   b34snd( cb34 );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```



SpecC Message-Passing Model (Top)



```
behavior Design() {
  ChMP cb13, c2, cb34;

  PE1 pe1( cb13, c2, cb34 );
  PE2 pe2( cb13, c2, cb34 );

  void main(void) {
    par { pe1.main(); pe2.main(); }
  }
};
```

Timed Computation

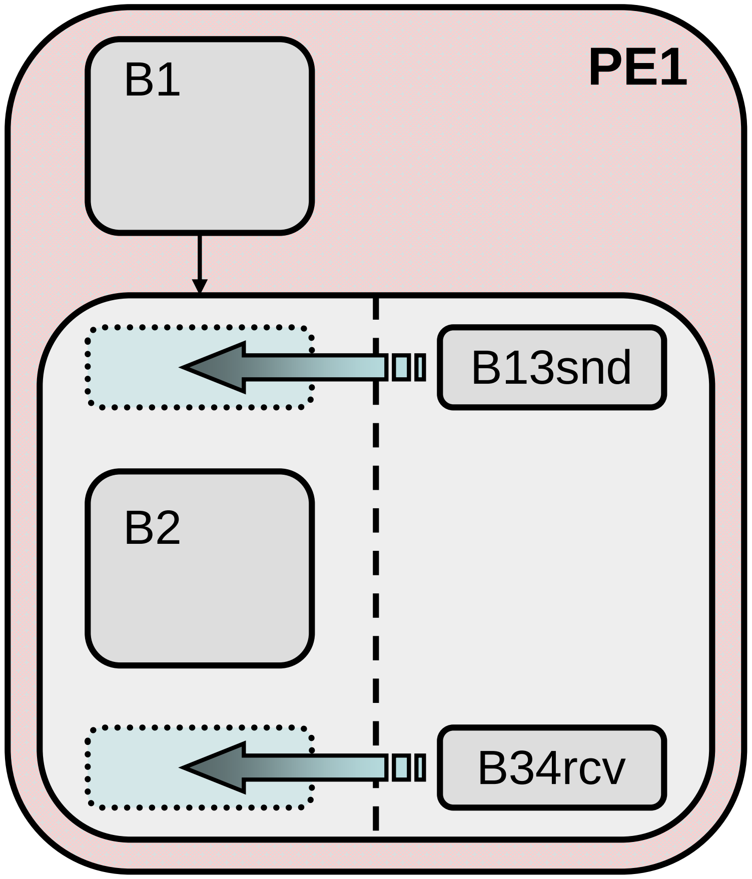
- **Execution time of behaviors**
  - Estimated target delay / timing budget
- **Granularity**
  - Behavior level / basic block level

➢ **Annotate behaviors**
  - Simulation feedback
  - Synthesis constraints

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

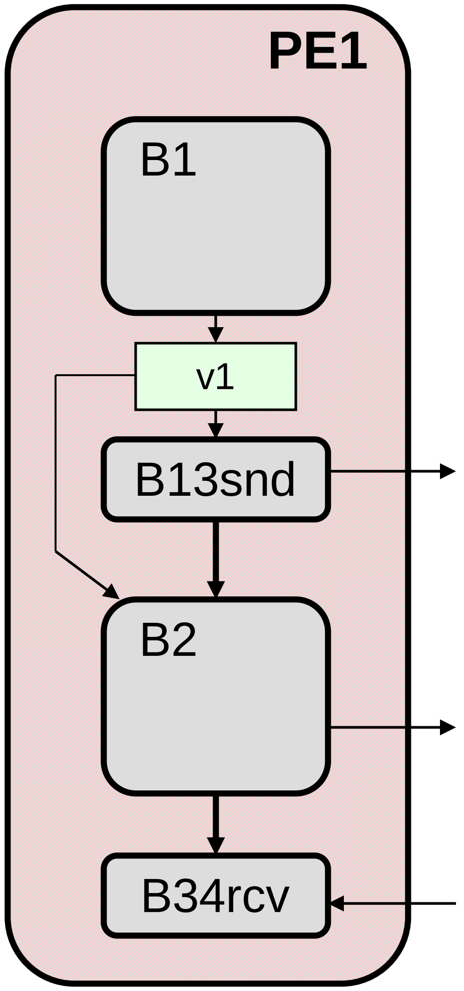
Scheduling

➤ **Serialize behavior execution on components**



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

SpecC Model after Scheduling (PE1)



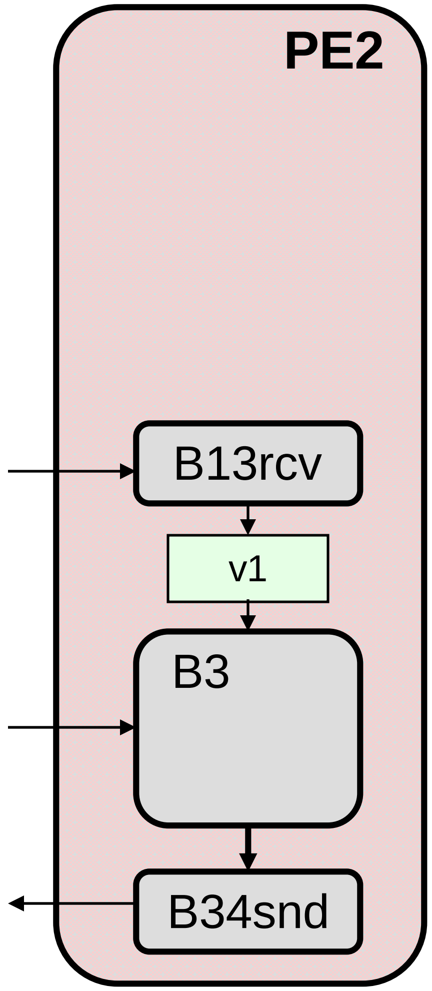
- **Statically scheduled PE1**

```
behavior PE1( ISend cb13,
              ISend c2,
              IRecv cb34 )
{
  int v1;

  B1     b1     ( v1 );
  B13Snd b13snd( v1, cb13 );
  B2     b2     ( v1, c2 );
  BRcv   b34rcv( cb34 );

  void main(void)
  {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};
```

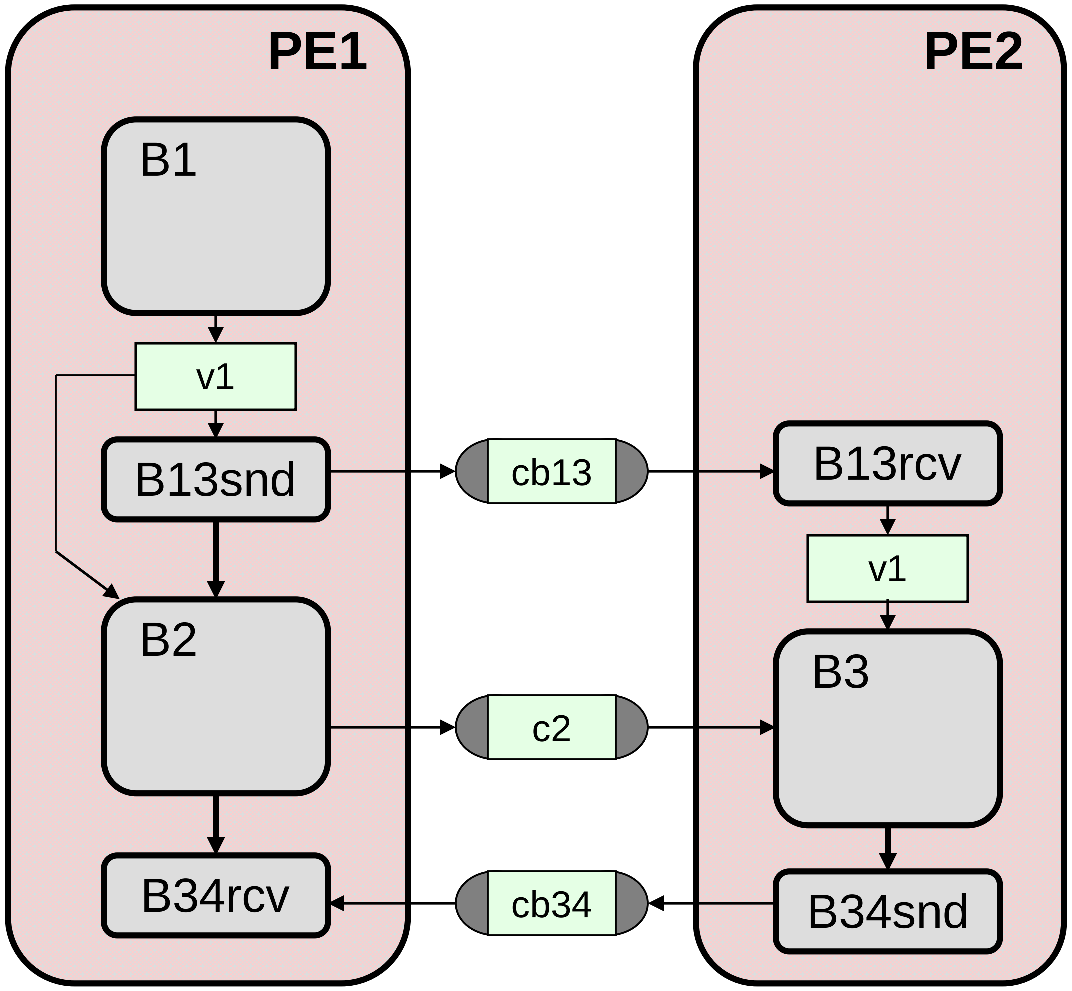
SpecC Model after Scheduling (PE2)

- No scheduling necessary for PE2

```
behavior PE2( IRecv cb13,
              IRecv c2,
              ISend cb34 )
{
  int v1;

  B13Rcv b13rcv( cb13, v1 );
  B3     b3     ( v1, c2 );
  BSnd   b34snd( cb34 );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```



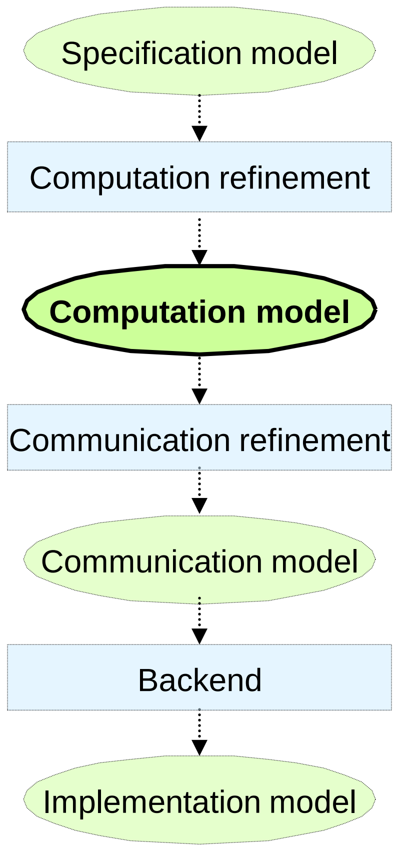
SpecC Model after Scheduling (Top)



```
behavior Design()
{
  ChMP cb13, c2, cb34;

  PE1 pe1( cb13, c2, cb34 );
  PE2 pe2( cb13, c2, cb34 );

  void main(void) {
    par {
      pe1.main();
      pe2.main();
    }
  }
};
```

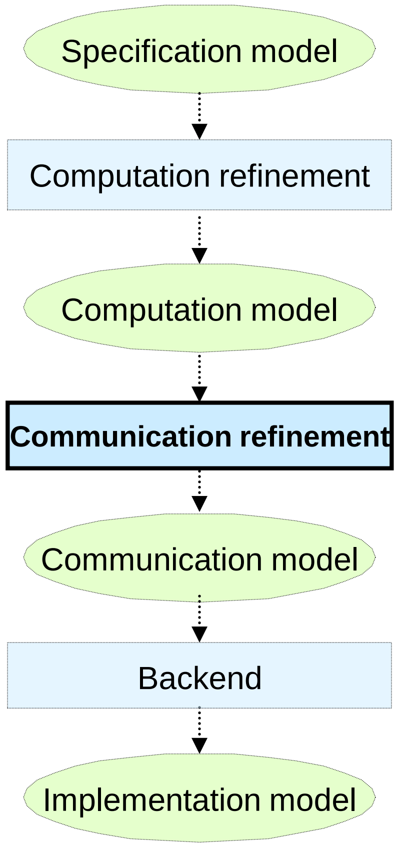
Computation Model

- **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

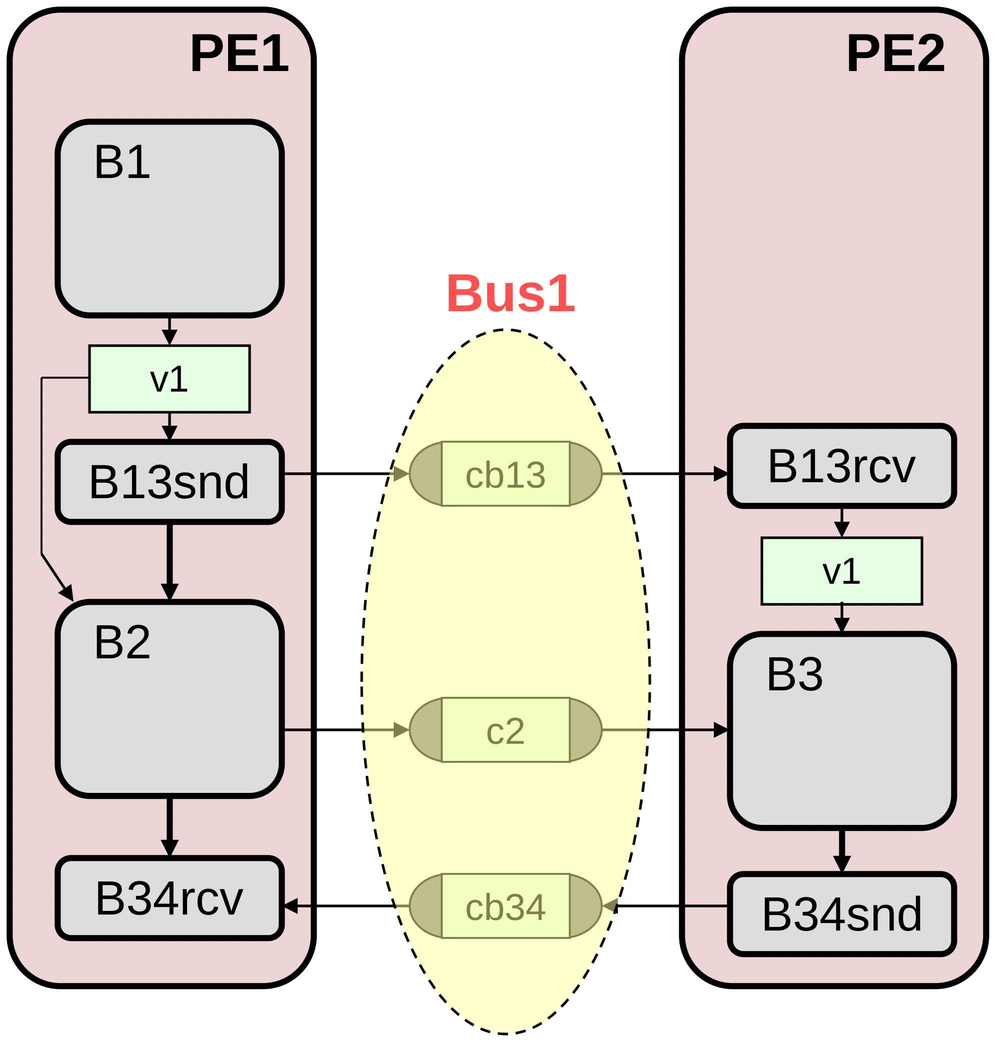


Communication refinement

- **Bus allocation / protocol selection**
- **Channel partitioning**
- **Protocol, transducer insertion**
- **Inlining**



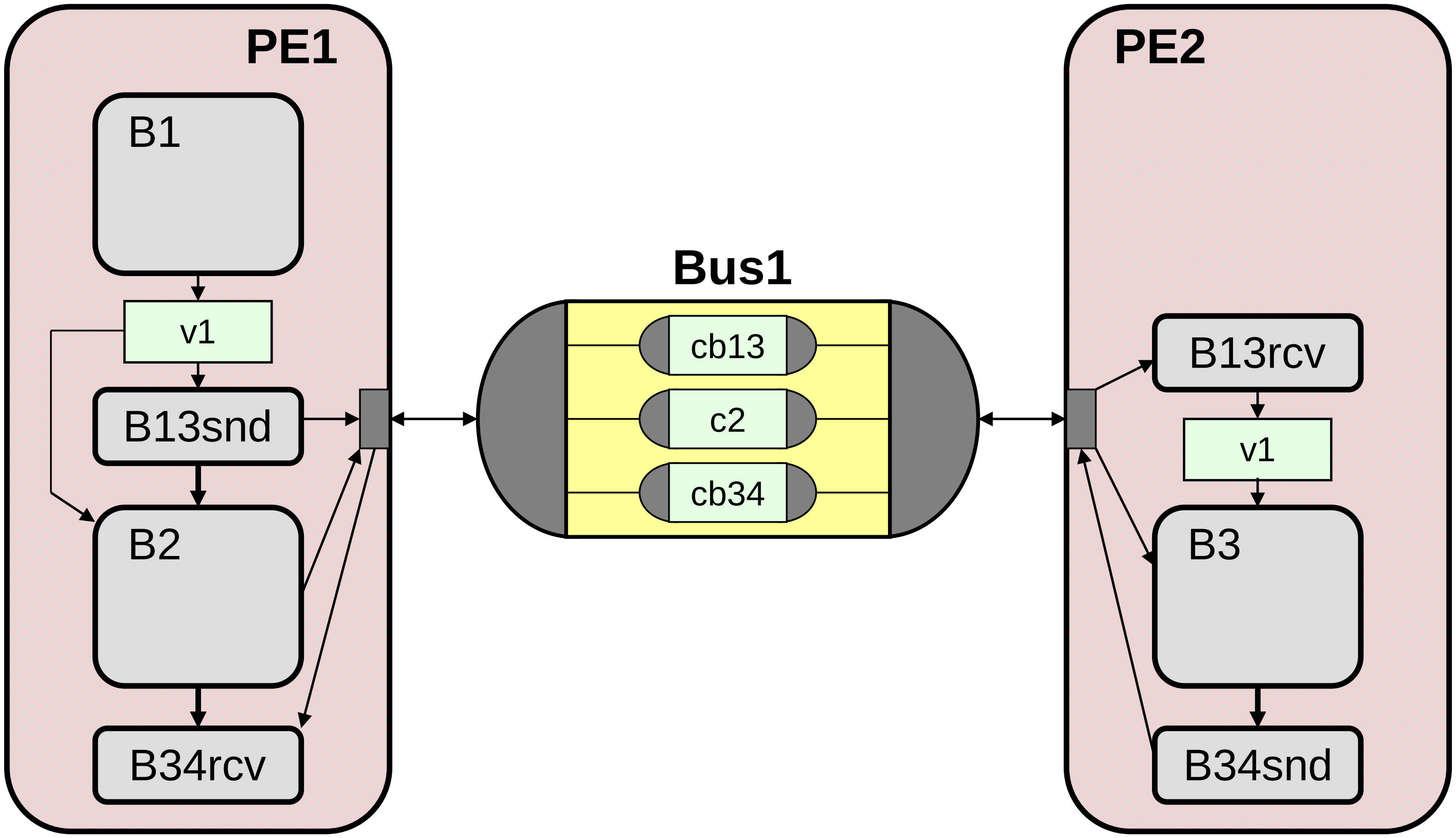
Bus Allocation / Channel Partitioning



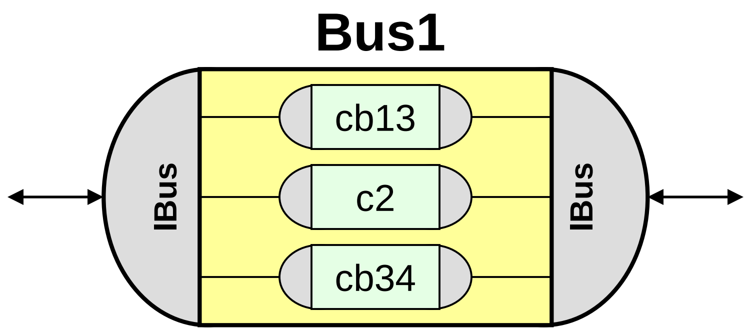
- Allocate busses
- Partition channels
- Update communication

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

Model after Channel Partitioning



SpecC Bus Channel



Virtual addressing:

```
typedef enum {
  CB13, C2, CB34
} BusAddr;
```

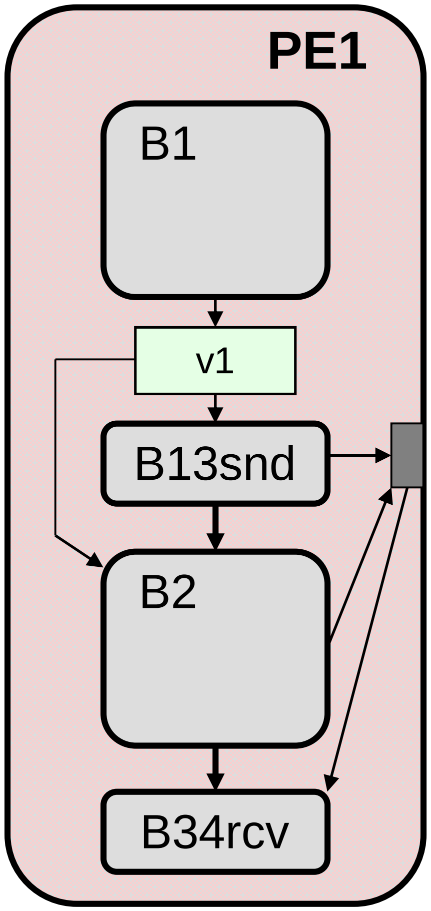
Bus interface:

```
interface IBus
{
  void send( BusAddr a,
             void* d, int size );
  void recv( BusAddr a,
             void* d, int size );
};
```

Hierarchical channel:

