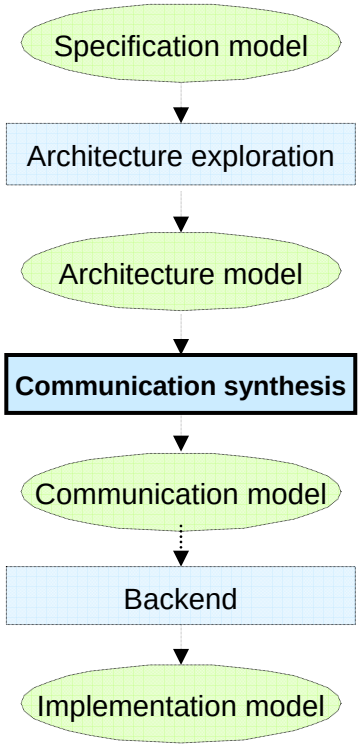
CprE 588
Embedded Computer Systems

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

Lecture #6 – Model Refinement

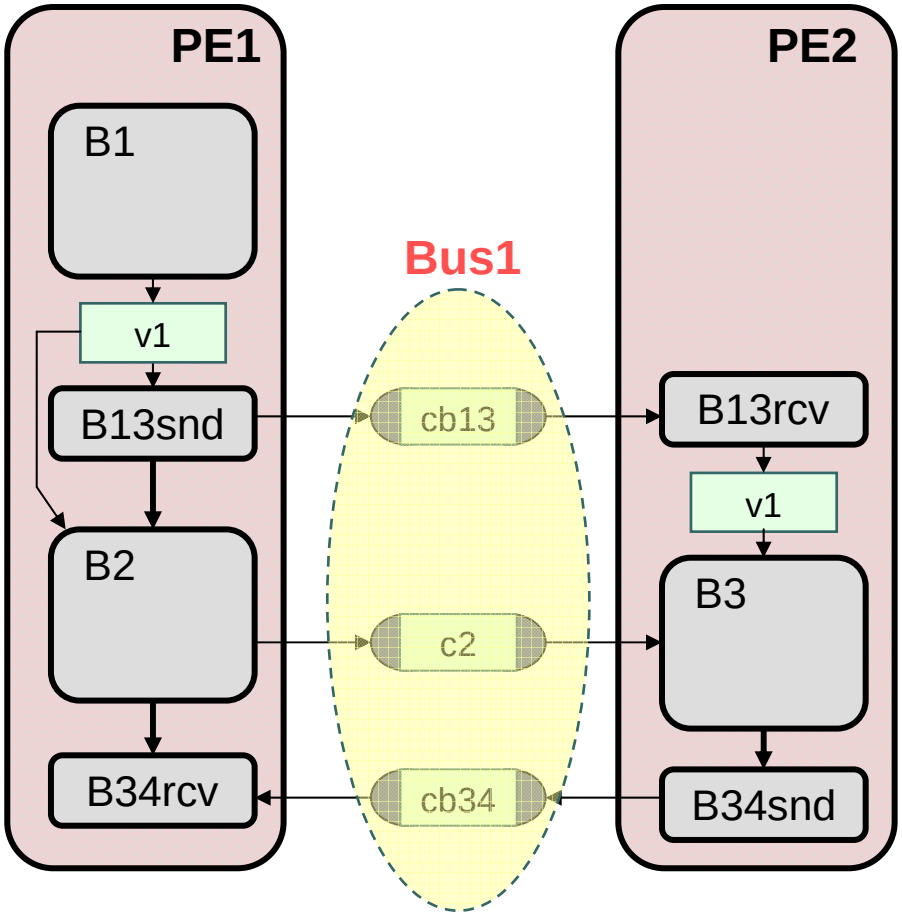
Communication Synthesis

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



R. Domer, *The SpecC System-Level Design Language and Methodology*, Center for Embedded Systems, University of California-Irvine, 2001.