```
channel Bus1() implements IBus
{
  ChMP cb13, c2, cb34;

  void send( BusAddr a, void* d, int size )
  {
    switch( a )
    {
      case CB13: return cb13.send( d, size );
      case C2:   return c2.send( d, size );
      case CB34: return cb34.send( d, size );
    }
  }

  void recv( BusAddr a, void* d, int size )
  {
    switch( a )
    {
      case CB13: return cb13.recv( d, size );
      case C2:   return c2.recv( d, size );
      case CB34: return cb34.recv( d, size );
    }
  }
};
```

SpecC Model after Channel Partitioning (PE1)



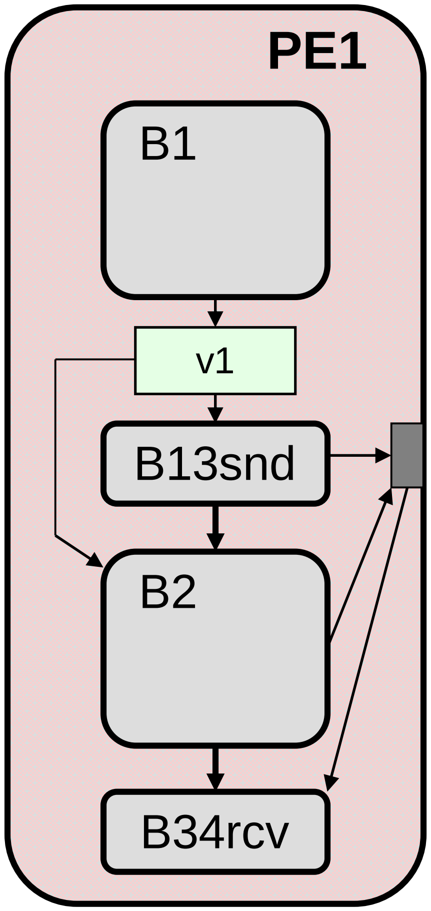
- **Leaf behaviors**

```
behavior B13Snd( in int v1, IBus bus1 )
{
  void main(void) {
    bus1.send( CB13, &v1, sizeof(v1) );
  }
};
```

```
behavior B2( in int v1, IBus bus1 )
{
  void main(void) {
    …
    bus1.send( C2, … );
    …
  }
};
```

```
behavior B34Rcv( IBus bus1 )
{
  void main(void) {
    bus1.recv( CB34, 0, 0 );
  }
};
```

SpecC Model after Channel Partitioning (PE1)



```
behavior PE1( IBus bus1 )
{
  int v1;

  B1     b1     ( v1 );

  B13Snd b13snd( v1, bus1 );

  B2     b2     ( v1, bus1 );

  B34Rcv b34rcv( bus1 );


  void main(void)
  {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};
```

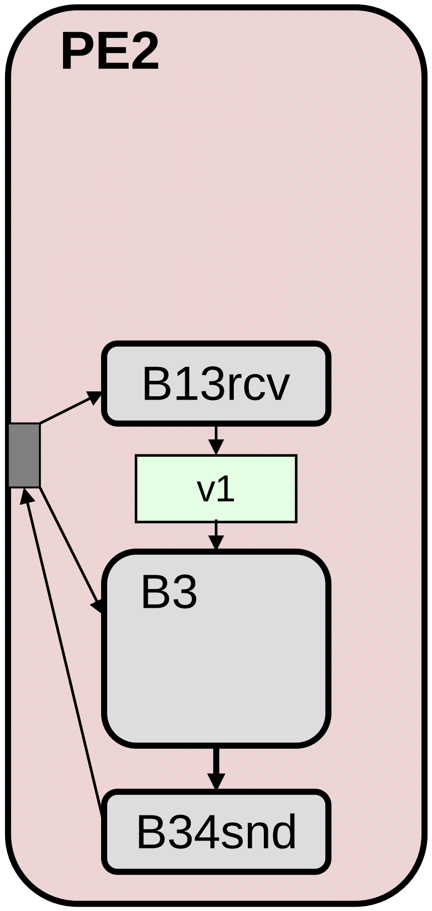
SpecC Model after Channel Partitioning (PE2)

- **Leaf behaviors**

```
behavior B13Rcv( IBus bus1 , out int v1 )
{
  void main(void) {
    bus1.recv( CB13, &v1, sizeof(v1) );
  }
};
```

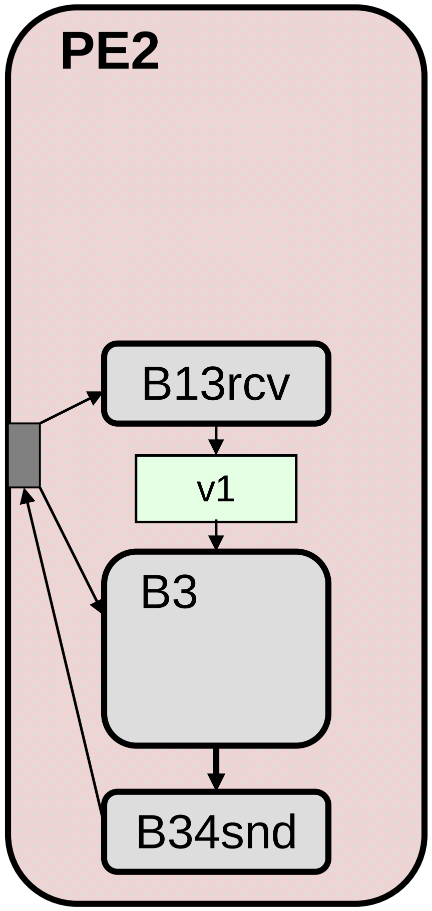
```
behavior B3( in int v1, IBus bus1 )
{
  void main(void) {
    …
    bus1.recv( C2, … );
    …
  }
};
```

```
behavior B34Snd( IBus bus1 )
{
  void main(void) {
    bus1.send( CB34, 0, 0 );
  }
};
```

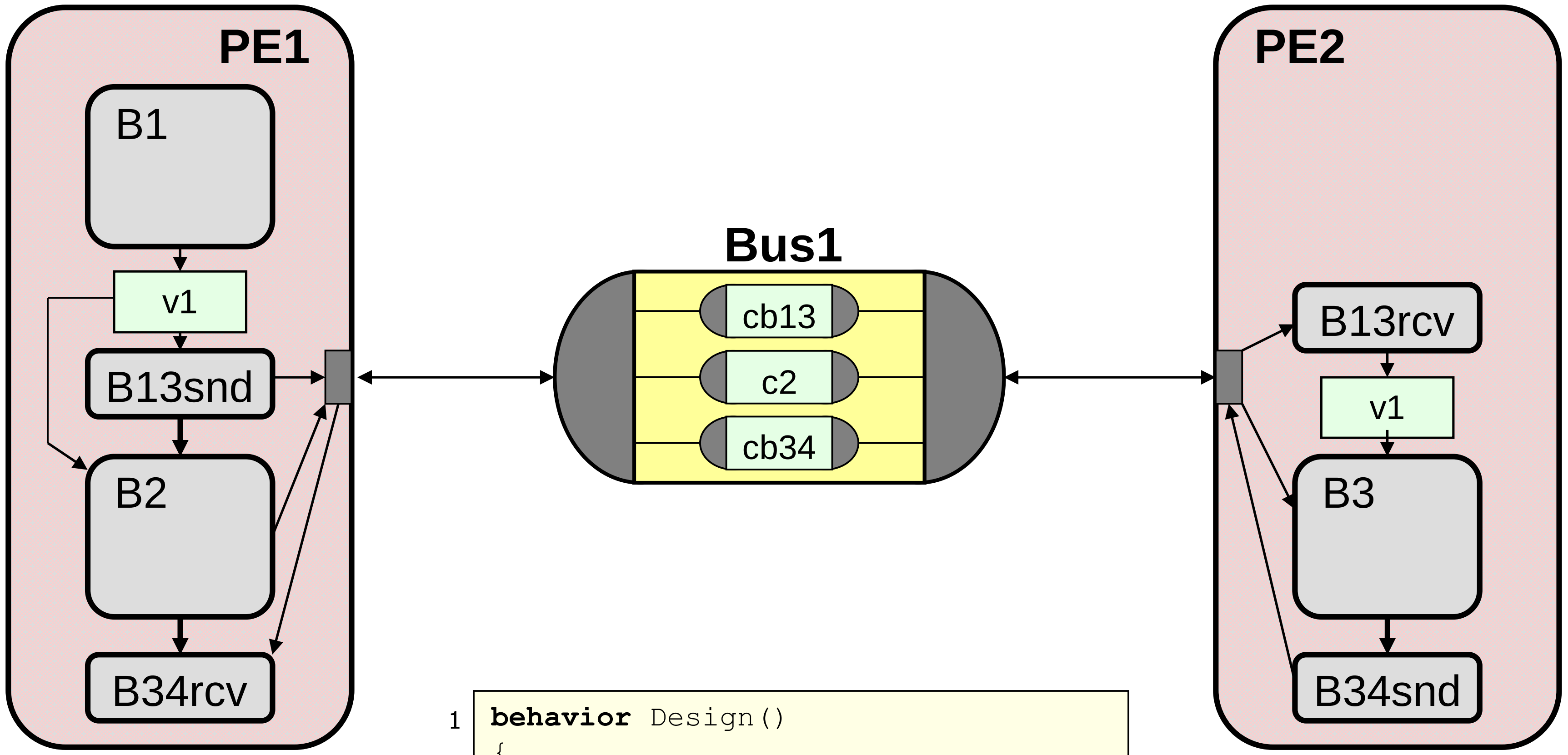


SpecC Model after Channel Partitioning (PE2)

```
behavior PE2( IBus bus1 )
{
  int v1;

  B13Rcv b13rcv( bus1 , v1 );

  B3     b3     ( v1, bus1 );

  B34Snd b34snd( bus1 );


  void main(void)
  {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```

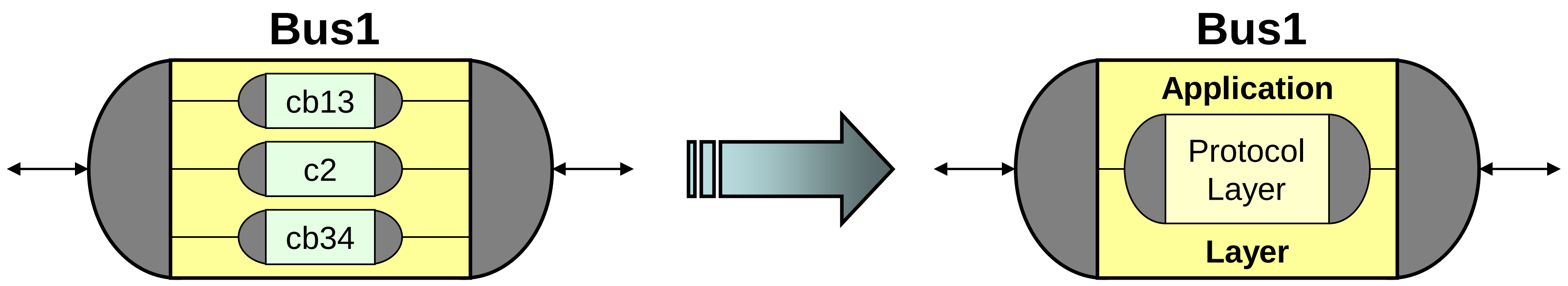


SpecC Model after Channel Partitioning (Top)



```
behavior Design()
{
  Bus1 bus1;

  PE1 pe1( bus1 );
  PE2 pe2( bus1 );

  void main(void) {
    par { pe1.main(); pe2.main(); }
  }
};
```

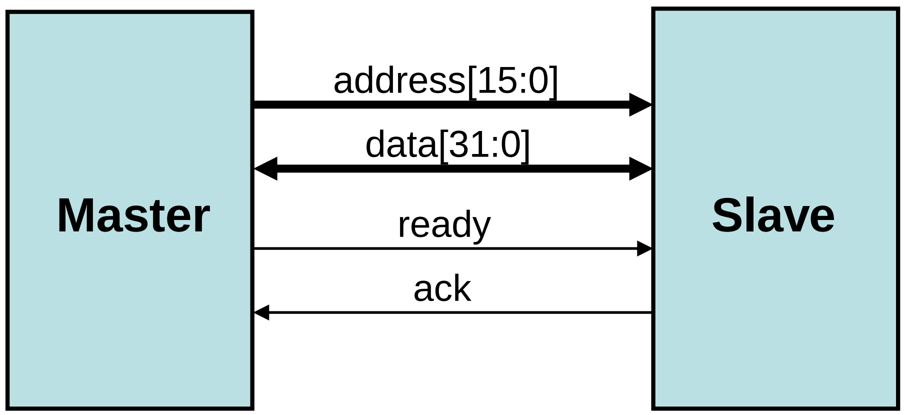
SpecC Protocol Insertion



- **Insert protocol layer**
  - Protocol channel
- **Create application layer**
  - Implement message-passing over bus protocol
- **Replace bus channel**
  - Hierarchical combination of application, protocol layers

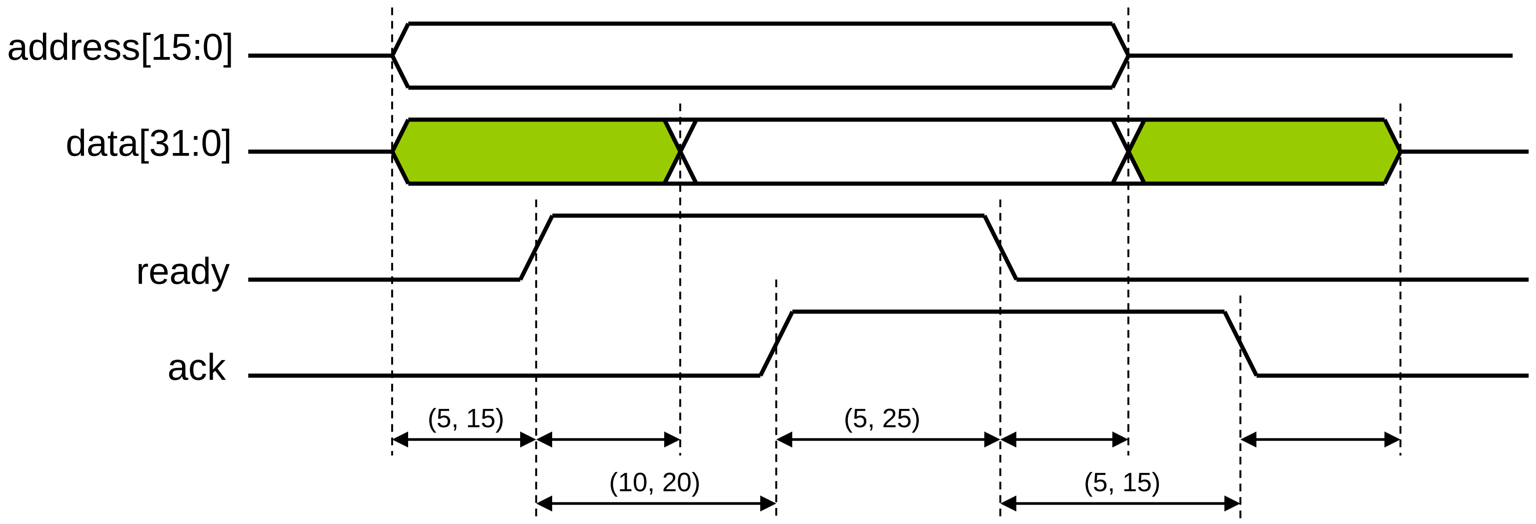
Protocol Example

- Double handshake protocol

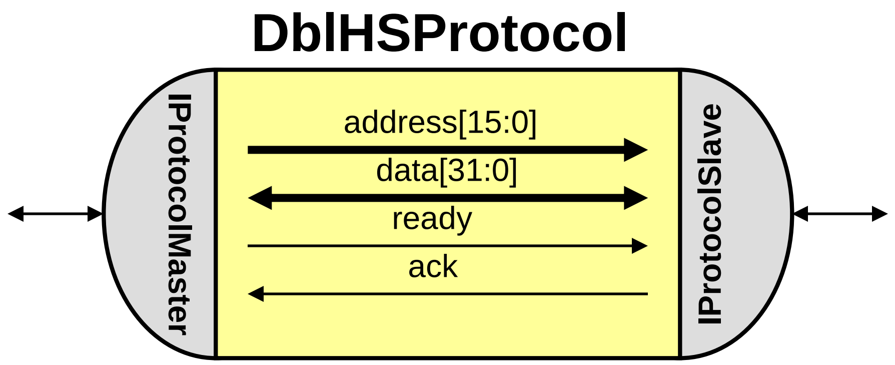


Protocol Example (2)

- **Timing diagram**

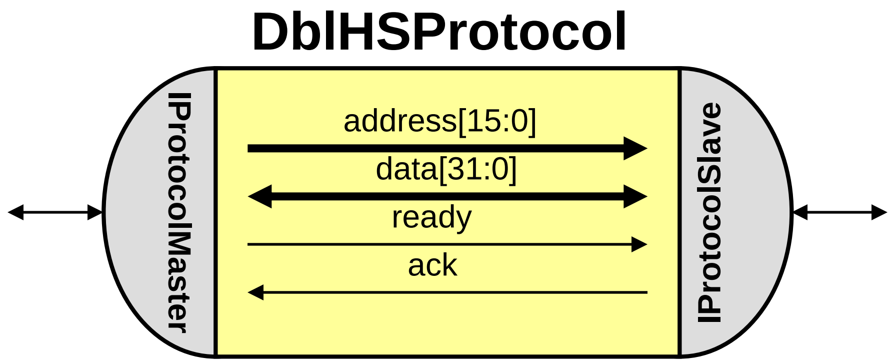


Protocol Layer



- **Protocol channel**
  - Encapsulate wires
  - Abstract bus transfers
  - Implement protocol
  - Bus timing

SpecC Protocol Channel



```
channel DblHSProtocol()
  implements IProtocolMaster,
             IProtocolSlave
{
  bit[15:0] address;
  bit[31:0] data;
  Signal    ready();
  Signal    ack();

  …
};
```

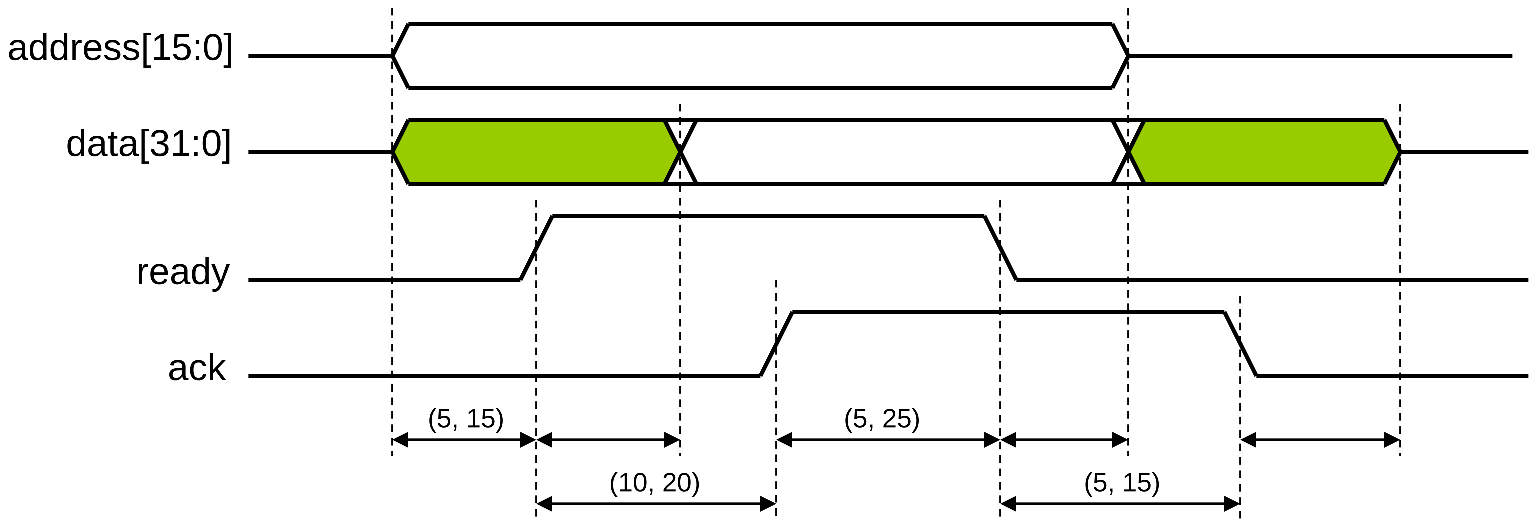
Master interface:

```
interface IProtocolMaster
{
  void masterWrite( bit[15:0] a,
                    bit[31:0] d );
  void masterRead(  bit[15:0] a,
                    bit[31:0] *d );
};
```

Slave interface:

```
interface IProtocolSlave
{
  void slaveWrite( bit[15:0] a,
                   bit[31:0] d );
  void slaveRead(  bit[15:0] a,
                   bit[31:0] *d );
};
```

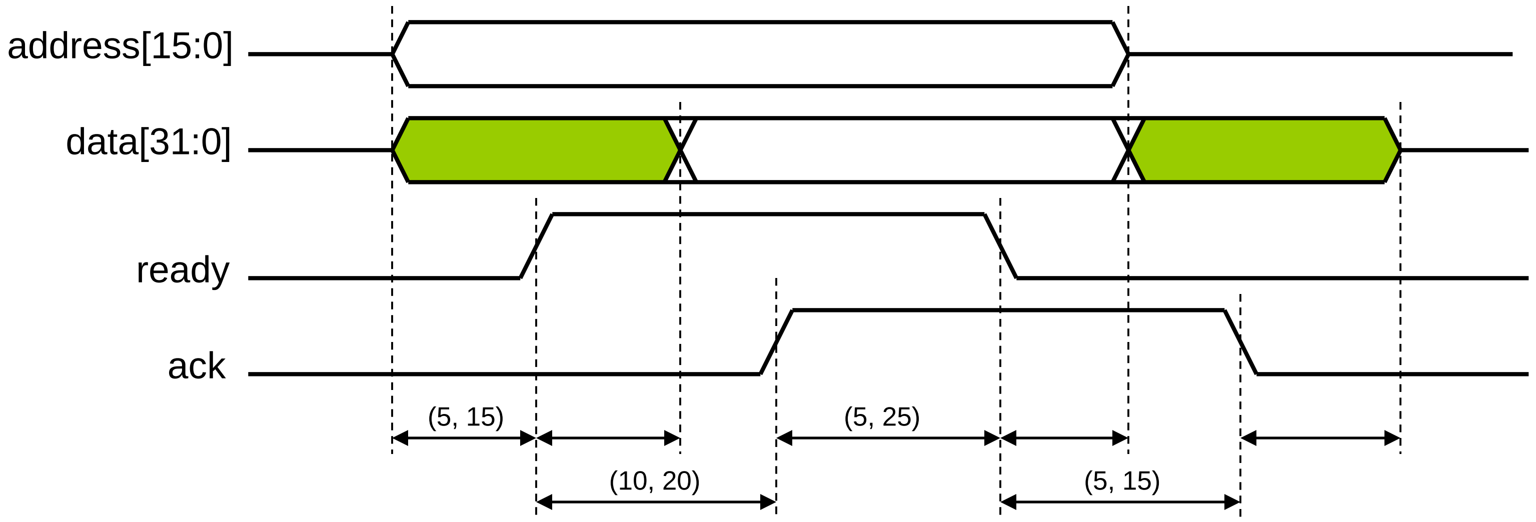
SpecC Protocol Master Interface



```
void masterRead( bit[15:0] a, bit[31:0] *d )
{
  do {
    t1: address = a;
        waitfor( 5 );     // estimated delay
    t2: ready.set( 1 );
        ack.waituntil( 1 );
    t3: *d = data;
        waitfor( 15 );    // estimated delay
    t4: ready.set( 0 );
        ack.waituntil( 0 );
  } timing {              // constraints
    range( t1; t2; 5; 15 );
    range( t3; t4; 10; 25 );
  }
}
```

```
void masterWrite( bit[15:0] a, bit[31:0] d )
{
  do {
    t1: address = a;
        data    = d;
        waitfor( 5 );     // estimated delay
    t2: ready.set( 1 );
        ack.waituntil( 1 );
    t3: waitfor( 10 );    // estimated delay
    t4: ready.set( 0 );
        ack.waituntil( 0 );
  } timing {              // constraints
    range( t1; t2; 5; 15 );
    range( t3; t4; 10; 25 );
  }
}
```