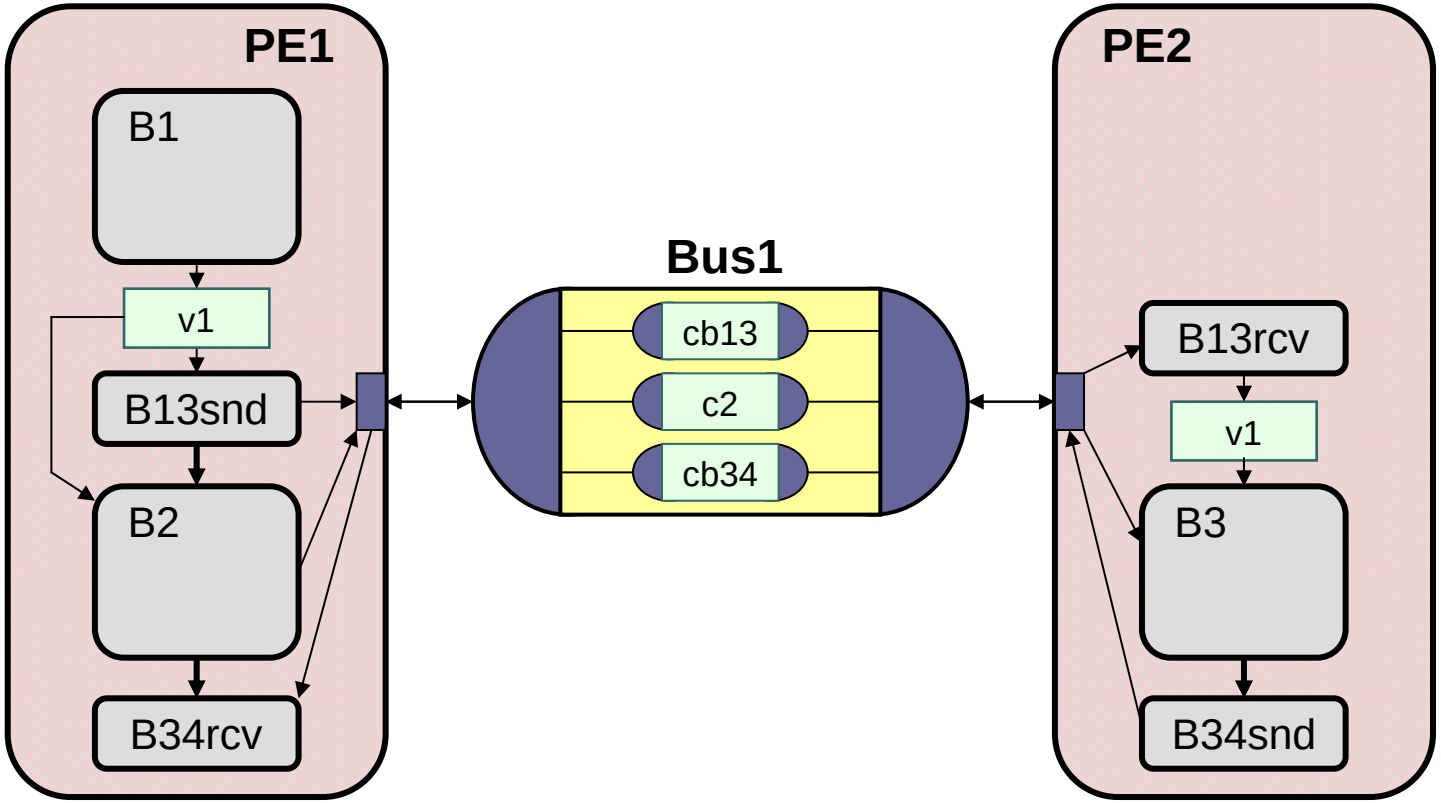
Bus Allocation / Channel Partitioning



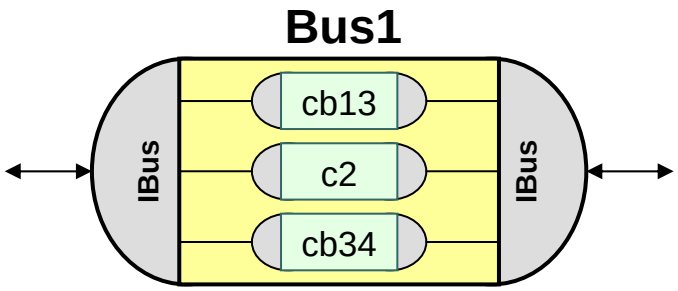
- Allocate busses
- Partition channels
- Update communication

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

Model after Channel Partitioning



Bus Channel



Virtual addressing:

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

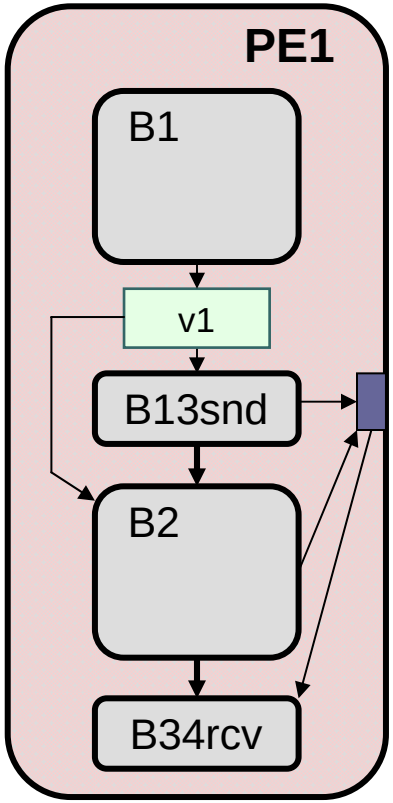
Bus interface:

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

Hierarchical channel:

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

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

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

Model after Channel Partitioning



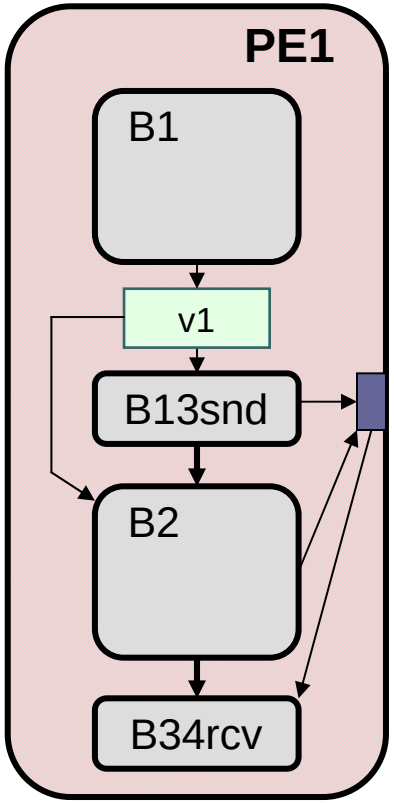
- Leaf behaviors

```
1  behavior B13Snd( in int v1, IBus bus1 )
   {
     void main(void) {
       bus1.send( CB13, &v1, sizeof(v1) );
5    }
   };
```

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

```
1  behavior B34Rcv( IBus bus1 )
   {
     void main(void) {
       bus1.recv( CB34, 0, 0 );
5    }
   };
```

After Channel Partitioning (cont.)




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

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


     void main(void)
15   {
       b1.main();
       b13snd.main();
       b2.main();
       b34rcv.main();
20   }
   };
```

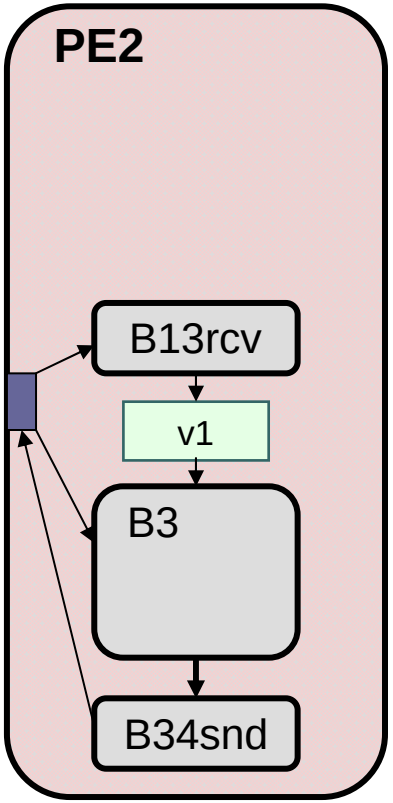
After Channel Partitioning (cont.)

- Leaf behaviors