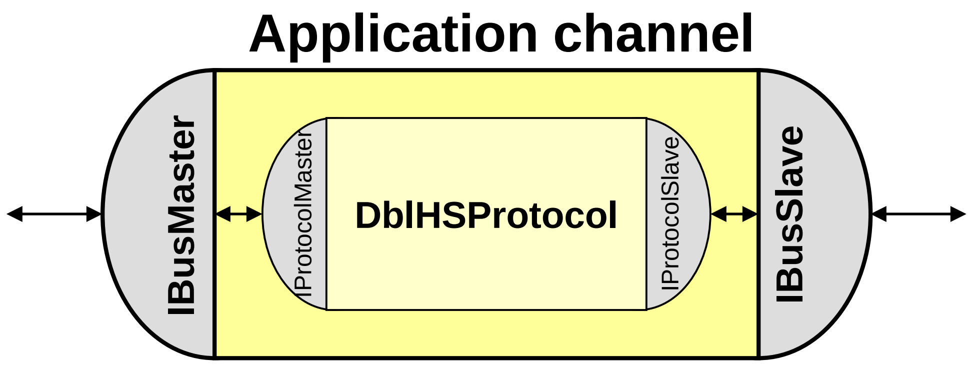
SpecC Protocol Slave Interface



```
void slaveRead( bit[15:0] a, bit[31:0] *d )
{
  do {
    t1: ready.waituntil( 1 );
    t2: if( a != address ) goto t1;
        *d = data;
        waitfor( 12 );    // estimated delay
    t3: ack.set( 1 );
        ready.waituntil( 0 );
    t4: waitfor( 7 );     // estimated delay
    t5: ack.set( 0 );
  } timing {              // constraints
    range( t2; t3; 10; 20 );
    range( t4; t5; 5; 15 );
  }
}
```

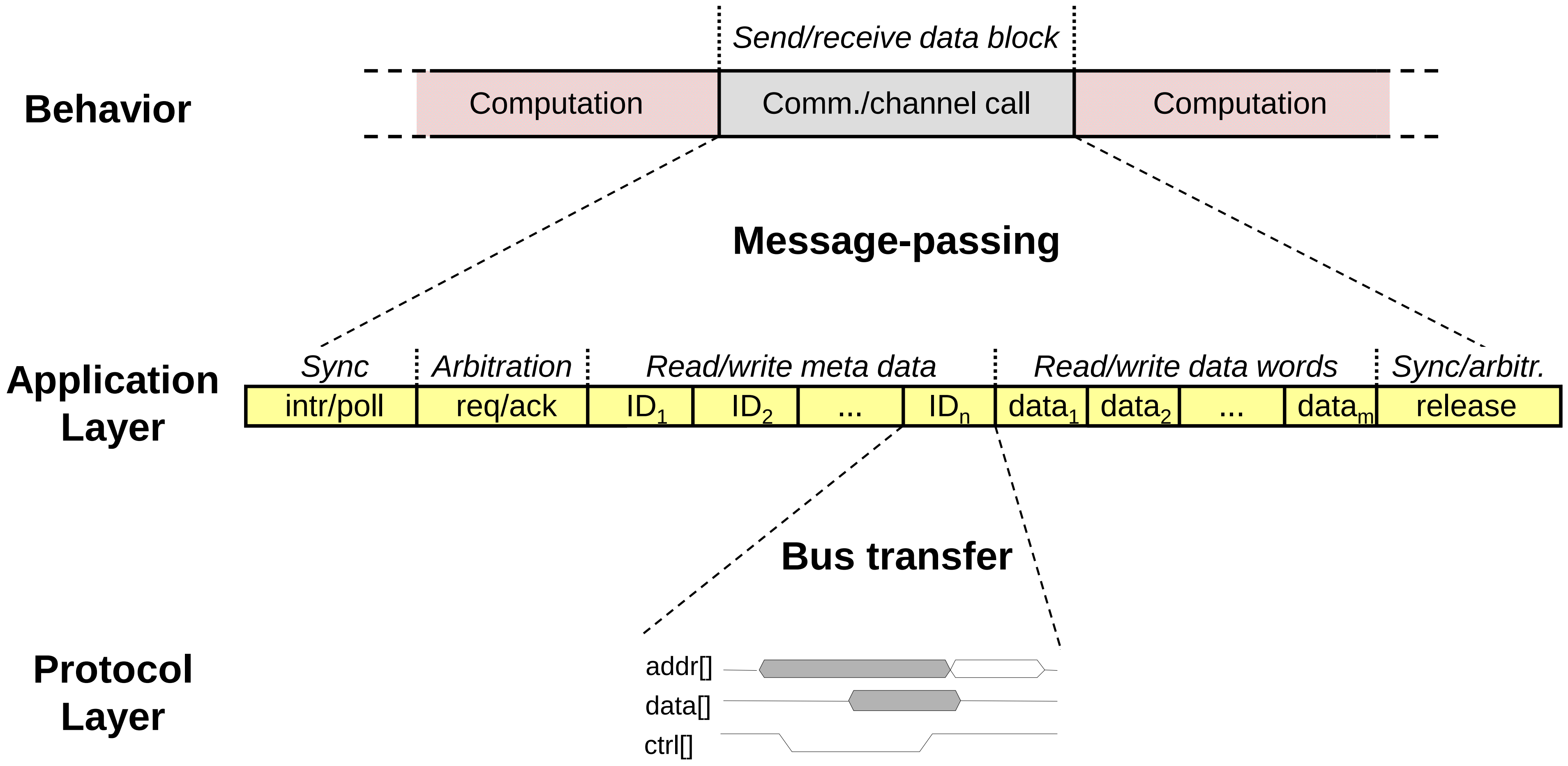
```
void slaveWrite( bit[15:0] a, bit[31:0] d )
{
  do {
    t1: ready.waituntil( 1 );
    t2: if( a != address ) goto t1;
        data = d;
        waitfor( 12 );    // estimated delay
    t3: ack.set( 1 );
        ready.waituntil( 0 );
    t4: waitfor( 7 );     // estimated delay
    t5: ack.set( 0 );
  } timing {              // constraints
    range( t2; t3; 10; 20 );
    range( t4; t5; 5; 15 );
  }
}
```

SpecC Application Layer

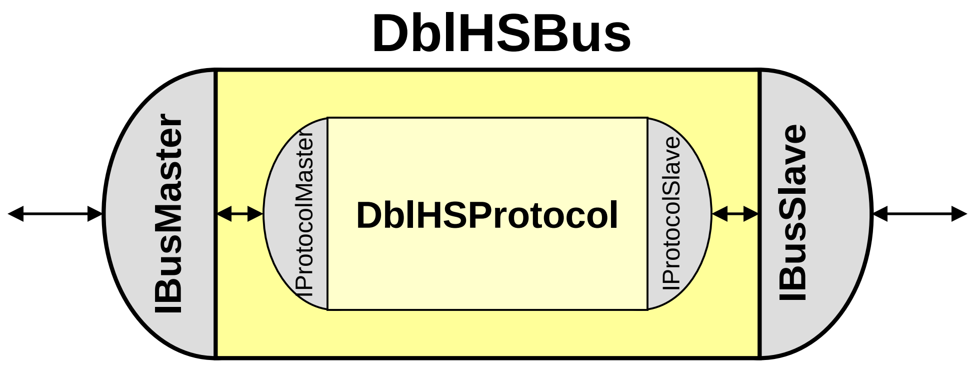


- **Implement abstract message-passing over protocol**
  - Synchronization
  - Arbitration
  - Addressing
  - Data slicing

SpecC Application Layer (2)



SpecC Application Layer Channel



```
channel DblHSBus()
  implements IBusMaster,
             IBusSlave
{
  DblHSProtocol protocol;

  ...
};
```

Master interface:

```
interface IBusMaster
{
  void masterSend( BusAddr a,
                   void* d, int size );
  void masterRecv( BusAddr a,
                   void *d, int size );
};
```

Slave interface:

```
interface IBusSlave
{
  void slaveSend( BusAddr a,
                  void* d, int size );
  void slaveRecv( BusAddr a,
                  void* d, int size );
};
```

Application Layer Methods

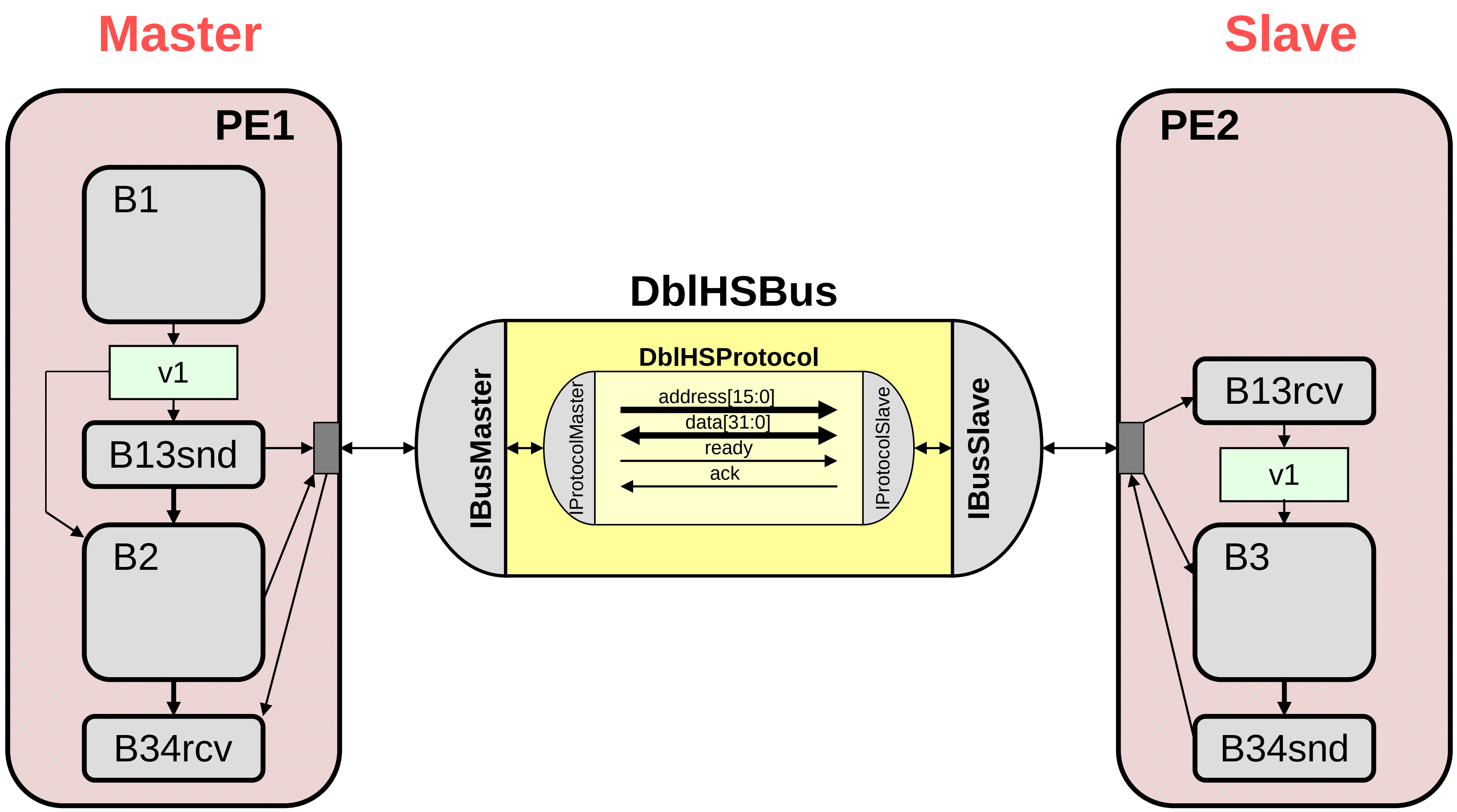
Master interface:

```
void masterSend( int a, void* d, int size )
{
  long *p = d;

  for( ; size > 0; size -= 4, p++ ) {
    protocol.masterWrite( a, *p );
  }
}

void masterRecv( int a, void* d, int size )
{
  long *p = d;

  for( ; size > 0; size -= 4, p++ ) {
    protocol.masterRead( a, p );
  }
}
```

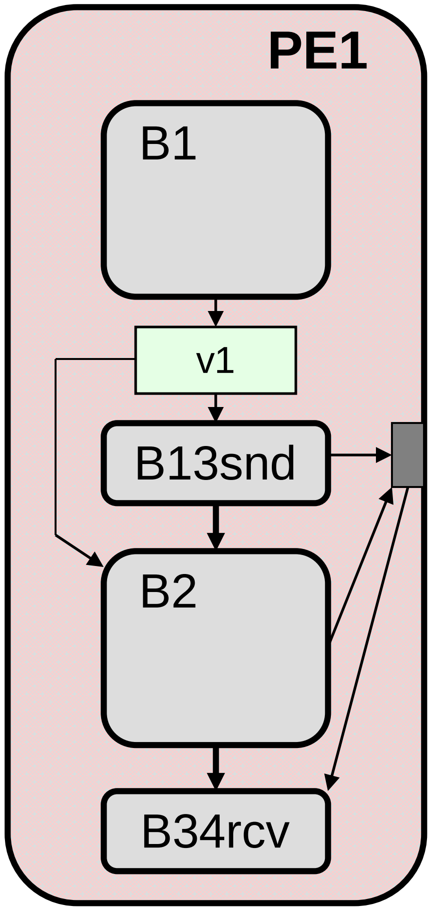
Slave interface:

```
void slaveSend( int a, void* d, int size )
{
  long *p = d;

  for( ; size > 0; size -= 4, p++ ) {
    protocol.slaveWrite( a, *p );
  }
}

void slaveRecv( int a, void* d, int size )
{
  long *p = d;

  for( ; size > 0; size -= 4, p++ ) {
    protocol.slaveRead( a, p );
  }
}
```

SpecC Model after Protocol Insertion



SpecC Model after Protocol Insertion (PE1)



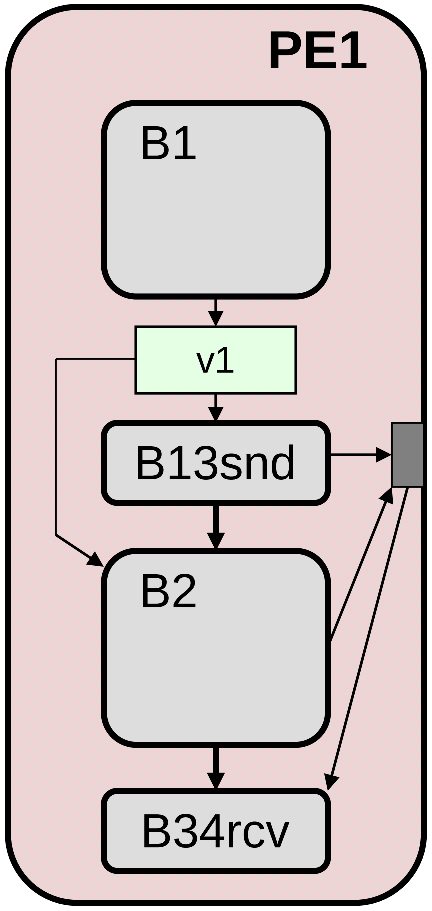
- **Leaf behaviors**

```
behavior B13Snd( in int v1, IBusMaster bus1 )
{
  void main(void) {
     bus1.masterSend( CB13, &v1, sizeof(v1) );
  }
};
```

```
behavior B2( in int v1, IBusMaster bus1 )
{
  void main(void) {
     …
     bus1.masterSend( C2, … );
     …
  }
};
```

```
behavior B34Rcv( IBusMaster bus1 )
{
  void main(void) {
     bus1.masterRecv( CB34, 0, 0 );
  }
};
```

SpecC Model after Protocol Insertion (PE1)



```
behavior PE1( IBusMaster bus1 )
{
  int v1;

  B1     b1     ( v1 );
  B13Snd b13snd( v1, bus1 );
  B2     b2     ( v1, bus1 );
  B34Rcv b34rcv( bus1 );

  void main(void)
  {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};
```

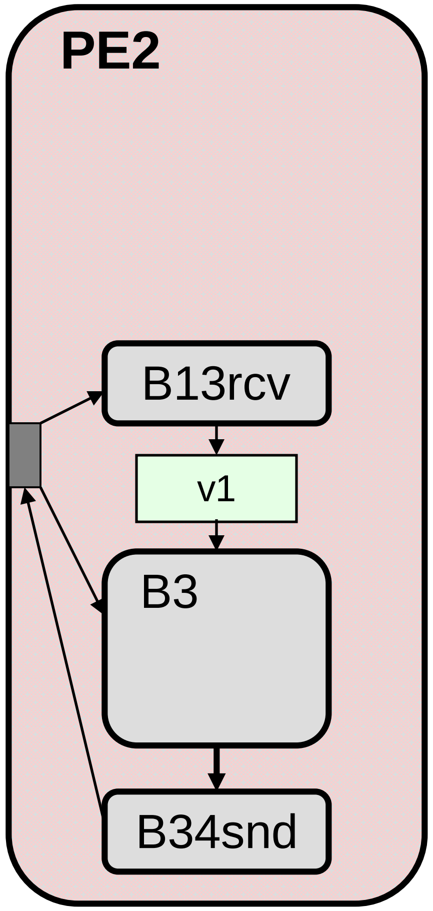
SpecC Model after Protocol Insertion (PE2)

- **Leaf behaviors**

```
behavior B13Rcv( IBusSlave bus1 , out int v1 )
{
  void main(void) {
    bus1.slaveRecv( CB13, &v1, sizeof(v1) );
  }
};
```

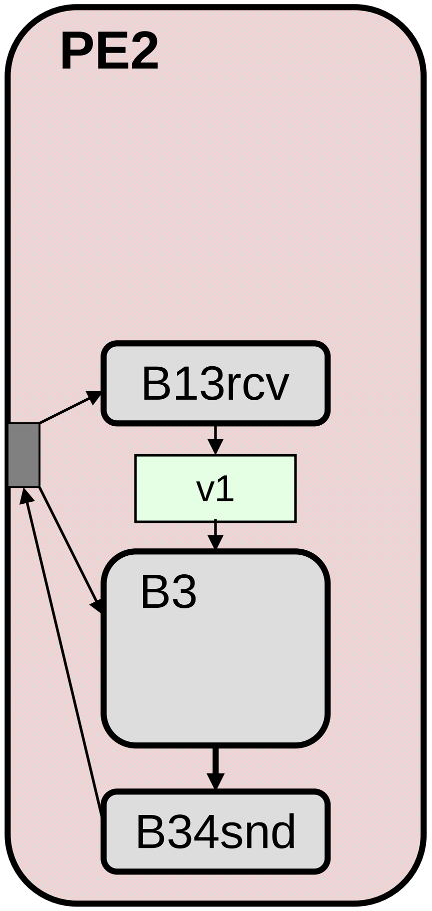
```
behavior B3( in int v1, IBusSlave bus1 )
{
  void main(void) {
    …
    bus1.slaveRecv( C2, … );
    …
  }
};
```

```
behavior B34Snd( IBusSlave bus1 )
{
  void main(void) {
    bus1.slaveSend( CB34, 0, 0 );
  }
};
```