```
1  behavior B13Rcv( IBus bus1 , out int v1 )
   {
     void main(void) {
       bus1.recv( CB13, &v1, sizeof(v1) );
5    }
   };
```

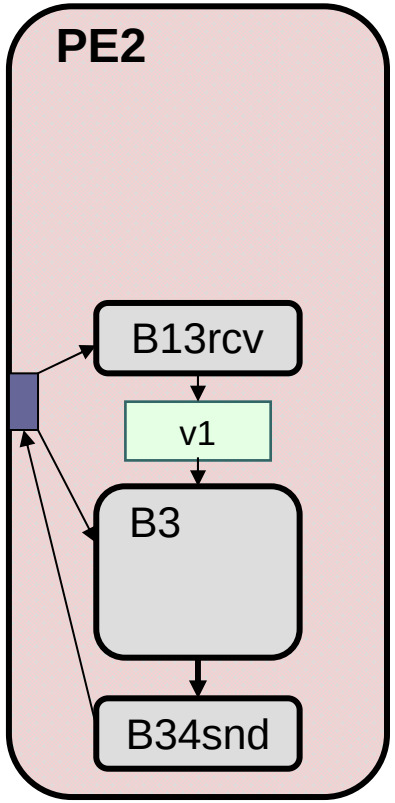
```
1  behavior B3( in int v1, IBus bus1 )
   {
     void main(void) {
       ...
5      bus1.recv( C2, ... );
       ...
     }
   };
```

```
1  behavior B34Snd( IBus bus1 )
   {
     void main(void) {
       bus1.send( CB34, 0, 0 );
5    }
   };
```

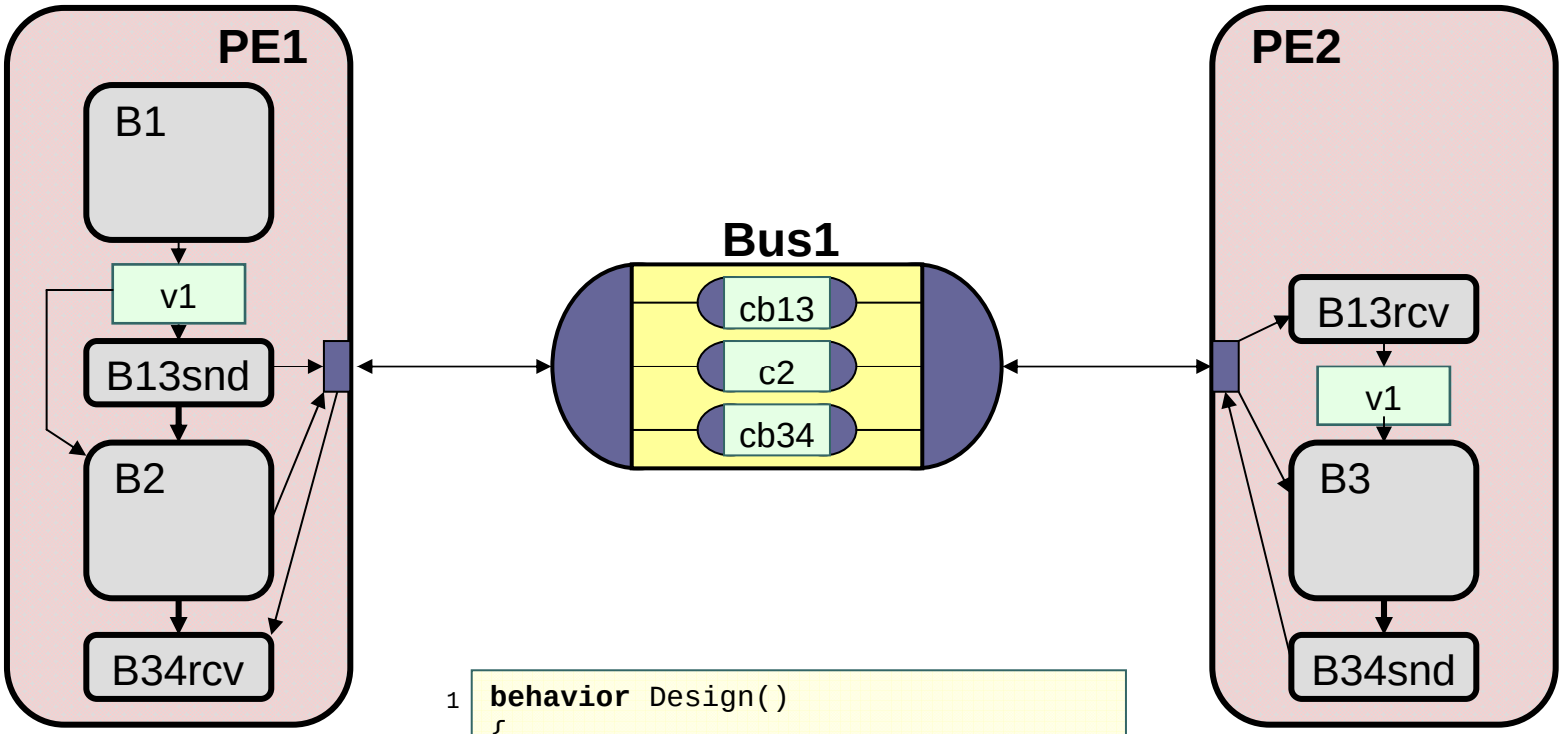


After Channel Partitioning (cont.)

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

5    B13Rcv b13rcv( bus1, v1 );
     B3     b3    ( v1, bus1 );
     B34Snd b34snd( bus1 );
10

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

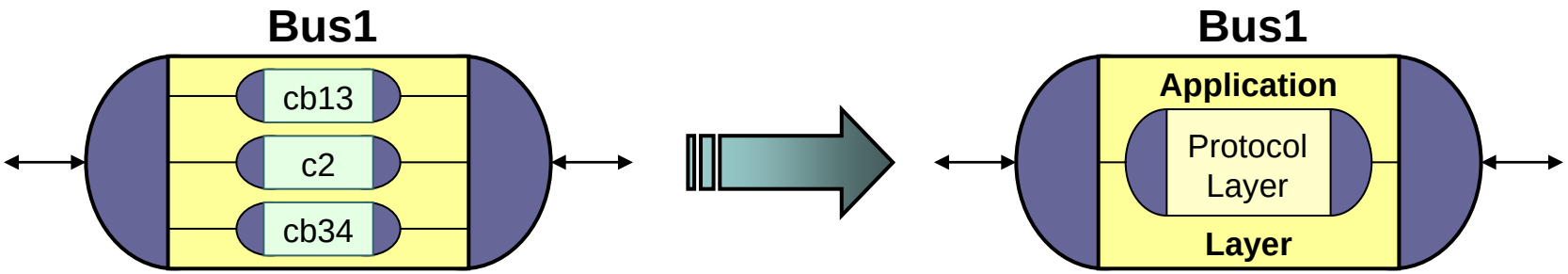


After Channel Partitioning (cont.)




```
1  behavior Design()
   {
     Bus1 bus1;

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

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

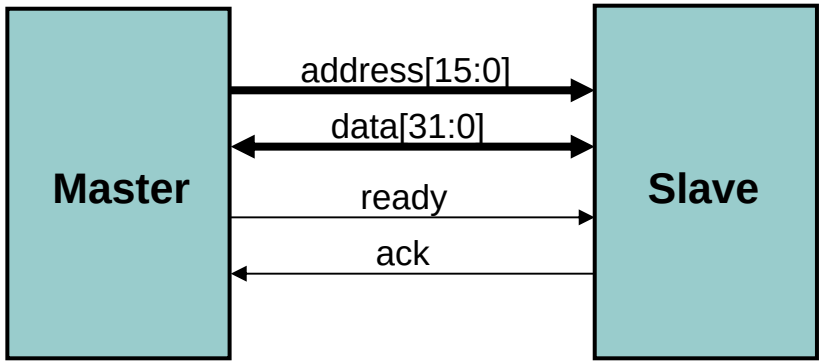
Protocol Insertion



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

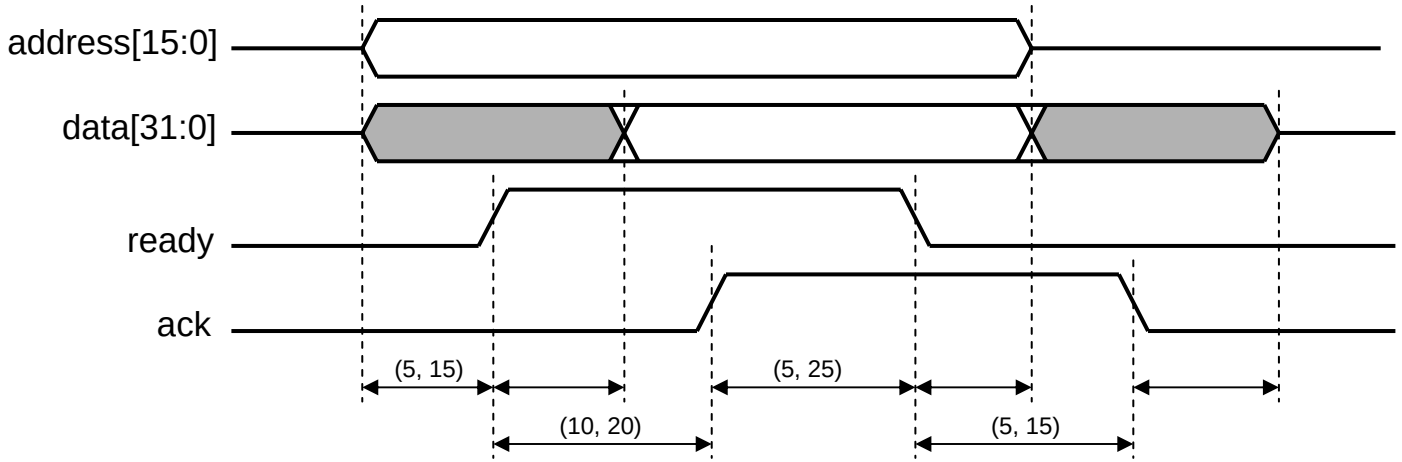
Protocol Example

- Double handshake protocol

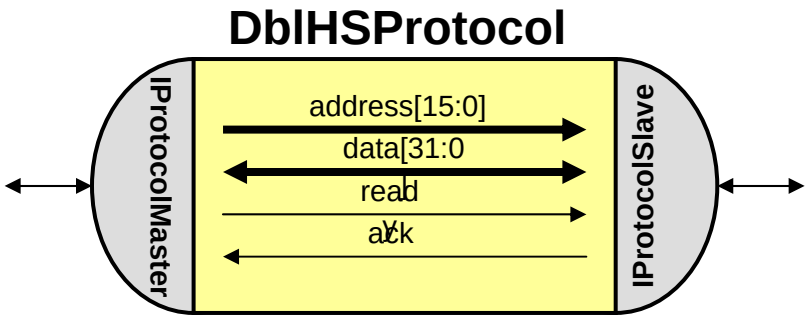


Protocol Example (cont.)