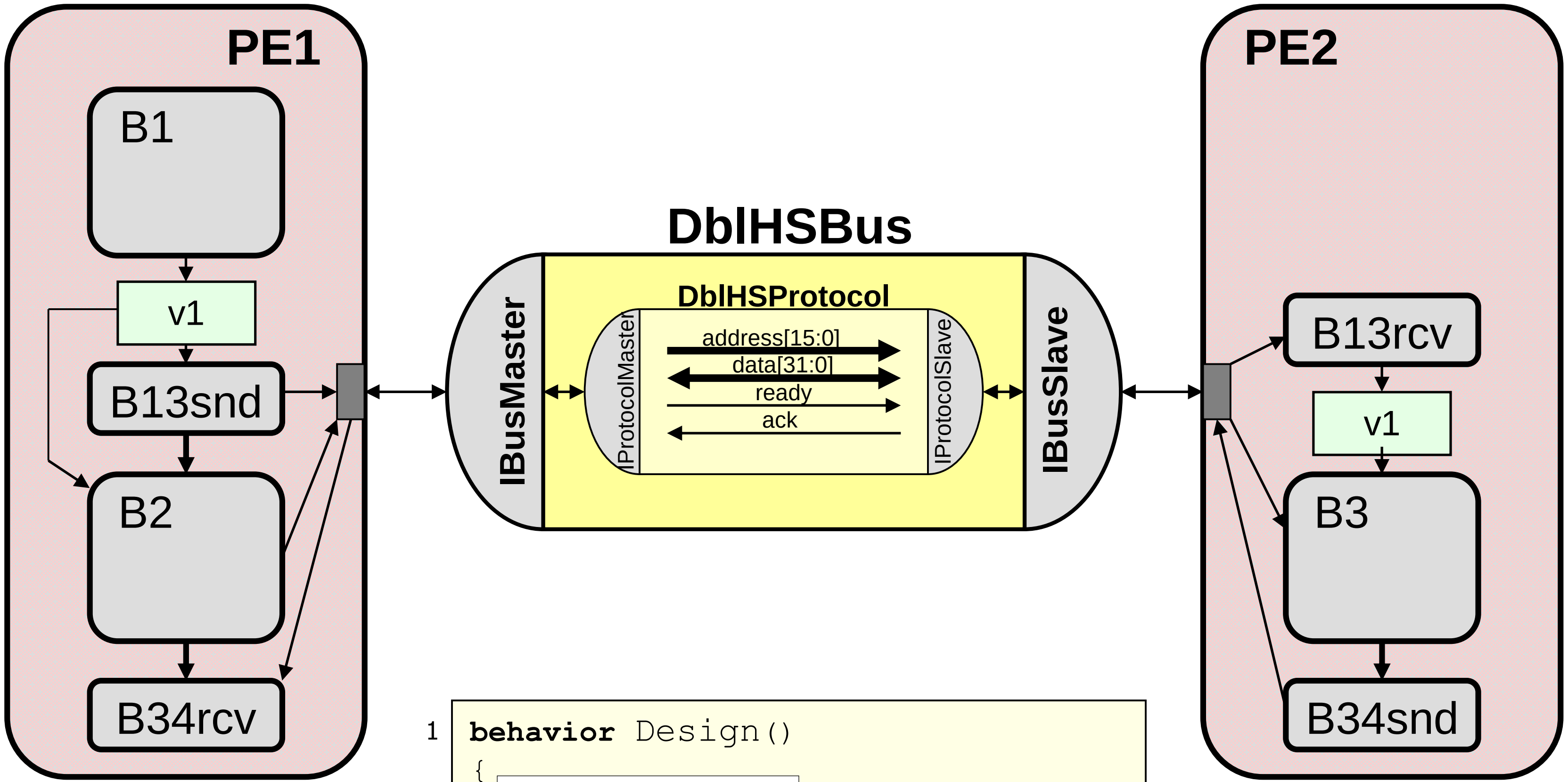


SpecC Model after Protocol Insertion (PE2)

```
behavior PE2( IBusSlave bus1 )
{
  int v1;

  B13Rcv b13rcv( bus1 , v1 );
  B3     b3     ( v1, bus1 );
  B34Snd b34snd( bus1 );


  void main(void)
  {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```

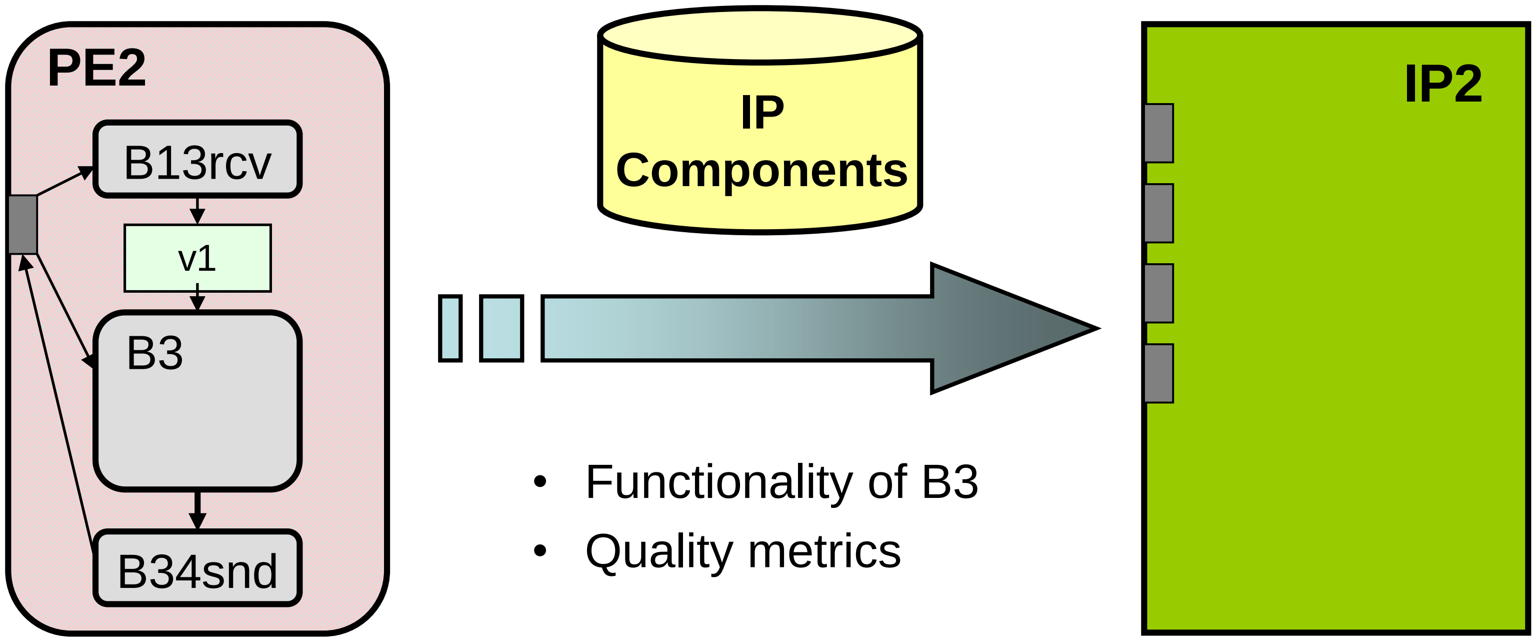


SpecC Model after Protocol Insertion (Top)



```
behavior Design()
{
  DblHSBus bus1;

  PE1 pe1( bus1 );
  PE2 pe2( bus1 );

  void main(void) {
    par { pe1.main(); pe2.main(); }
  }
};
```

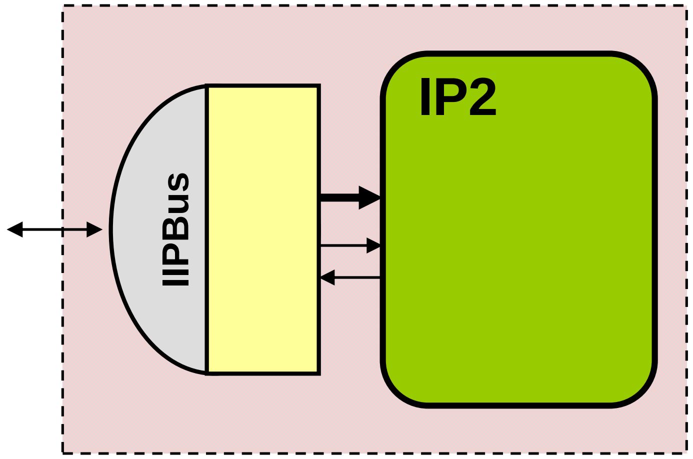
Intellectual Property (IP)



- Functionality of B3
- Quality metrics

- **IP component library**
  - Pre-designed components
    - Fixed functionality
    - Fixed interface / protocols
  - Allocate IP components
    - Implement functionality through IP reuse

IP Component Model

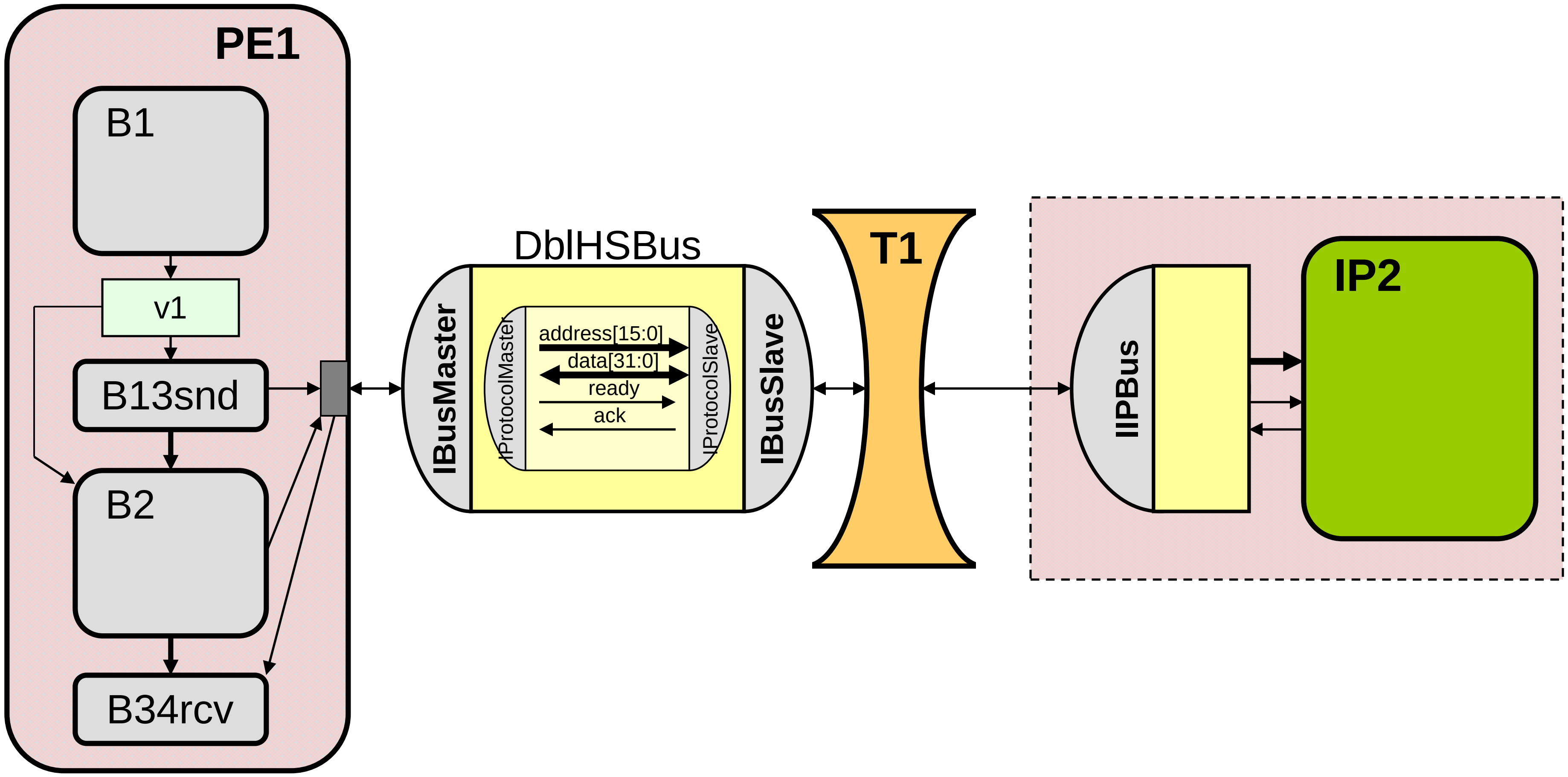


- **Component behavior**
  - Simulation, synthesis

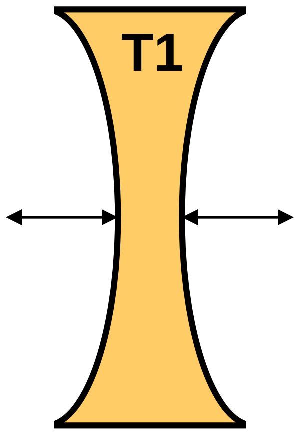
- **Wrapper**
  - Encapsulate fixed IP protocol
  - Provide canonical interface

```
interface IIPBus
{
  void start( int v1, … );
  void done( void );
};
```

Transducer Insertion



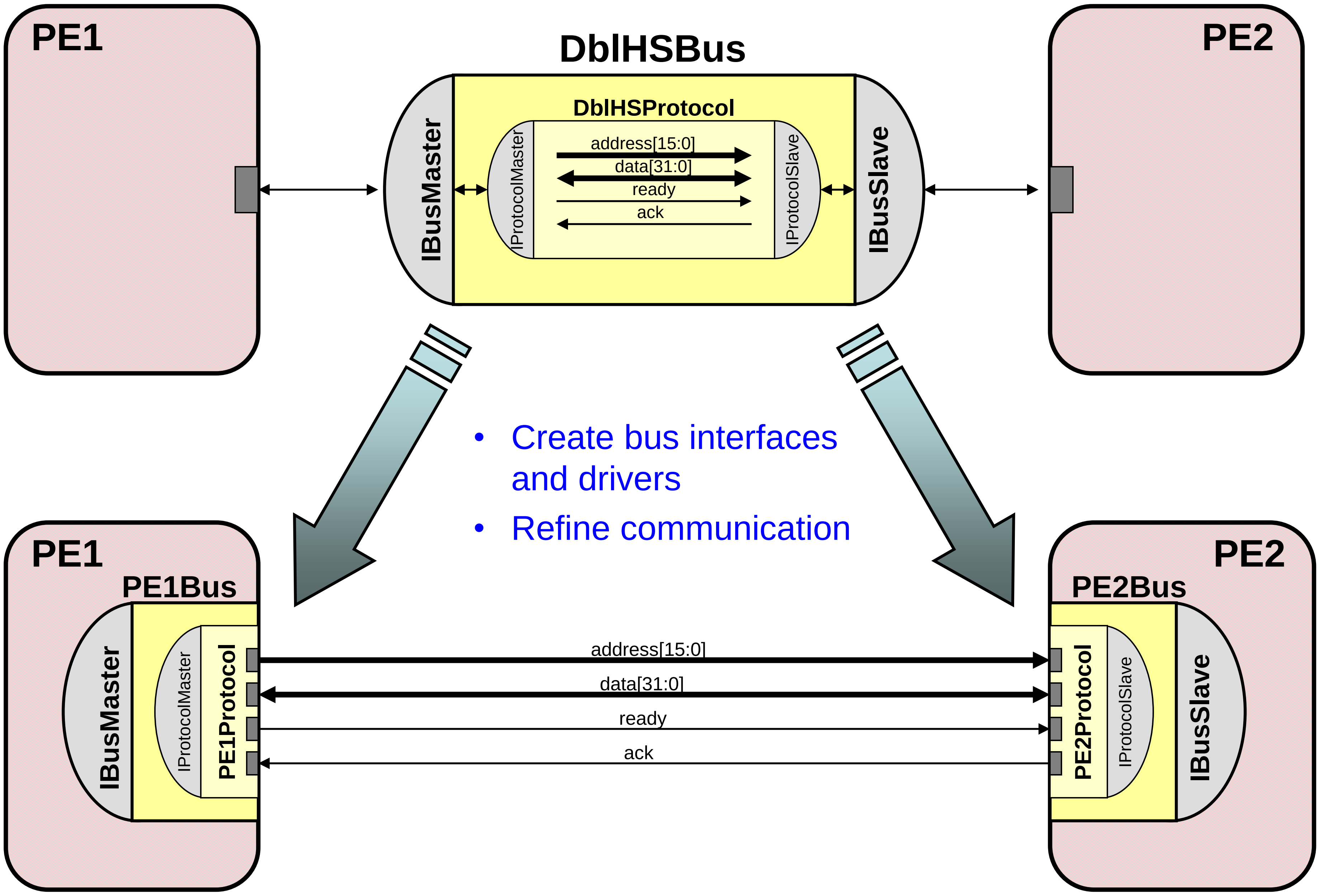
Transducer



```
behavior T1( IBusSlave bus,
             IIPbus ip )
{
  int v1, … ;

  void main(void)
  {
    bus.slaveReceive( CB13, &v1, sizeof(v1) );
    bus.slaveReceive( C2, … );
    ip.start( v1, … );
    ip.done();
    bus.slaveSend( CB34, 0, 0 );
  }
};
```

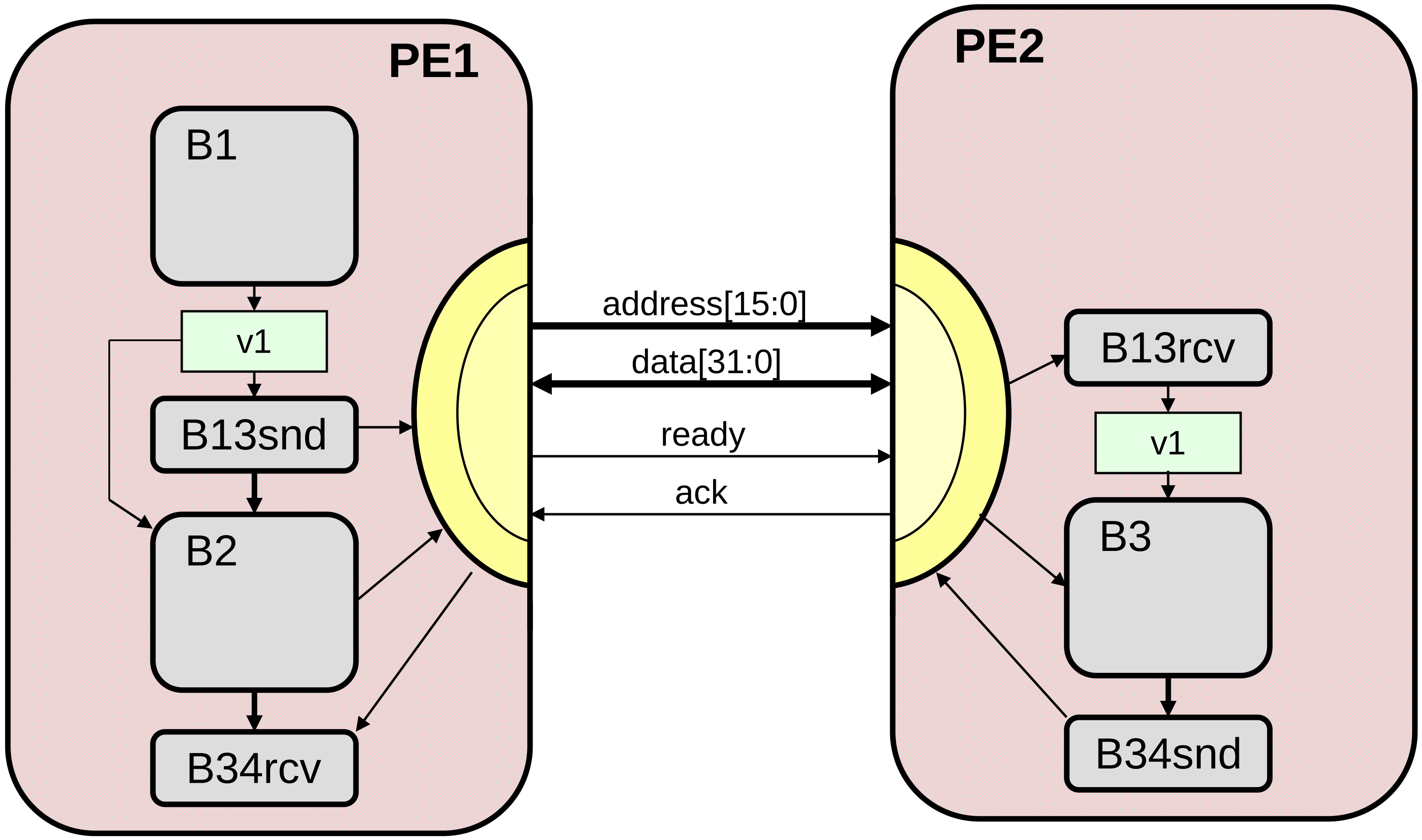
- **Translate between protocols**
  - Send / receive messages
  - Buffer in local memory

Inlining



- Create bus interfaces and drivers
- Refine communication

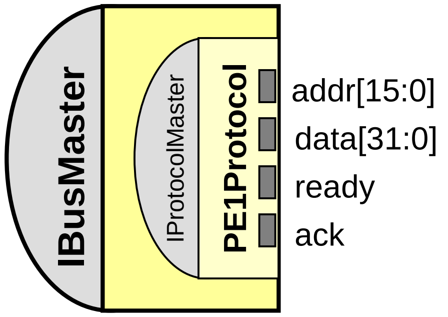
Model after Inlining



SpecC Model after Inlining (PE1)

- **PE1 bus driver**

PE1Bus



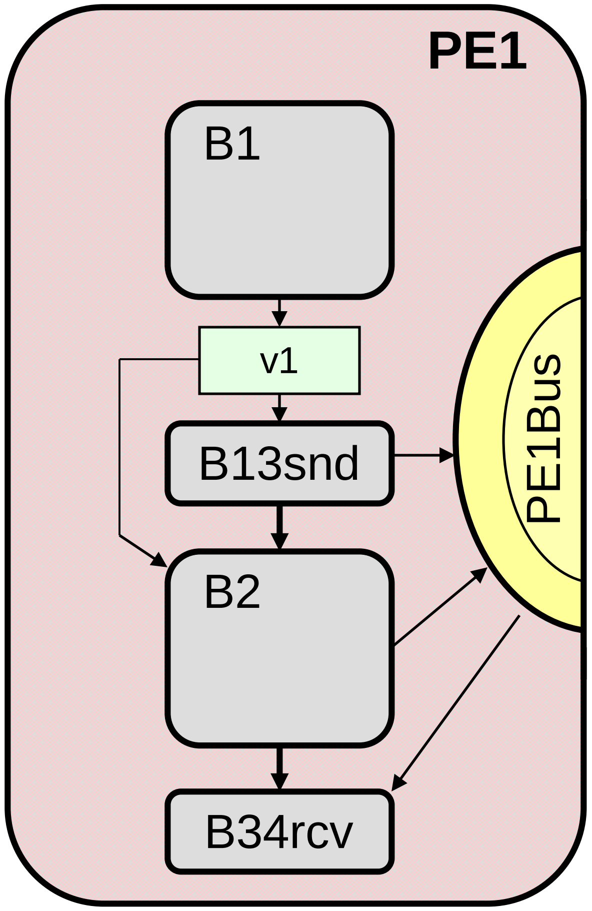
Application layer:

```
channel PE1Bus( out   bit[15:0] addr.
                inout bit[31:0] data,
                OSignal         ready,
                ISignal         ack )
  implements IBusMaster
{
  PE1Protocol protocol( addr, data, ready, ack );

  void masterSend( int a, void* d, int size ) { … }
  void masterRecv( int a, void* d, int size ) { … }
};
```

Protocol layer:

```
channel PE1Protocol( out   bit[15:0] addr.
                     inout bit[31:0] data,
                     OSignal         ready,
                     ISignal         ack )
  implements IProtocolMaster
{
  void masterWrite( bit[15:0] a, bit[31:0] d ) { … }
  void masterRead( bit[15:0] a, bit[31:0] *d ) { … }
};
```

SpecC Model after Inlining (PE1)



```
behavior PE1( out   bit[15:0] addr,
              inout bit[31:0] data,
              OSignal         ready,
              ISignal         ack     )
{
  PE1Bus bus1( addr, data, ready, ack );

  int v1;

  B1     b1     ( v1 );
  B13Snd b13snd( v1, bus1 );
  B2     b2     ( v1, bus1 );
  B34Rcv b34rcv( bus1 );

  void main(void) {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};
```

SpecC Model after Inlining (PE2)

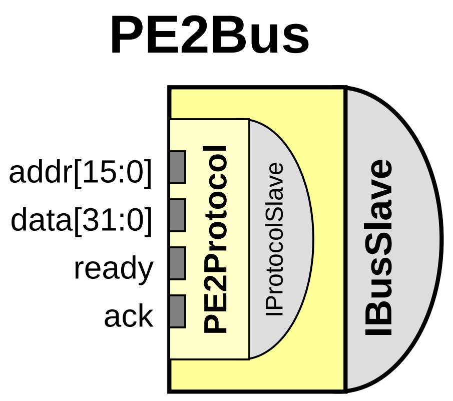
- **PE2 bus interface**

Application layer:

```
channel PE2Bus( in    bit[15:0] addr.
                inout bit[31:0] data,
                ISignal         ready,
                OSignal         ack )
  implements IBusSlave
{
  PE2BusInterface protocol( addr, data, ready, ack );

  void slaveSend( int a, void* d, int size ) { … }
  void slaveRecv( int a, void* d, int size ) { … }
};
```

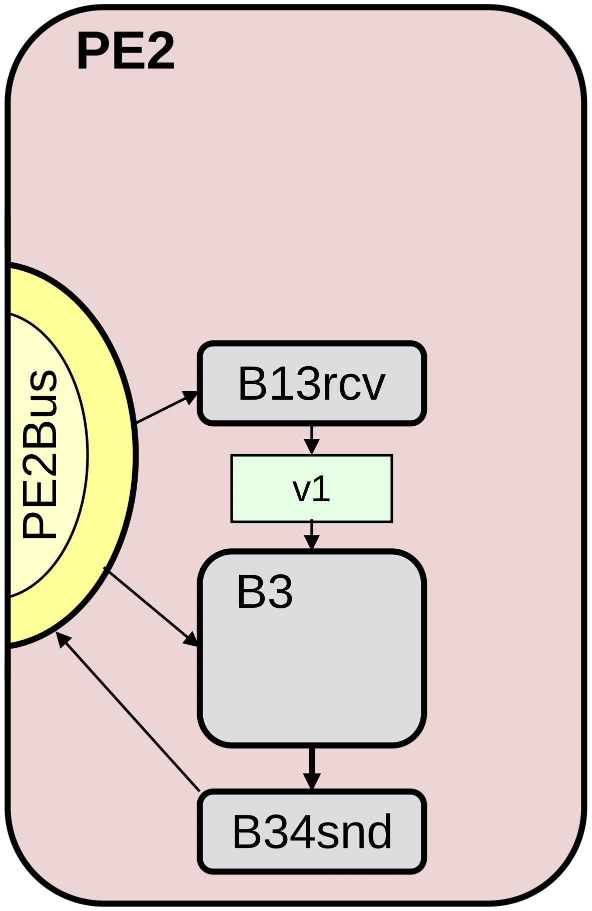
Protocol layer:

```
channel PE2Protocol( in    bit[15:0] addr.
                     inout bit[31:0] data,
                     ISignal         ready,
                     OSignal         ack )
  implements IProtocolSlave
{
  void slaveWrite( bit[15:0] a, bit[31:0] d ) { … }
  void slaveRead( bit[15:0] a, bit[31:0] *d ) { … }
};
```

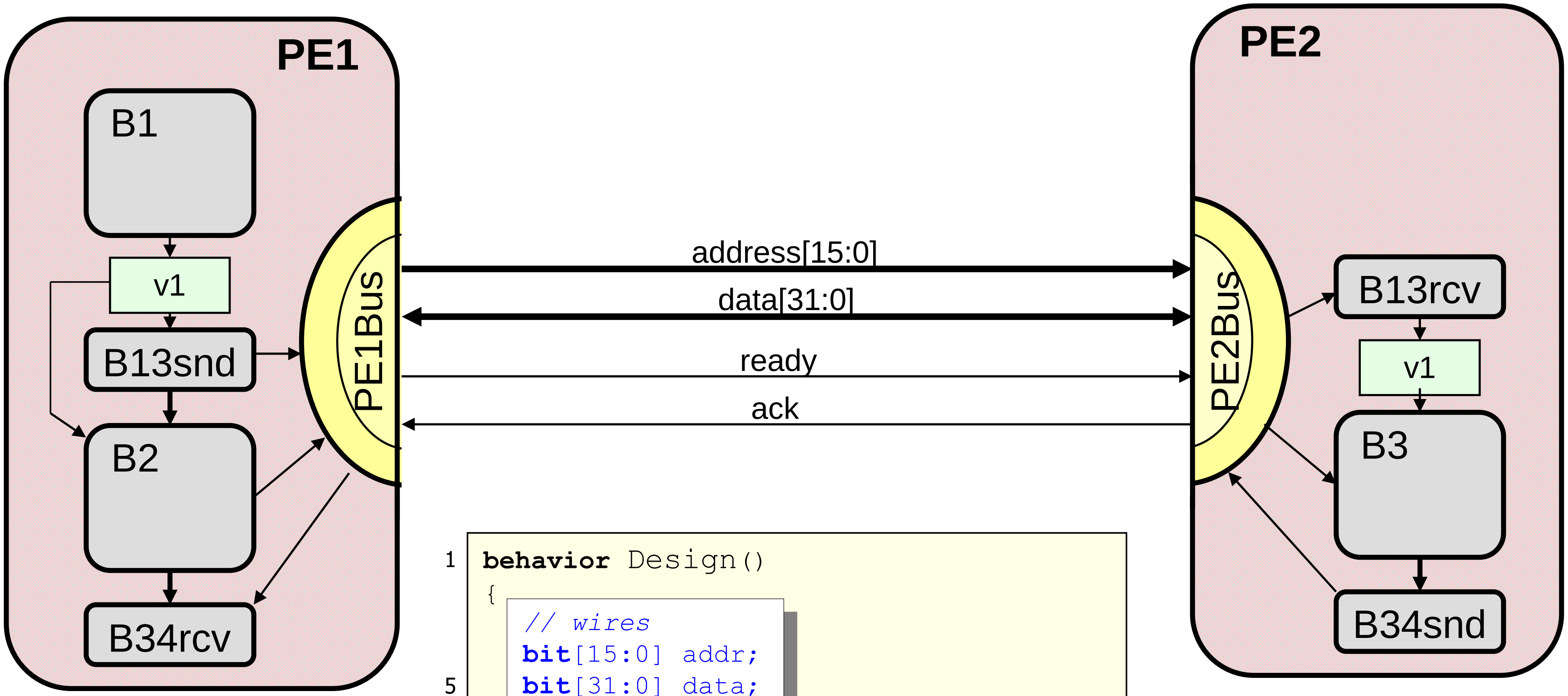


SpecC Model after Inlining (PE2)

```
behavior PE2( in    bit[15:0] addr,
              inout bit[31:0] data,
              ISignal         ready,
              OSignal         ack       )
{
 PE2Bus bus1( addr, data, ready, ack );

  int v1;

  B13Rcv b13rcv( bus1, v1 );
  B3     b3    ( v1, bus1 );
  B34Snd b34snd( bus1 );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```



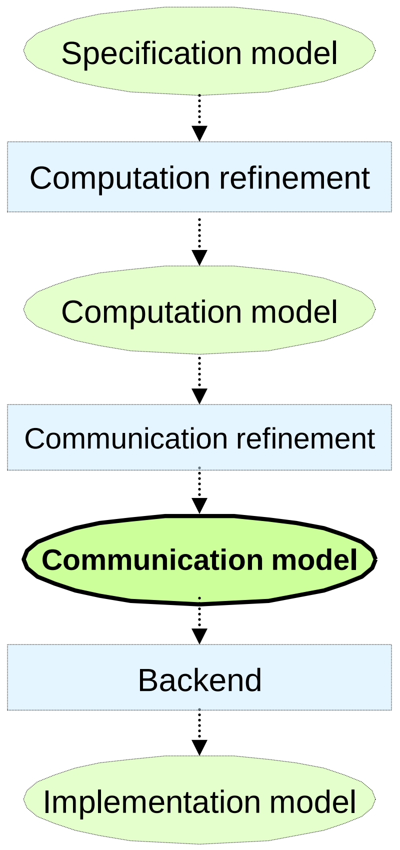
SpecC Model after Inlining (Top)



```
behavior Design()
{
  // wires
  bit[15:0] addr;
  bit[31:0] data;
  Signal ready;
  Signal ack;

  PE1 pe1( addr, data, ready, ack );
  PE2 pe2( addr, data, ready, ack );

  void main(void) {
    par { pe1.main(); pe2.main(); }
  }
};
```

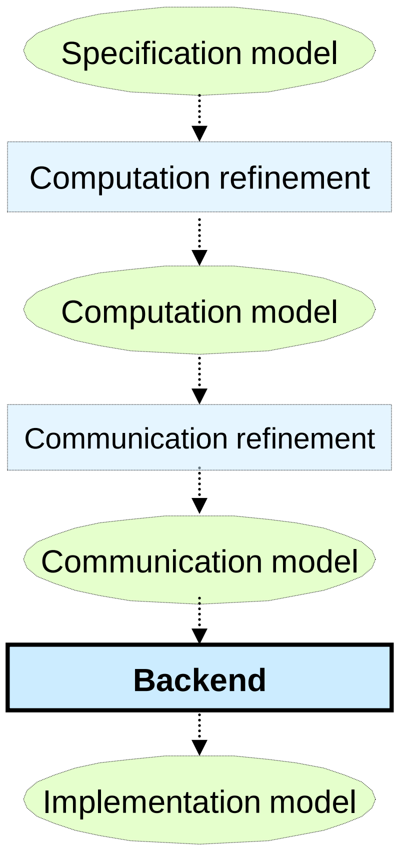
Communication Model

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

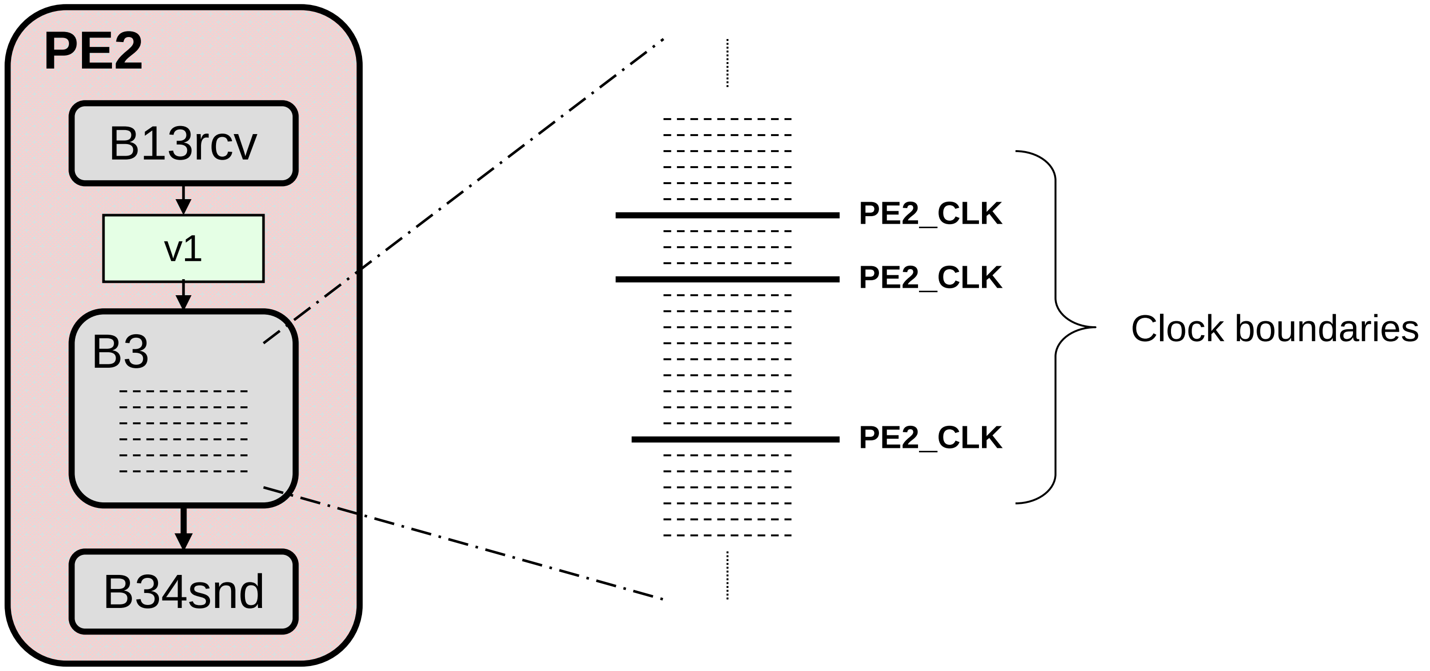


Backend

- **Clock-accurate implementation of PEs**
  - Hardware synthesis
  - Software development
  - Interface synthesis



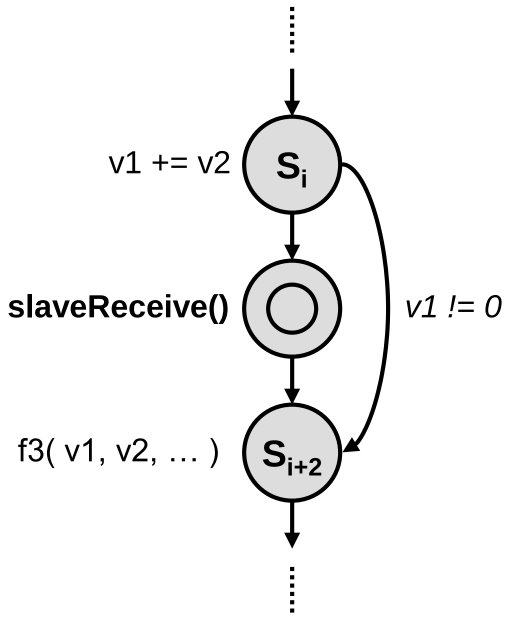
Hardware Synthesis



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

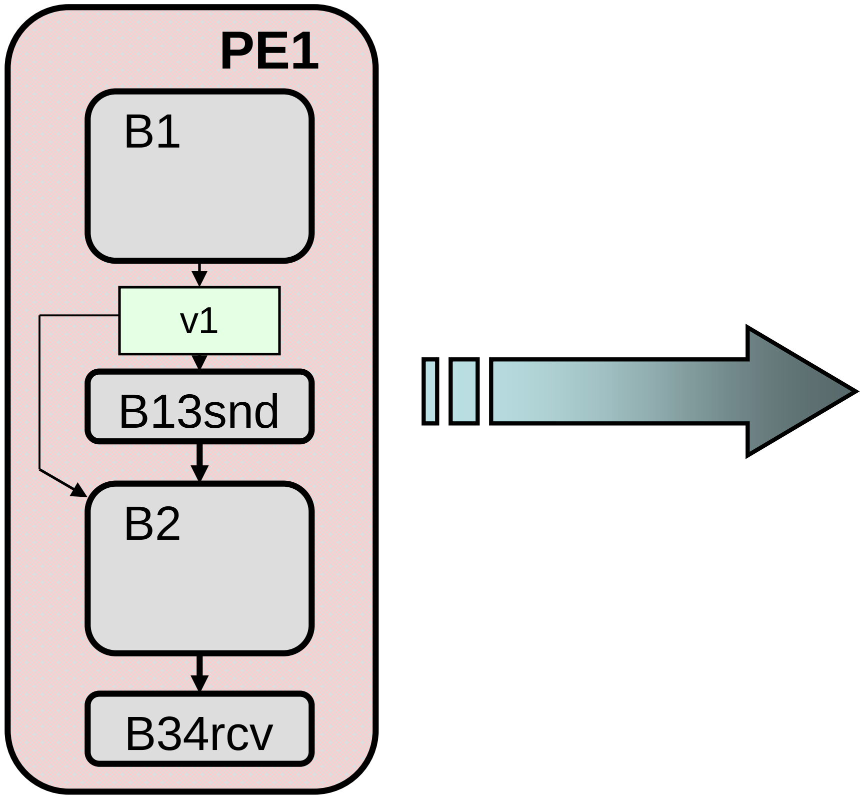
SpecC Scheduled Hardware Model

Hierarchical FSMD:



```
behavior B3( in int v1, IBusSlave bus )
{
  void main(void) {
    enum { S0, S1, S2, …, Sn } state = S0;

    while( state != Sn )
    {
      waitfor( PE2_CLK );   // wait for clock period

      switch( state ) {
        …
        case Si:
          v1 += v2;   // data-path operations
          if( v1 )
            state = Si+1;
          else
            state = Si+2;
          break;
        case Si+1:  // receive message
          bus.slaveReceive( C2, … );
          state = Si+2;
          break;
        case Si+2:
          f3( v1, v2, … );
          state = Si+3;
          break;
        …
    }}
  }
};
```

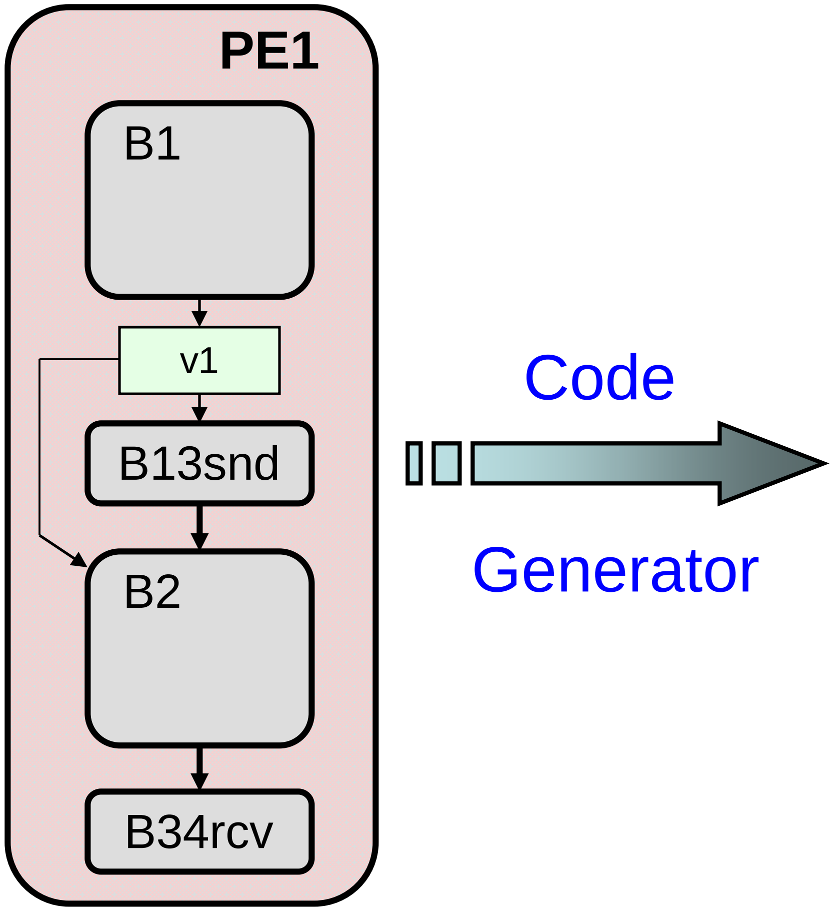
Software Synthesis



```
        ⋮
Ff2
  MOVE     r0, r1
        ⋮
  SHL      r3
  ADD      r2, r3, r4
  INC      r2
        ⋮
  PUSH     r1
  CALL     Ff3
  POP      r0
        ⋮
```

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

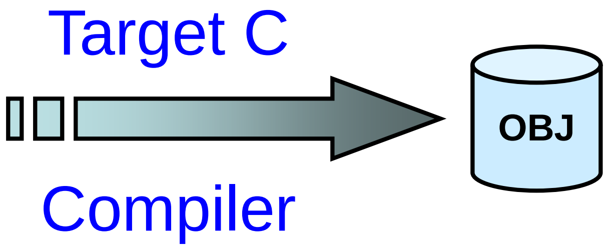
Code Generation



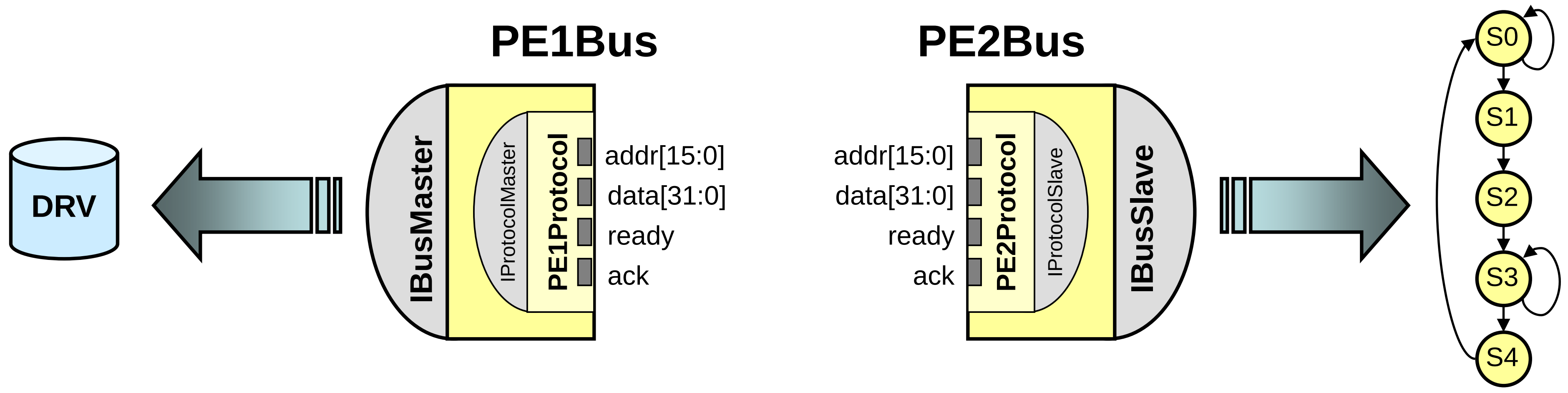
```
void B1( int *v1 ) {
  …
}

void B13Snd( int v1 ) {
  masterSend( CB13, &v1, sizeof(v1) );
}

void B2( int v1 ) {
  …
  masterSend( C2, );
  …
}

void B34Rcv( void ) {
  masterReceive( CB34, 0, 0 );
}

void main(void)
{
  int v1;

  B1( &v1 );
  B13Snd( v1 );
  B2( v1 );
  B34Rcv();
}
```

Compilation

```
void b1( int *v1 ) {
  …
}
void b13snd( int v1 ) {
  masterSend( CB13, &v1,
              sizeof(v1) );
}
void b2( int v1 ) {
  …
  masterSend( C2, );
  …
}
void b34rcv() {
  masterReceive( CB34, 0, 0 );
}

void main(void) {
  int v1;
  b1( &v1 );
  b13send( v1 );
  b2( v1 );
  b34rcv();
}
```