- Timing diagram

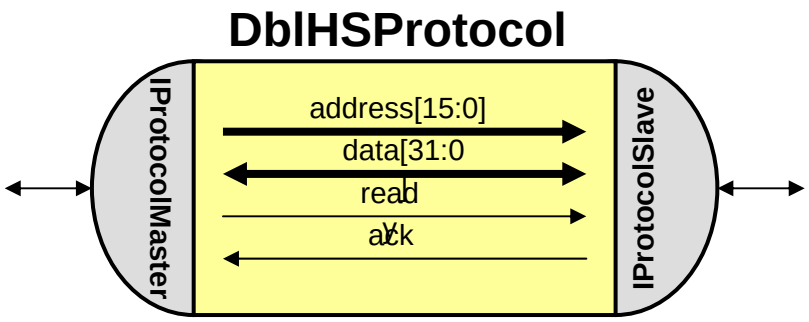


Protocol Layer



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

Protocol Channel




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

10   ...
   };
```

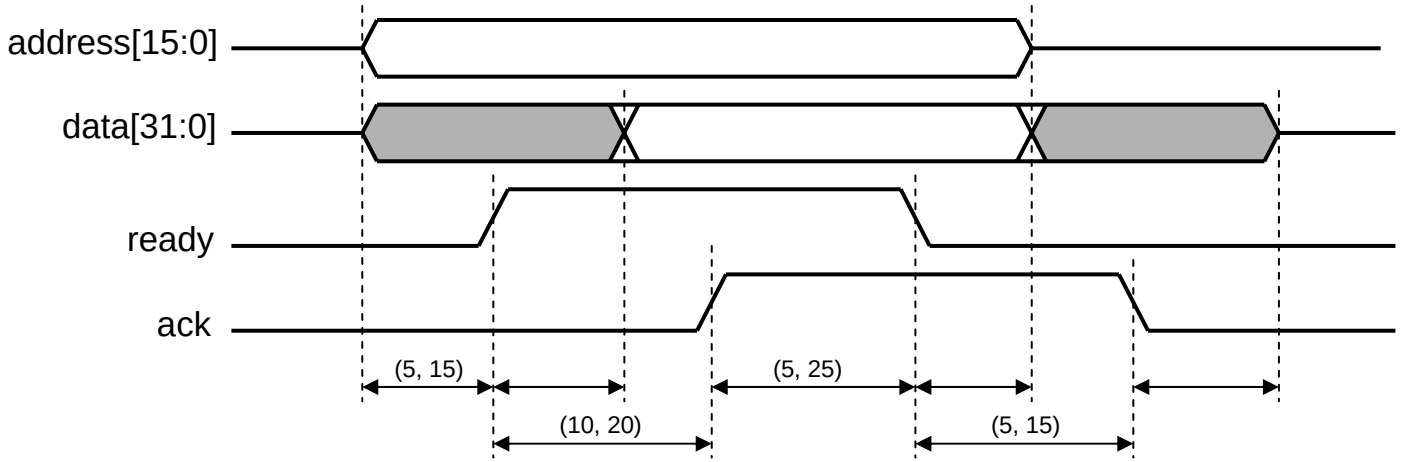
Master interface:

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

Slave interface:

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

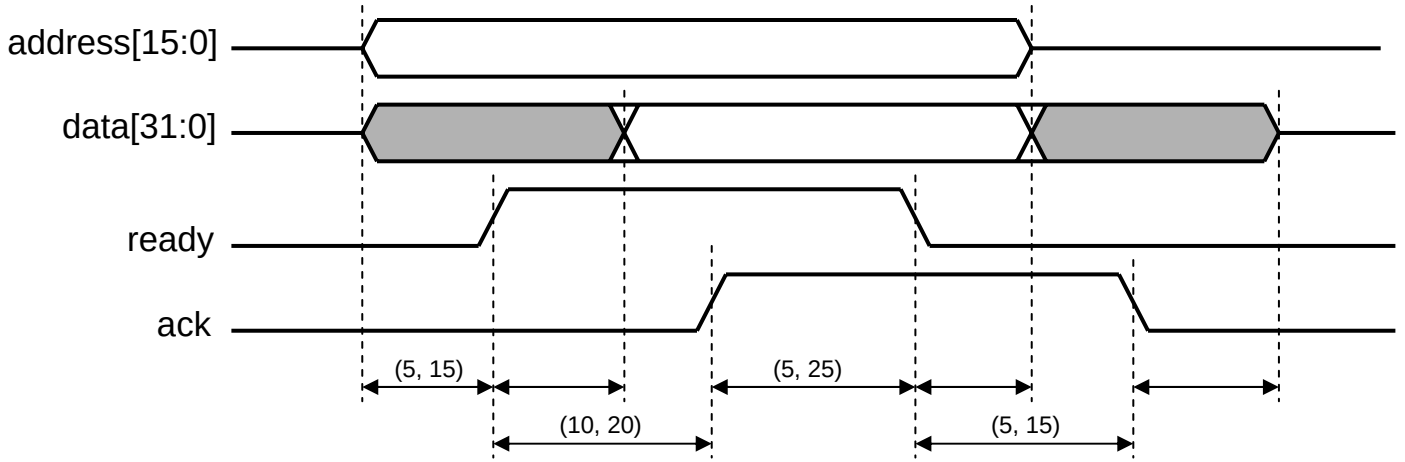
Protocol Master Interface




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

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

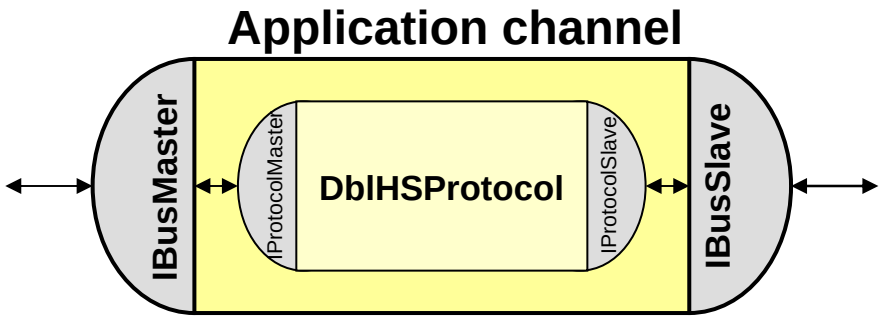
Protocol Slave Interface




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

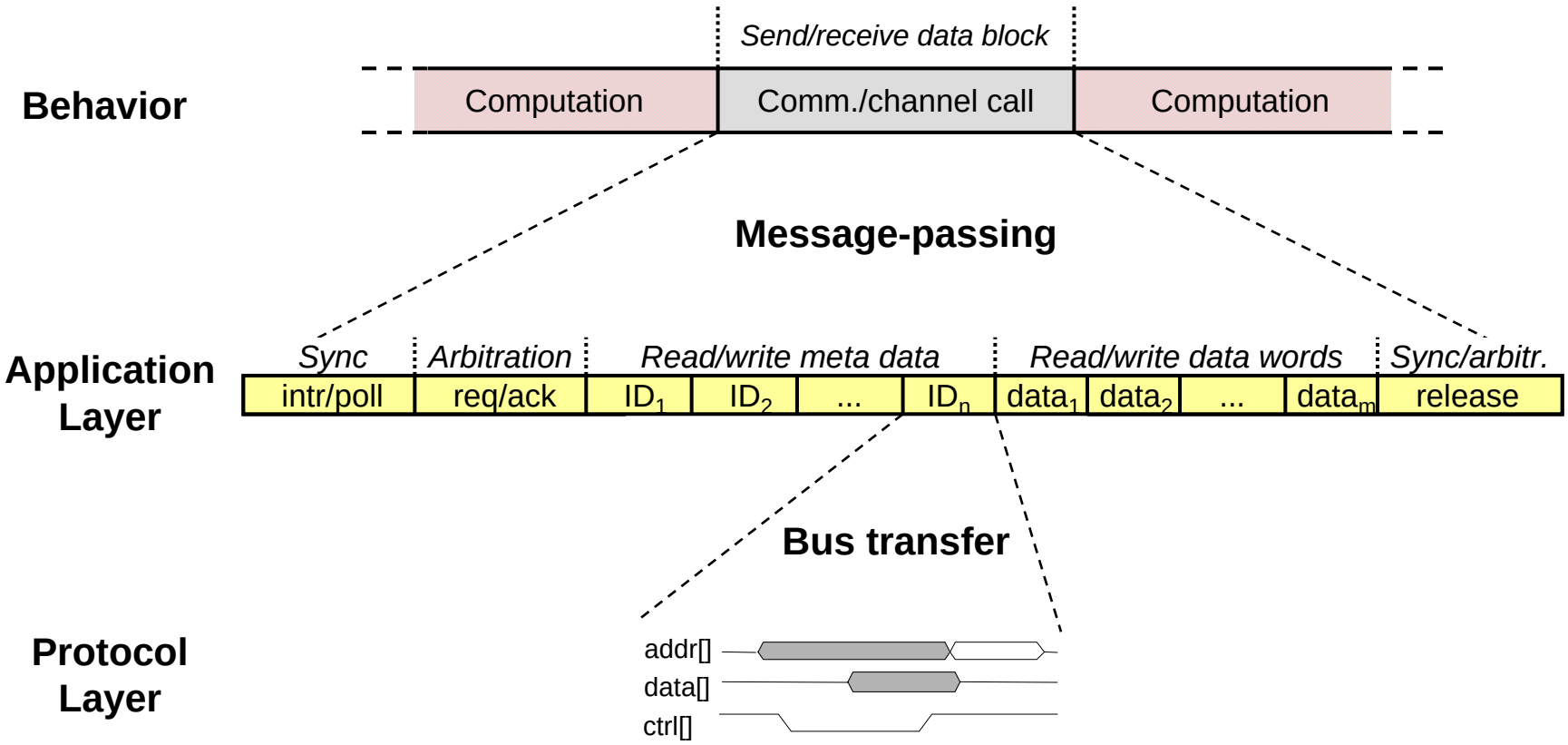
```
1  void slaveWrite( bit[15:0] a, bit[31:0] d )
   {
     do {
       t1: ready.waituntil( 1 );
5      t2: if( a != address ) goto t1;
           data = d;
           waitfor( 12 );   // estimated delay
       t3: ack.set( 1 );
           ready.waituntil( 0 );
10     t4: waitfor( 7 );    // estimated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