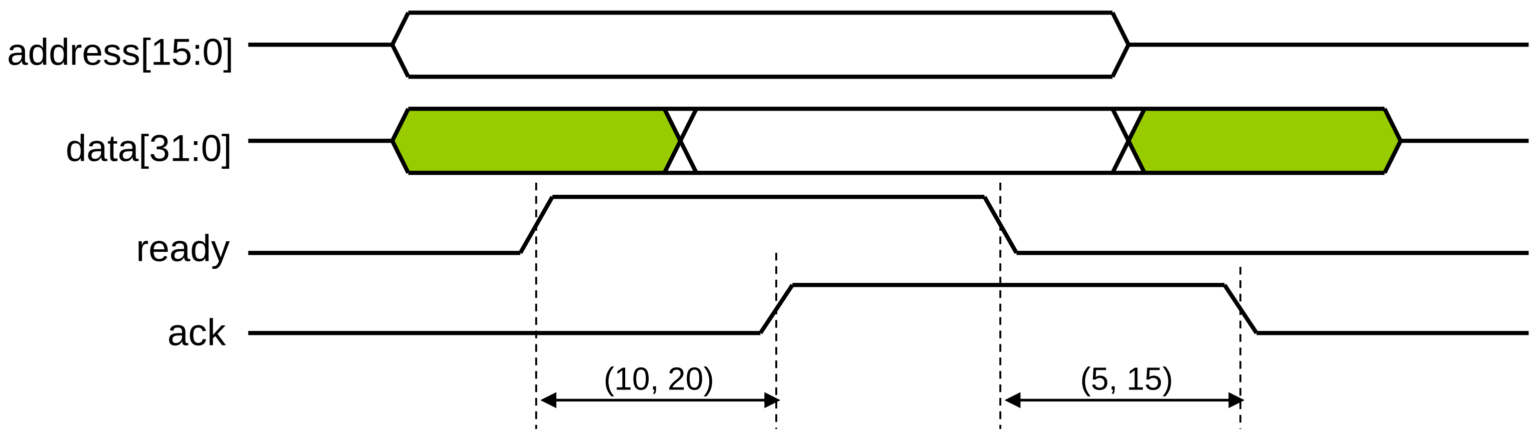
Interface Synthesis



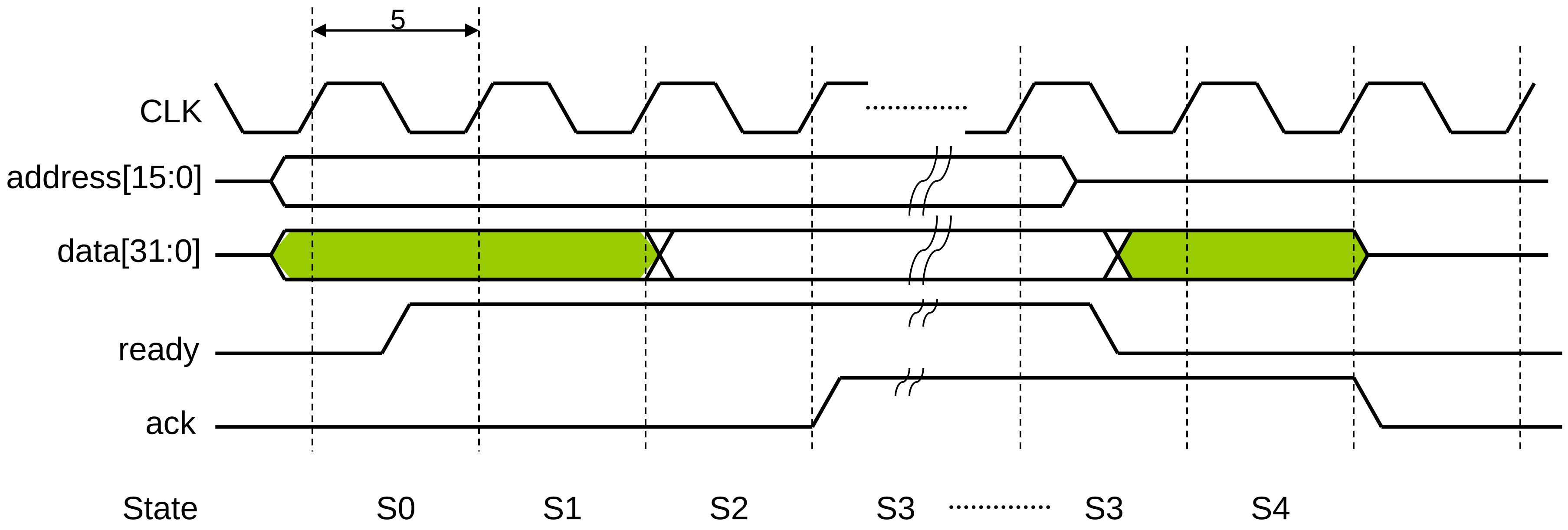
- **Implement communication on components**
  - Hardware bus interface logic
  - Software bus drivers

Hardware Interface Synthesis

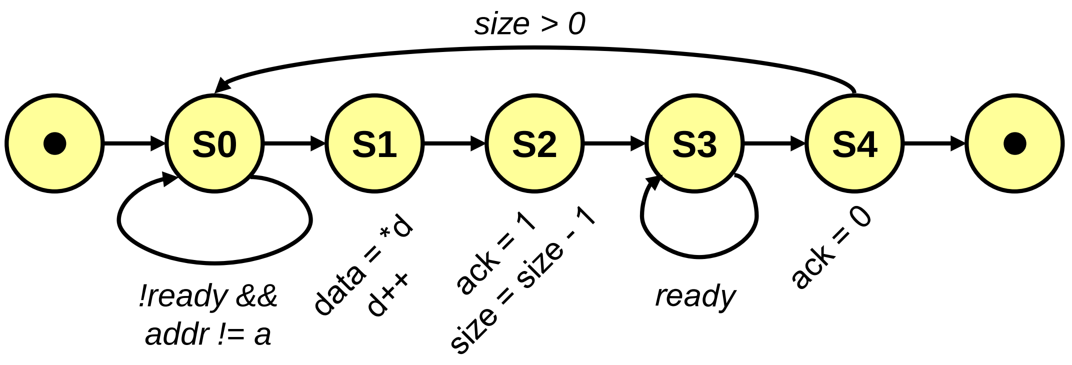
- **Specification: timing diagram / constraints**

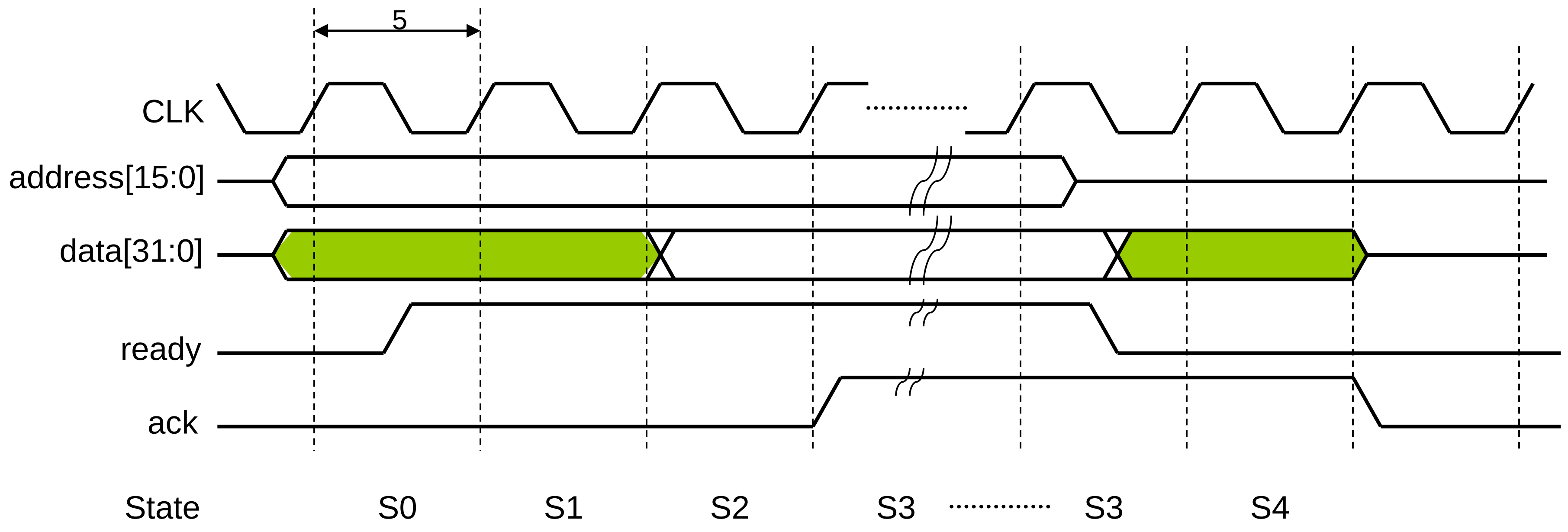


- **FSMD implementation: clock cycles**



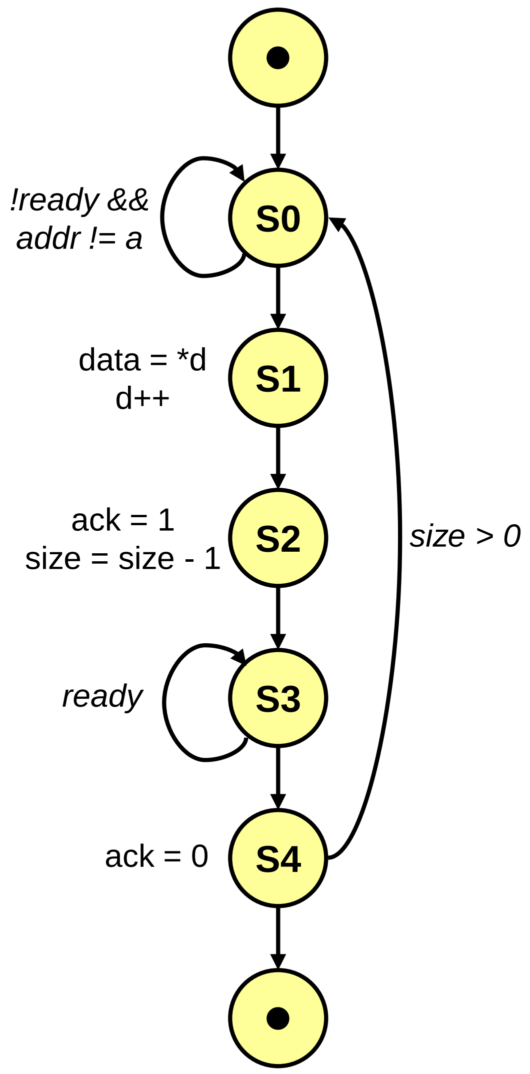
Hardware Interface FSMD





Hardware Interface FSMD (2)

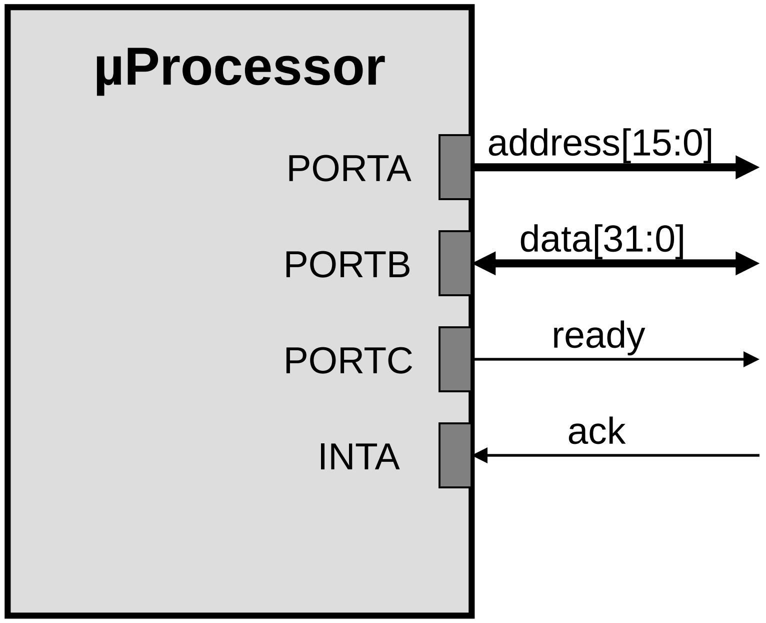
```
channel PE2Bus( in bit[15:0] addr, inout bit[31:0] data,
                ISignal ready, OSignal ack )
  implements IBusSlave
{
  void slaveSend( int a, void* d, int size ) {
    enum { S0, S1, S2, S3, S4, S5 } state = S0;
    while( state != S5 ) {
      waitfor( PE2_CLK );        // wait for clock period
      switch( state ) {
          case S0:  // sample ready, address
            if( (ready.val() == 1) && (addr == a) ) state = S1;
            break;
          case S1:  // read memory, drive data bus
            data = *( ((long*)d)++ );
            state = S2;
            break;
          case S2: // raise ack, advance counter
            ack.set( 1 );
            size--;
            state = S3;
            break;
          case S3: // sample ready
            if( ready.val() == 0 ) state = S4;
            break;
          case S4: // reset ack, loop condition
            ack.set( 0 );
            if( size != 0 ) state = S0 else state = S5;
     }}
   }
  void slaveReceive( int a, void* d, int size ) { … }
};
```



Software Interface Synthesis

- **Implement communication over processor bus**
  - Map bus wires to processor ports
    - Match with processor ports
    - Map to general I/O ports
  - Generate assembly code
    - Protocol layer: bus protocol timing via processor’s I/O instructions
    - Application layer: synchronization, arbitration, data slicing
    - Interrupt handlers for synchronization

Software Interface Driver



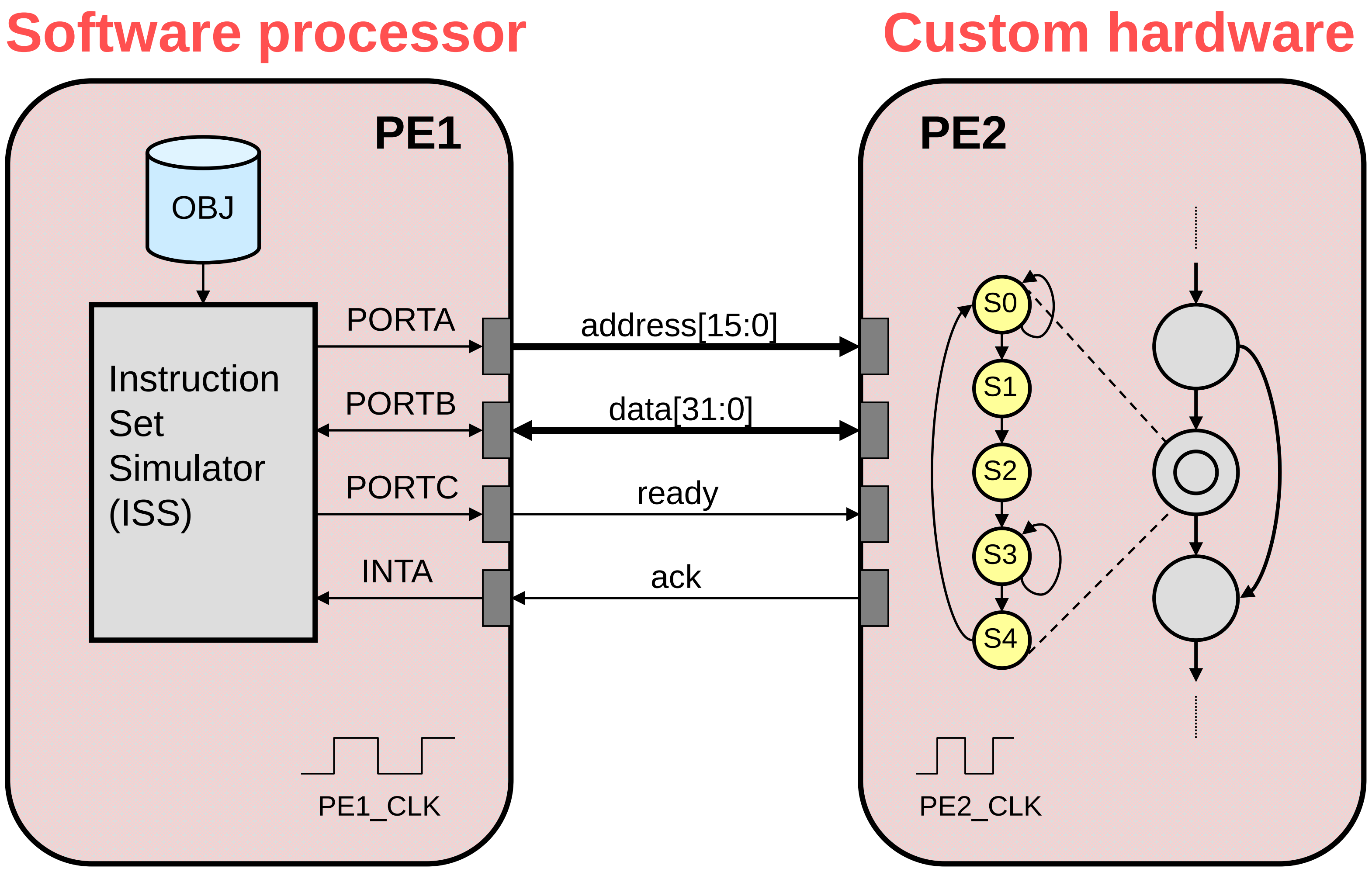
Bus driver:

```
FmasterWrite   ; protocol layer
  OUTA     r0            ; write addr
  OUTB     r1            ; write data
  OUTC     #0001         ; raise ready
  MOVE     ack_event,r2
  CALL     Fevent_wait   ; wait for ack ev.
  OUTC     #0000         ; lower ready
  RET

FmasterSend    ; application layer
  LOOP     L1END,r2      ; loop over data
  MOVE     (r6)+,r1
  CALL     FmasterWrite ; call protocol
L1END
  RET
```

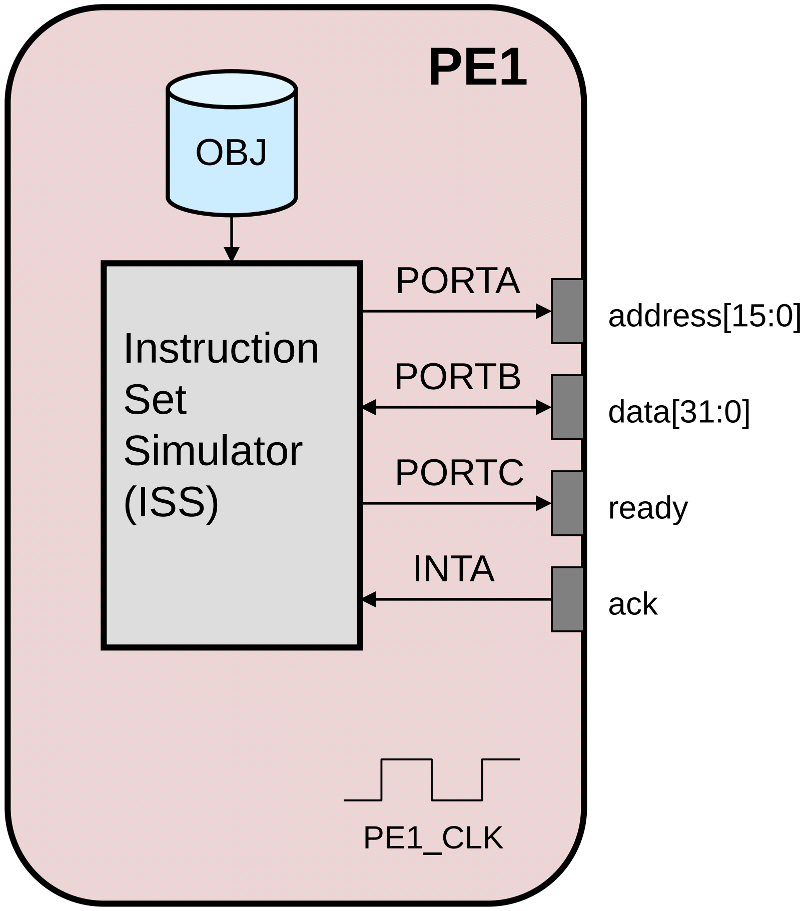
Interrupt handler:

```
FintaHandler
  PUSH     r2
  MOVE     ack_event,r2
  CALL     Fevent_notify  ; notify ack ev.
  POP      r2
  RTI
```

Implementation Model



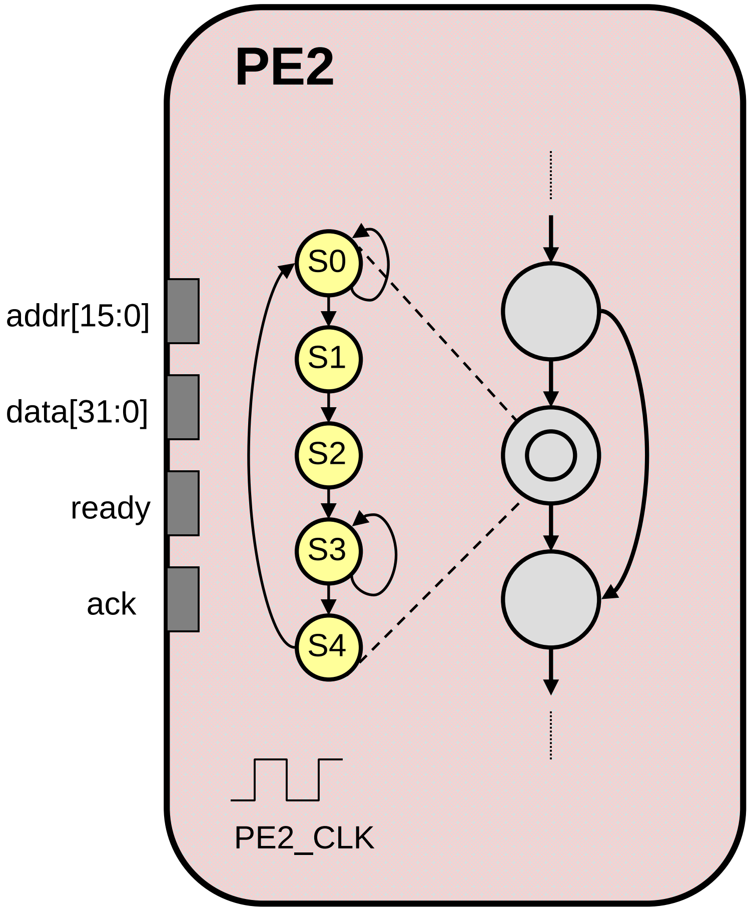
Implementation Model (PE1)



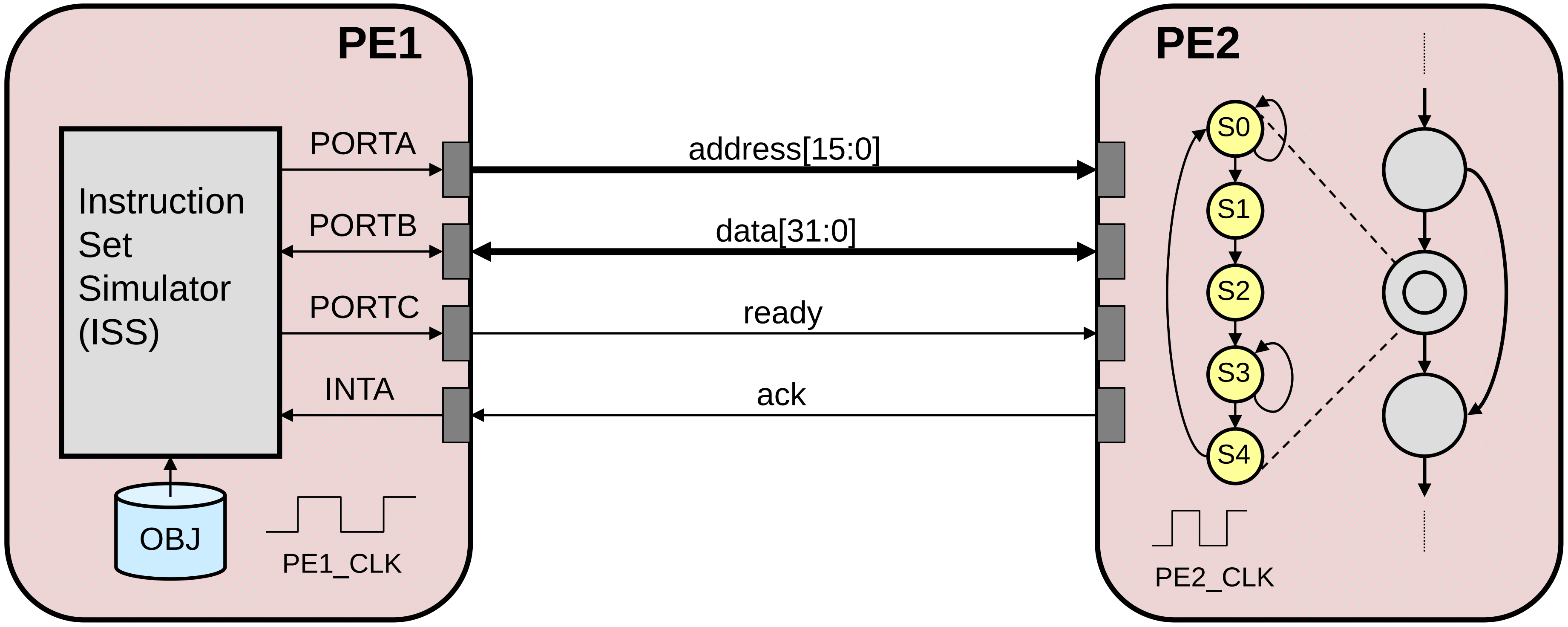
```
#include <iss.h>     // ISS API

behavior PE1( out   bit[15:0] addr,
              inout bit[31:0] data,
              OSignal   ready,
              ISignal   ack )
{
  void main(void)
  {
    // initialize ISS, load program
    iss.startup();
    iss.load("a.out");

    // run simulation
    for( ; ; ) {
      // drive inputs
      iss.porta = addr;
      iss.portb = data;
      iss.inta  = ack.val();

      // run processor cycle
      iss.exec();
      waitfor( PE1_CLK );

      // update outputs
      data = iss.portb;
      ready.set( iss.portc & 0x01 );
    }
  }
};
```

Implementation Model (PE2)

```
behavior PE2( in    bit[15:0] addr,
              inout bit[31:0] data,
              ISignal         ready,
              OSignal         ack )
{
  // Interface FSM
  PE2Bus bus1( addr, data, ready, ack );

  int v1;                          // memory

  B13Rcv b13rcv( bus1, v1 );   // FSMDs
  B3     b3    ( v1, bus1 );
  B34Snd b34snd( bus1 );

  void main(void)
  {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```



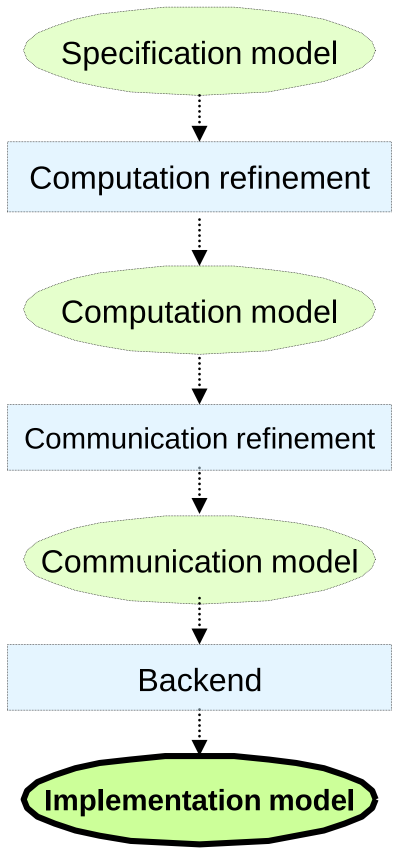
Implementation Model (Top)



```
behavior Design() {
  bit[15:0] addr;
  bit[31:0] data;
  Signal    ready;
  Signal    ack;

  PE1 pe1( address, data, ready, ack );
  PE2 pe2( address, data, ready, ack );

  void main(void) {
    par { pe1.main(); pe2.main(); }
  }
};
```

Implementation Model

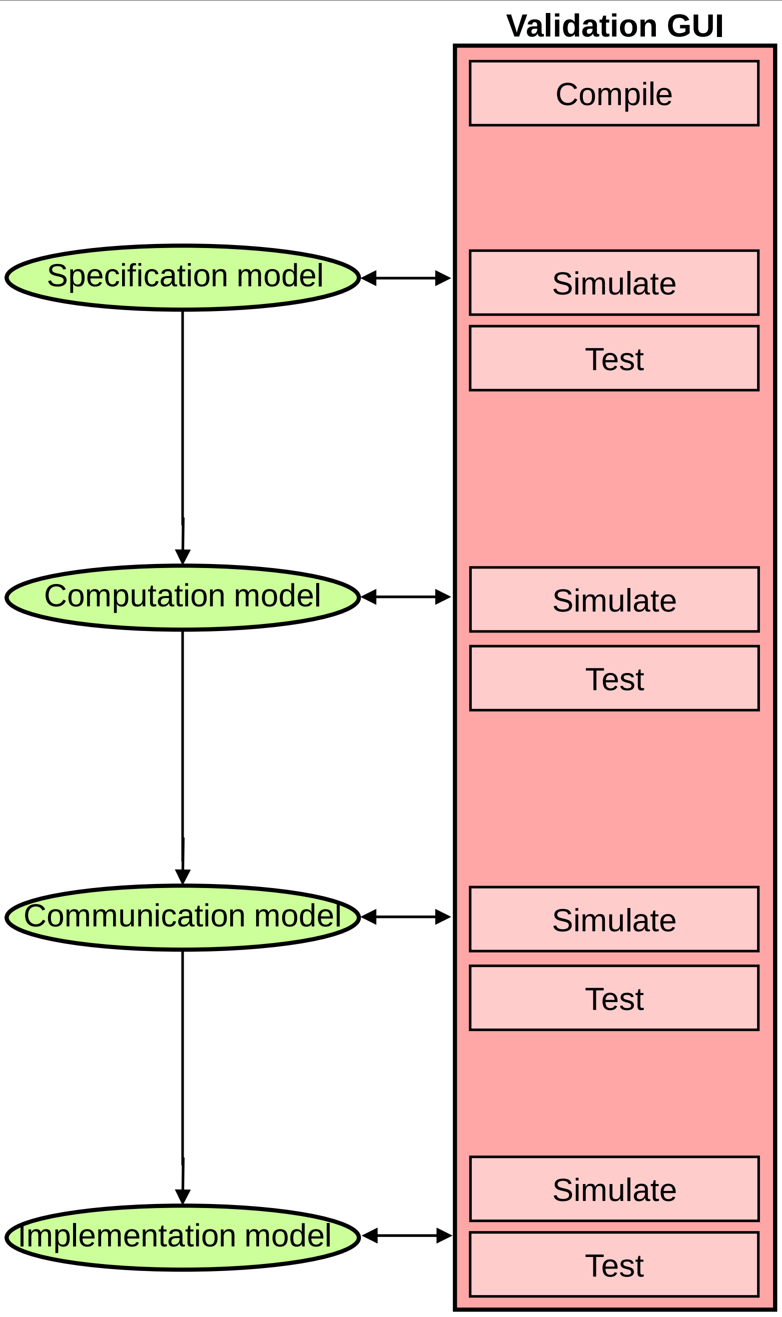
- **Cycle-accurate system description**
  - RTL description of hardware
    - Behavioral/structural FSMD view
  - Object code for processors
    - Instruction-set co-simulation
  - Clocked bus communication
    - Bus interface timing based on PE clock



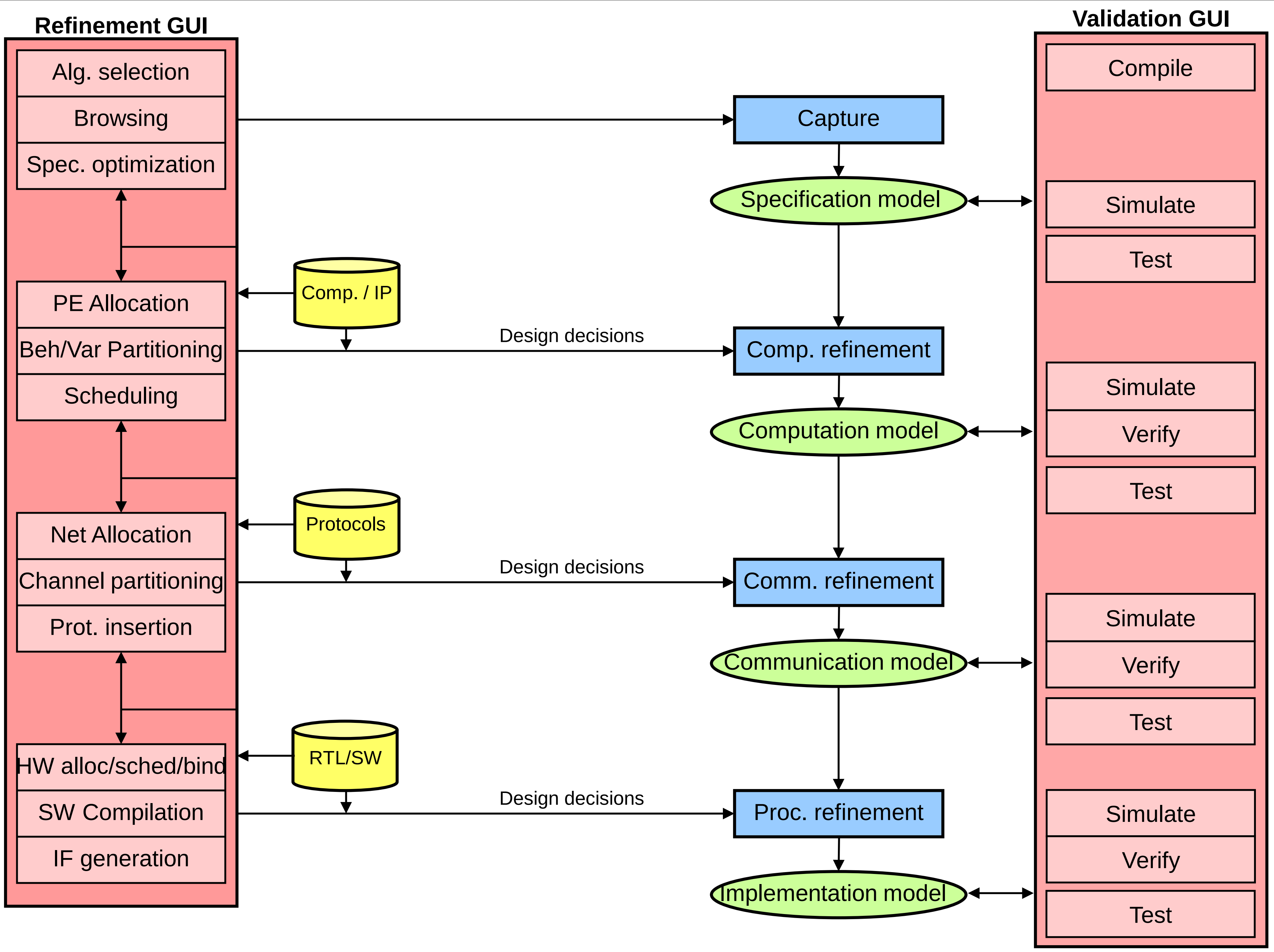
Lecture 5: Outline

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

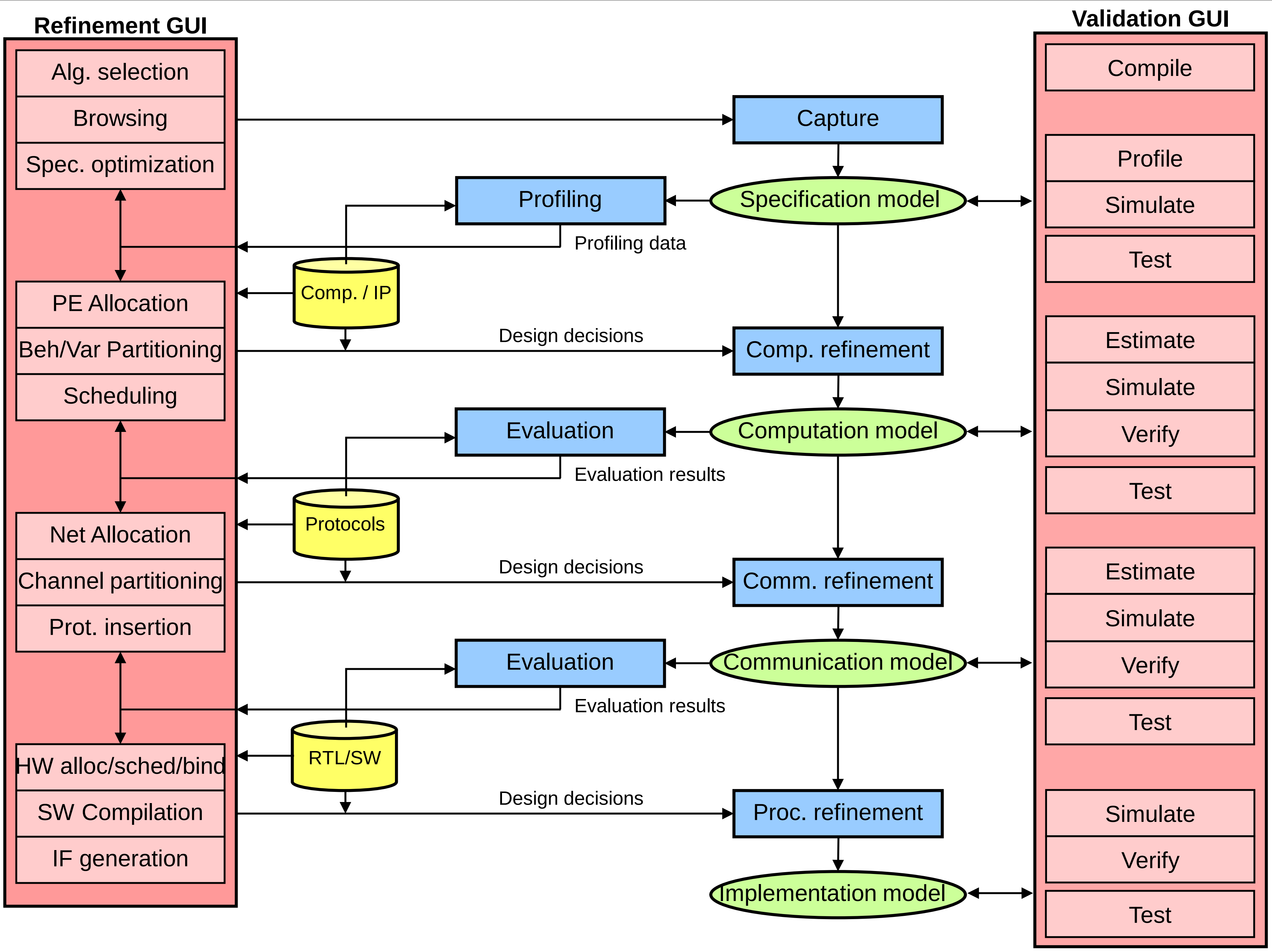
Design Environment (1): Modeling



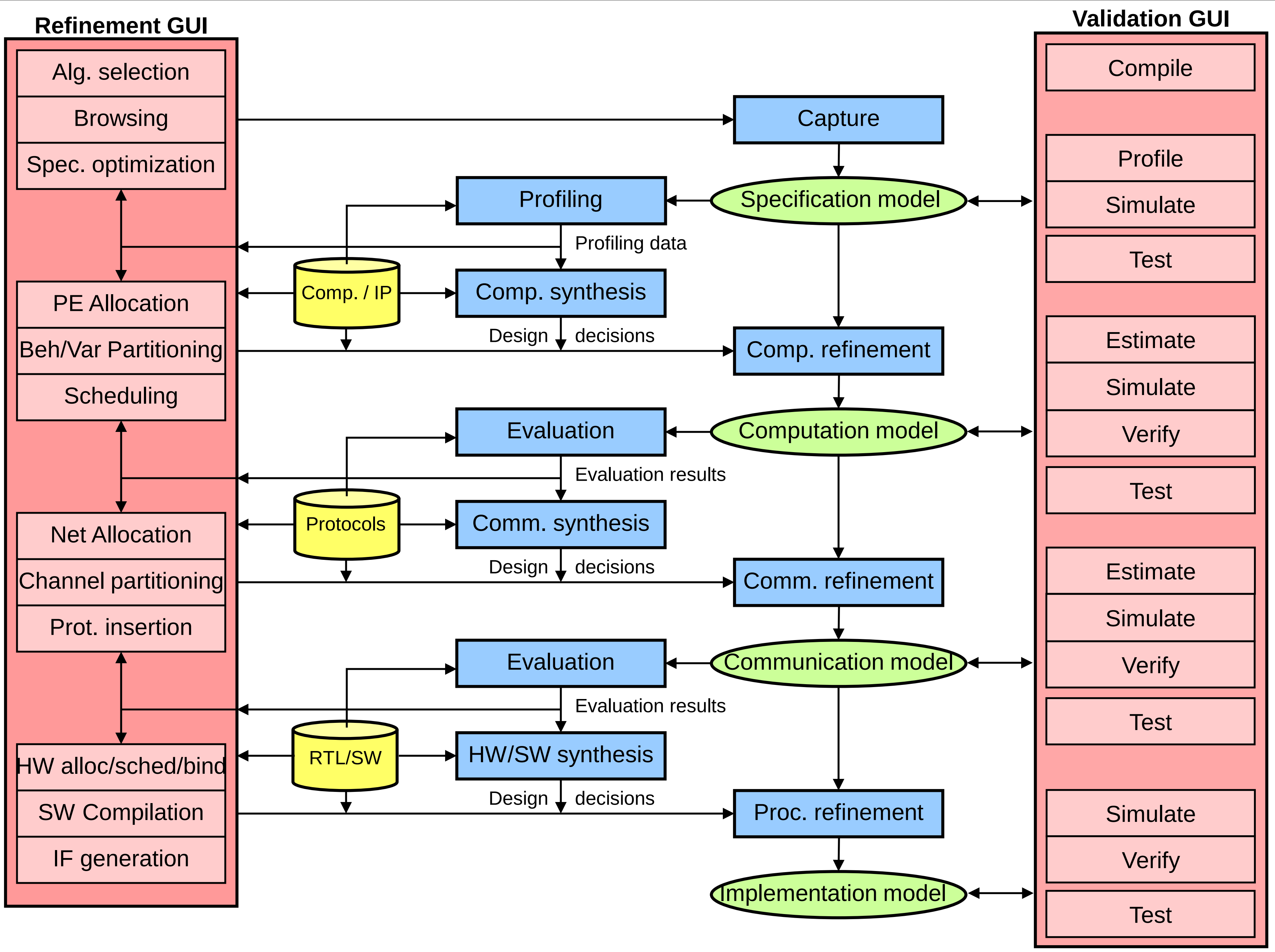
Design Environment (2): Refinement



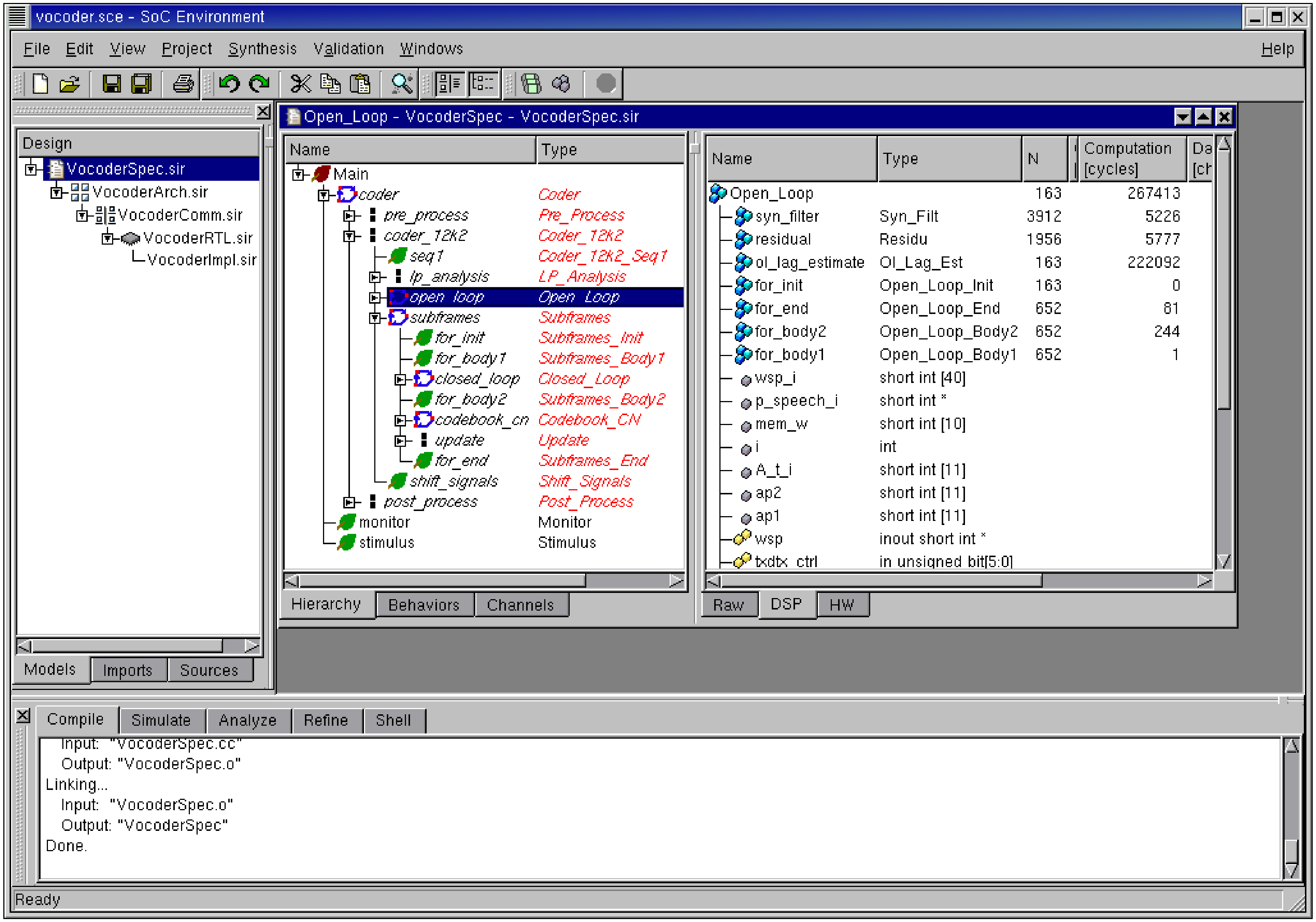
Design Environment (3): Exploration



Design Environment (4): Synthesis



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


vocoder.sce - SoC Environment
File Edit View Project Synthesis Validation Windows Help
Open_Loop - VocoderSpec - VocoderSpec.sir
Models Imports Sources
Hierarchy Behaviors Channels
Raw DSP HW
Compile Simulate Analyze Refine Shell

Design
- VocoderSpec.sir
  - VocoderArch.sir
    - VocoderComm.sir
      - VocoderRTL.sir
        - VocoderImpl.sir

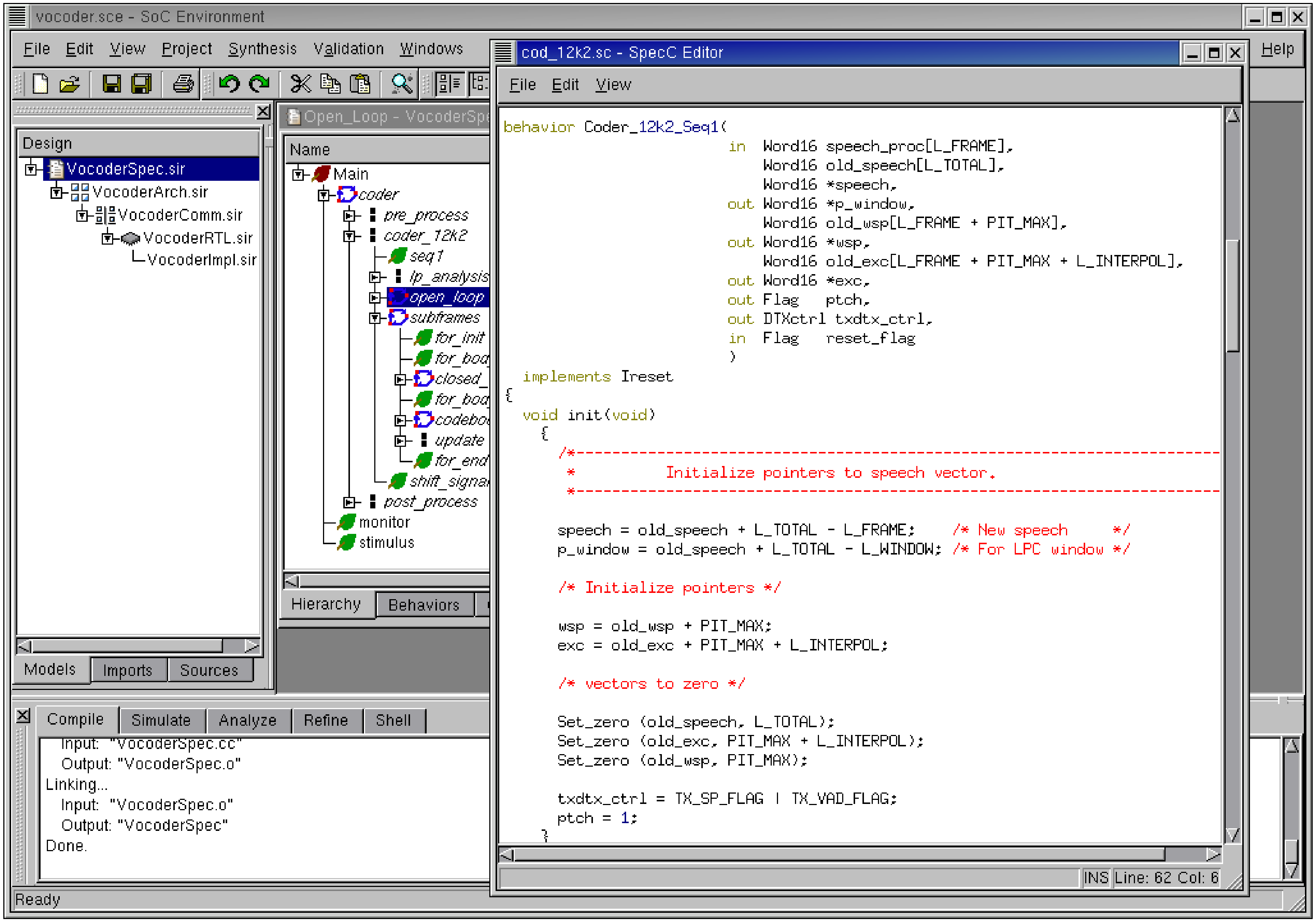
Name	Type
Main	
coder	Coder
pre_process	Pre_Process
coder_12k2	Coder_12k2
seq1	Coder_12k2_Seq1
lp_analysis	LP_Analysis
open_loop	Open_Loop
subframes	Subframes
for_init	Subframes_Init
for_body1	Subframes_Body1
closed_loop	Closed_Loop
for_body2	Subframes_Body2
codebook_cn	Codebook_CN
update	Update
for_end	Subframes_End
shift_signals	Shift_Signals
post_process	Post_Process
monitor	Monitor
stimulus	Stimulus

Name	Type	N	Computation [cycles]	Da [ch
Open_Loop		163	267413	
syn_filter	Syn_Filt	3912	5226	
residual	Residu	1956	5777	
ol_lag_estimate	Ol_Lag_Est	163	222092	
for_init	Open_Loop_Init	163	0	
for_end	Open_Loop_End	652	81	
for_body2	Open_Loop_Body2	652	244	
for_body1	Open_Loop_Body1	652	1	
wsp_i	short int [40]			
p_speech_i	short int *			
mem_w	short int [10]			
i	int			
A_t_i	short int [11]			
ap2	short int [11]			
ap1	short int [11]			
wsp	inout short int *			
txdtx_ctrl	in unsigned bit[5:0]			