Application Layer

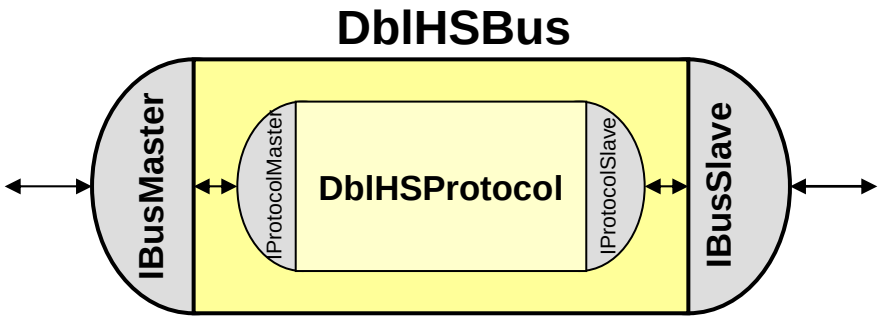


- Implement abstract message-passing over protocol
  - Synchronization
  - Arbitration
  - Addressing
  - Data slicing

Application Layer (cont.)



Application Layer Channel




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

     …
   };
```

Master interface:

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

Slave interface:

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

Application Layer Methods

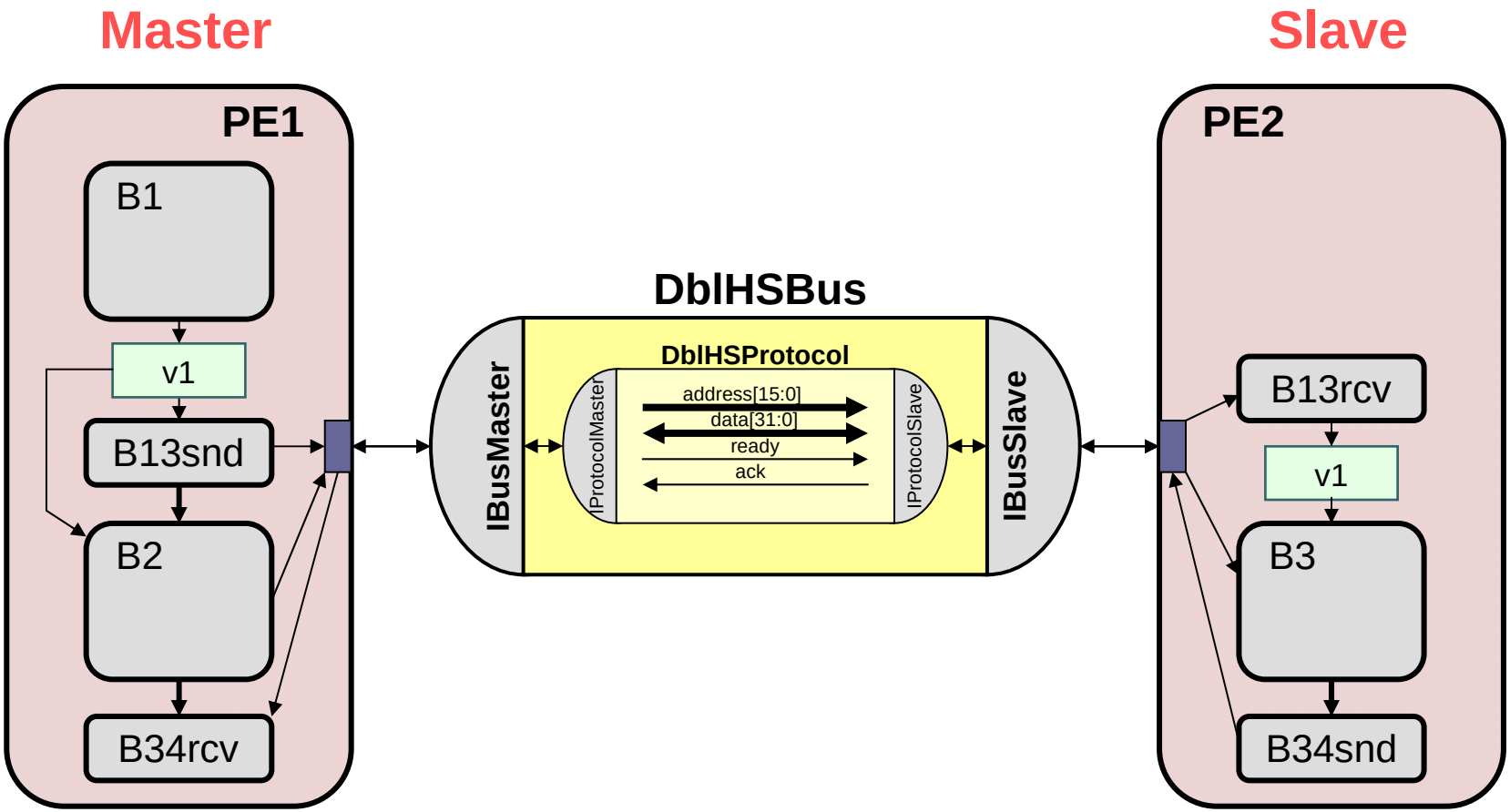
Master interface:

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

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

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

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

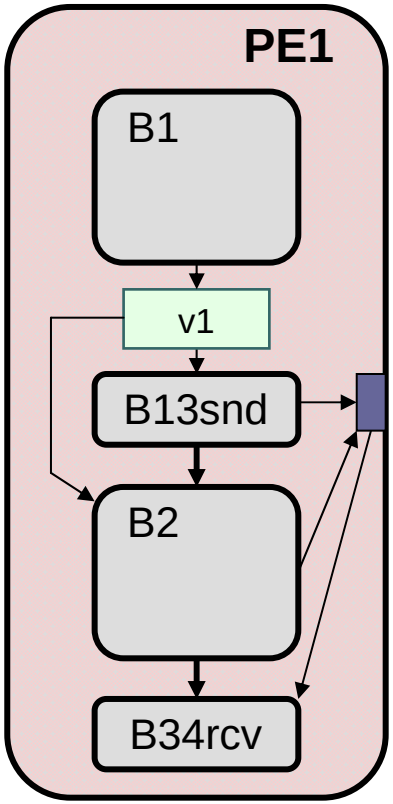
Slave interface:

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

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

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

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

Model after Protocol Insertion



Model after Protocol Insertion (cont.)



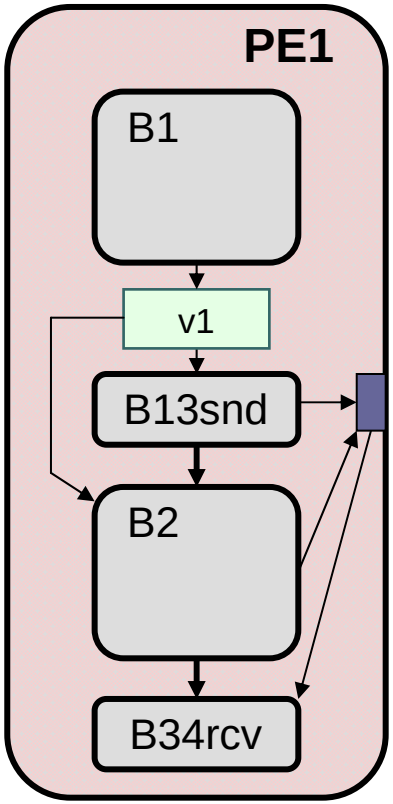
- Leaf behaviors