```
  Input:  "VocoderSpec.cc"
  Output: "VocoderSpec.o"
Linking...
  Input:  "VocoderSpec.o"
  Output: "VocoderSpec"
Done.
```

Ready

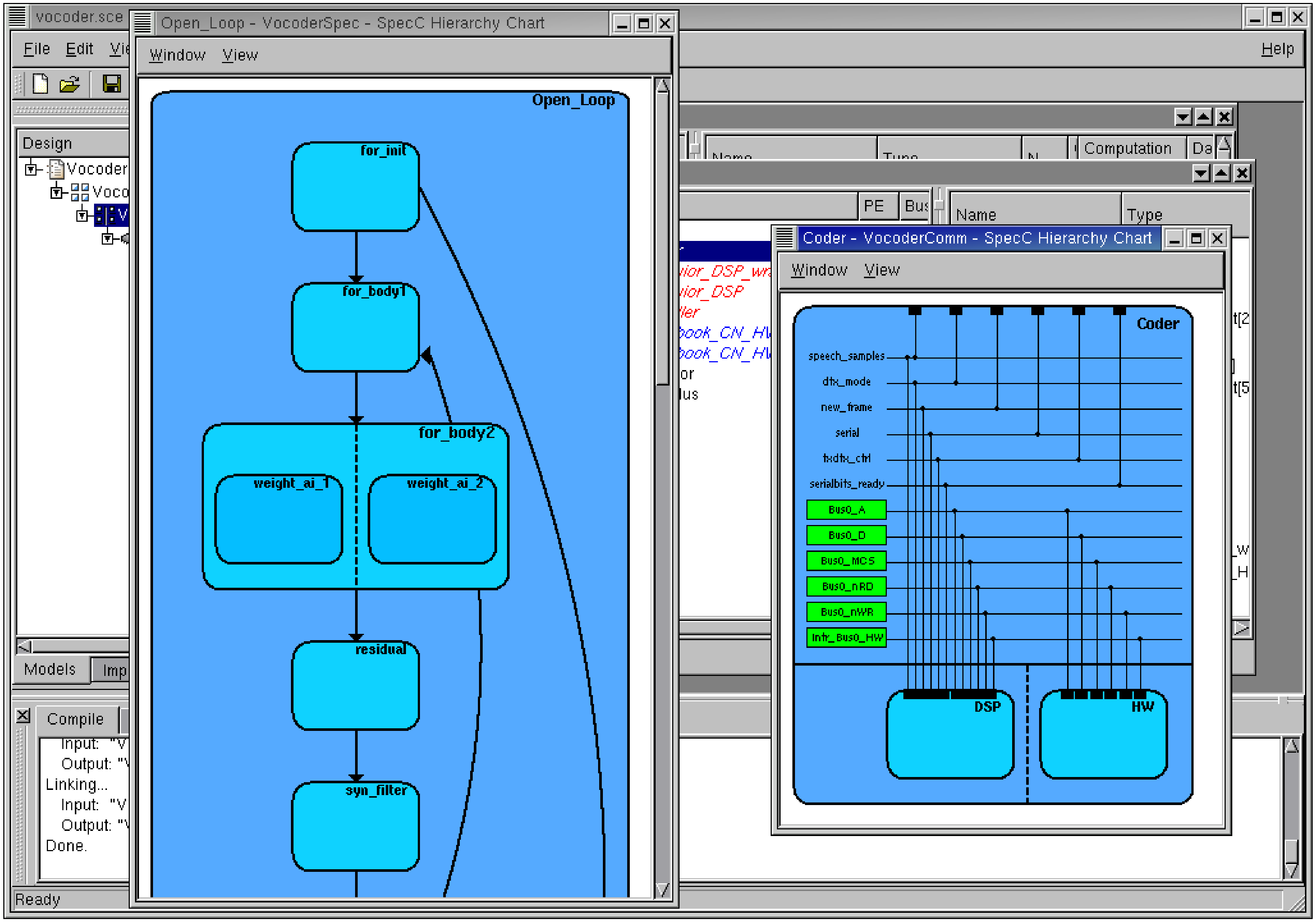
SCE Source Editor



```
behavior Coder_12k2_Seq1(
                         in  Word16 speech_proc[L_FRAME],
                             Word16 old_speech[L_TOTAL],
                             Word16 *speech,
                         out Word16 *p_window,
                             Word16 old_wsp[L_FRAME + PIT_MAX],
                         out Word16 *wsp,
                             Word16 old_exc[L_FRAME + PIT_MAX + L_INTERPOL],
                         out Word16 *exc,
                         out Flag   ptch,
                         out DTXctrl txdtx_ctrl,
                         in  Flag   reset_flag
                         )
  implements Ireset
{
  void init(void)
    {
      /*------------------------------------------------------------------------
       *          Initialize pointers to speech vector.
       *------------------------------------------------------------------------


      speech = old_speech + L_TOTAL - L_FRAME;     /* New speech     */
      p_window = old_speech + L_TOTAL - L_WINDOW; /* For LPC window */

      /* Initialize pointers */

      wsp = old_wsp + PIT_MAX;
      exc = old_exc + PIT_MAX + L_INTERPOL;

      /* vectors to zero */

      Set_zero (old_speech, L_TOTAL);
      Set_zero (old_exc, PIT_MAX + L_INTERPOL);
      Set_zero (old_wsp, PIT_MAX);

      txdtx_ctrl = TX_SP_FLAG | TX_VAD_FLAG;
      ptch = 1;
    }
```

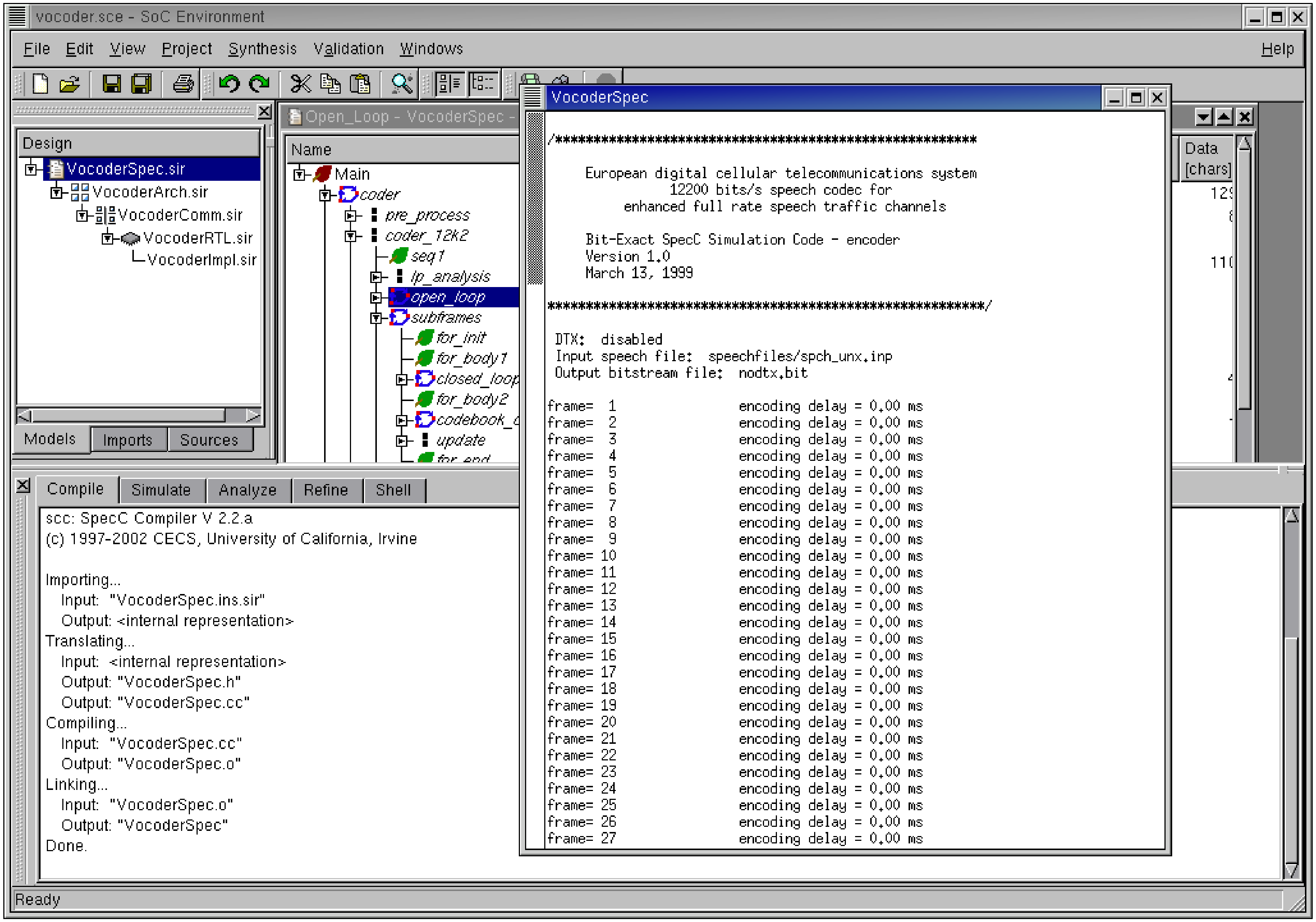
```
   Input:  "VocoderSpec.cc"
   Output: "VocoderSpec.o"
Linking...
   Input:  "VocoderSpec.o"
   Output: "VocoderSpec"
Done.
```

SCE Hierarchy Displays



```
  Input:  "V
  Output: "V
Linking...
  Input:  "V
  Output: "V
Done.
```

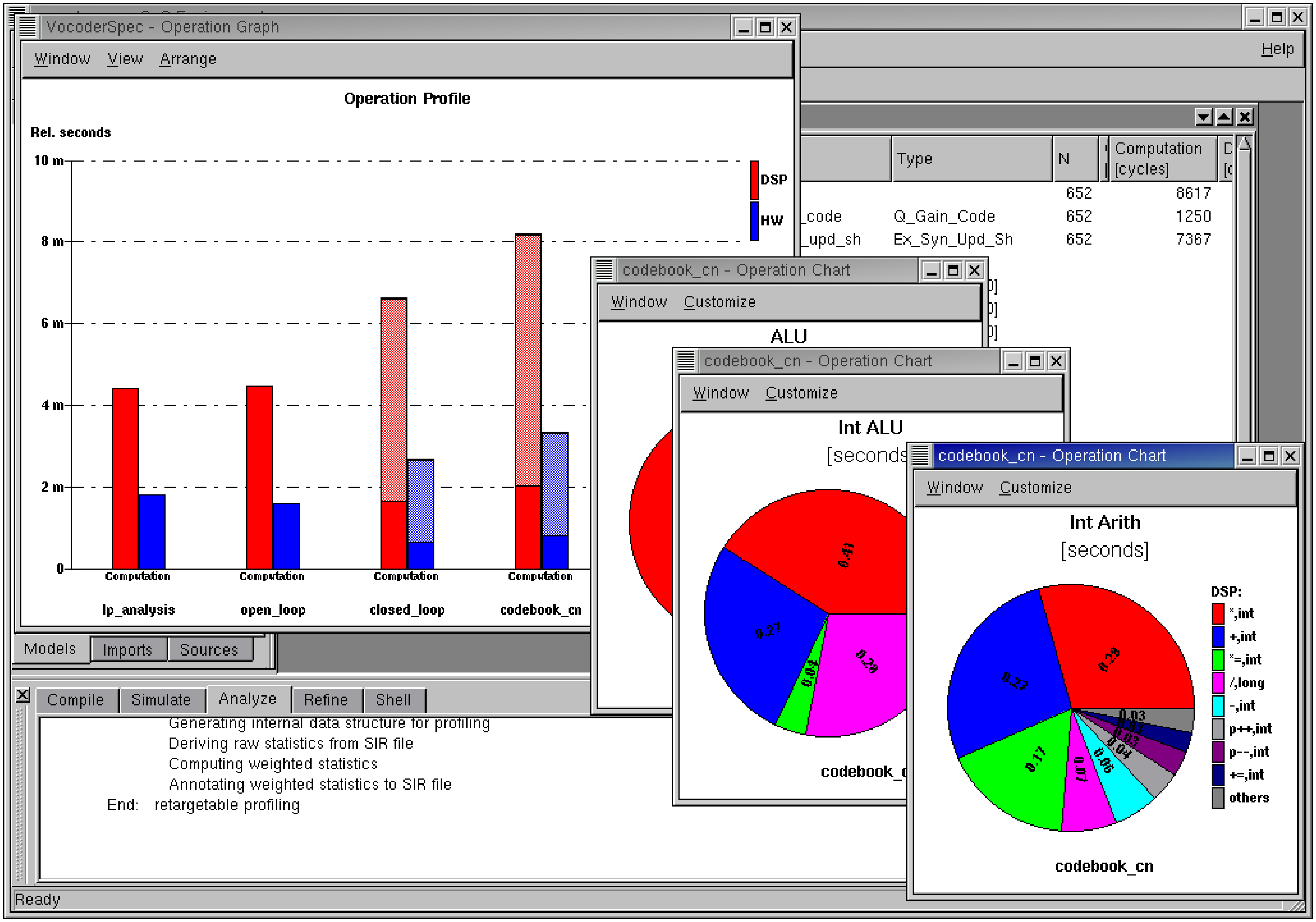
SCE Compiler and Simulator



```
scc: SpecC Compiler V 2.2.a
(c) 1997-2002 CECS, University of California, Irvine

Importing...
   Input:  "VocoderSpec.ins.sir"
   Output: <internal representation>
Translating...
   Input:  <internal representation>
   Output: "VocoderSpec.h"
   Output: "VocoderSpec.cc"
Compiling...
   Input:  "VocoderSpec.cc"
   Output: "VocoderSpec.o"
Linking...
   Input:  "VocoderSpec.o"
   Output: "VocoderSpec"
Done.
```

```
/*************************************************************

    European digital cellular telecommunications system
                12200 bits/s speech codec for
          enhanced full rate speech traffic channels

    Bit-Exact SpecC Simulation Code - encoder
    Version 1.0
    March 13, 1999

*************************************************************/

 DTX:  disabled
 Input speech file:  speechfiles/spch_unx.inp
 Output bitstream file:  nodtx.bit

frame=  1               encoding delay = 0.00 ms
frame=  2               encoding delay = 0.00 ms
frame=  3               encoding delay = 0.00 ms
frame=  4               encoding delay = 0.00 ms
frame=  5               encoding delay = 0.00 ms
frame=  6               encoding delay = 0.00 ms
frame=  7               encoding delay = 0.00 ms
frame=  8               encoding delay = 0.00 ms
frame=  9               encoding delay = 0.00 ms
frame= 10               encoding delay = 0.00 ms
frame= 11               encoding delay = 0.00 ms
frame= 12               encoding delay = 0.00 ms
frame= 13               encoding delay = 0.00 ms
frame= 14               encoding delay = 0.00 ms
frame= 15               encoding delay = 0.00 ms
frame= 16               encoding delay = 0.00 ms
frame= 17               encoding delay = 0.00 ms
frame= 18               encoding delay = 0.00 ms
frame= 19               encoding delay = 0.00 ms
frame= 20               encoding delay = 0.00 ms
frame= 21               encoding delay = 0.00 ms
frame= 22               encoding delay = 0.00 ms
frame= 23               encoding delay = 0.00 ms
frame= 24               encoding delay = 0.00 ms
frame= 25               encoding delay = 0.00 ms
frame= 26               encoding delay = 0.00 ms
frame= 27               encoding delay = 0.00 ms
```

SCE Profiling and Analysis



	Type	N	Computation [cycles]	C
		652	8617	
_code	Q_Gain_Code	652	1250	
_upd_sh	Ex_Syn_Upd_Sh	652	7367	

```
                  Generating internal data structure for profiling
                  Deriving raw statistics from SIR file
                  Computing weighted statistics
                  Annotating weighted statistics to SIR file
         End:   retargetable profiling
```

Ready

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