```
1  behavior B13Snd( in int v1, IBusMaster bus1 )
   {
     void main(void) {
       bus1.masterSend(CB13, &v1, sizeof(v1) );
5    }
   };
```

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

```
1  behavior B34Rcv( IBusMaster bus1 )
   {
     void main(void) {
       bus1.masterRecv( CB34, 0, 0 );
5    }
   };
```

Model after Protocol Insertion (cont.)




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

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


     void main(void)
15   {
       b1.main();
       b13snd.main();
       b2.main();
       b34rcv.main();
20   }
   };
```

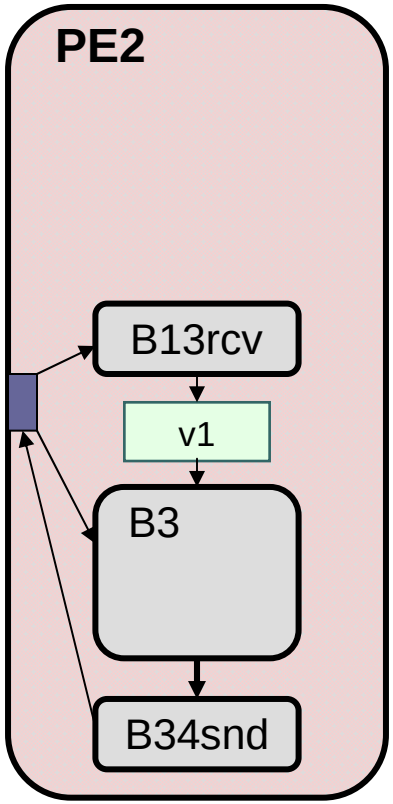
Model after Protocol Insertion (cont.)

- Leaf behaviors

```
1  behavior B13Rcv( IBusSlave bus1 , out int v1 )
   {
     void main(void) {
       bus1.slaveRecv( CB13, &v1, sizeof(v1) );
5    }
   };
```

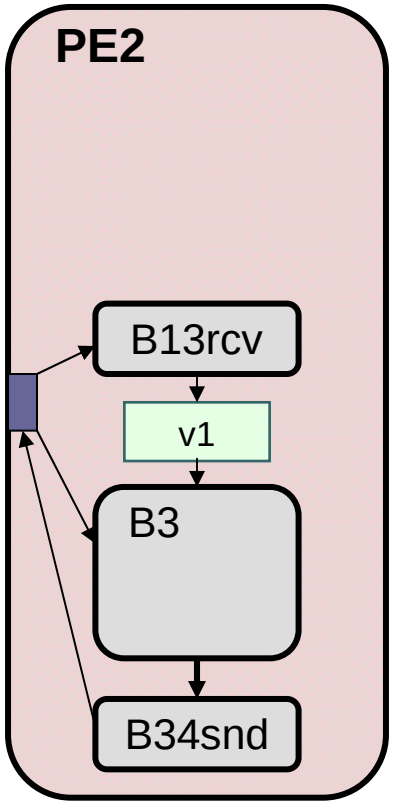
```
1  behavior B3( in int v1, IBusSlave bus1 )
   {
     void main(void) {
       …
5      bus1.slaveRecv( C2, … );
       …
     }
   };
```

```
1  behavior B34Snd( IBusSlave bus1 )
   {
     void main(void) {
       bus1.slaveSend( CB34, 0, 0 );
5    }
   };
```

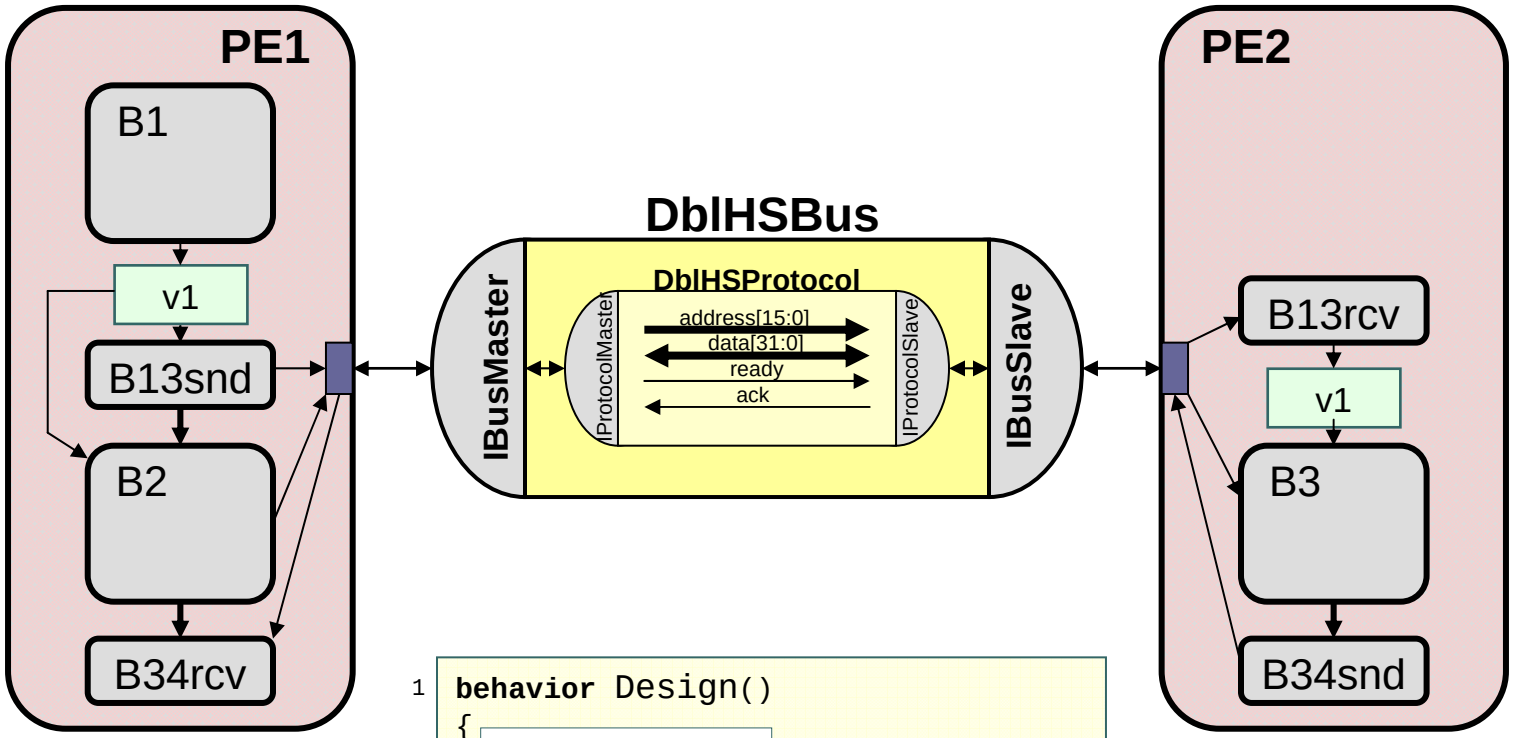


Model after Protocol Insertion (cont.)

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

5    B13Rcv b13rcv( bus1, v1 );
     B3     b3    ( v1, bus1 );
     B34Snd b34snd( bus1 );
10

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

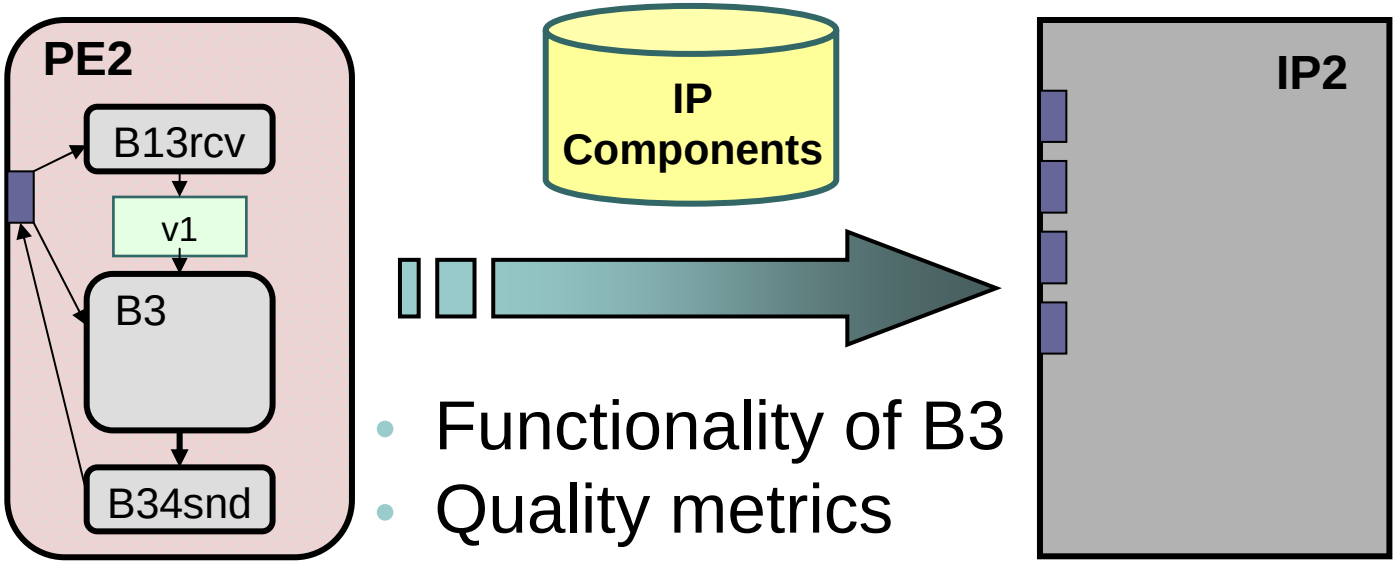


Model after Protocol Insertion (cont.)



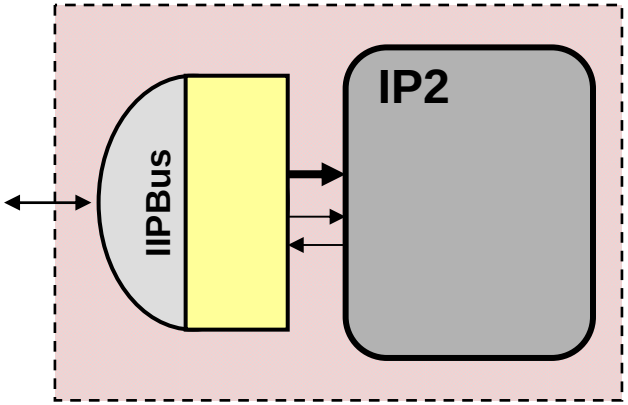

```
1  behavior Design()
   {
     DblHSBus bus1;

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

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

Intellectual Property (IP)



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

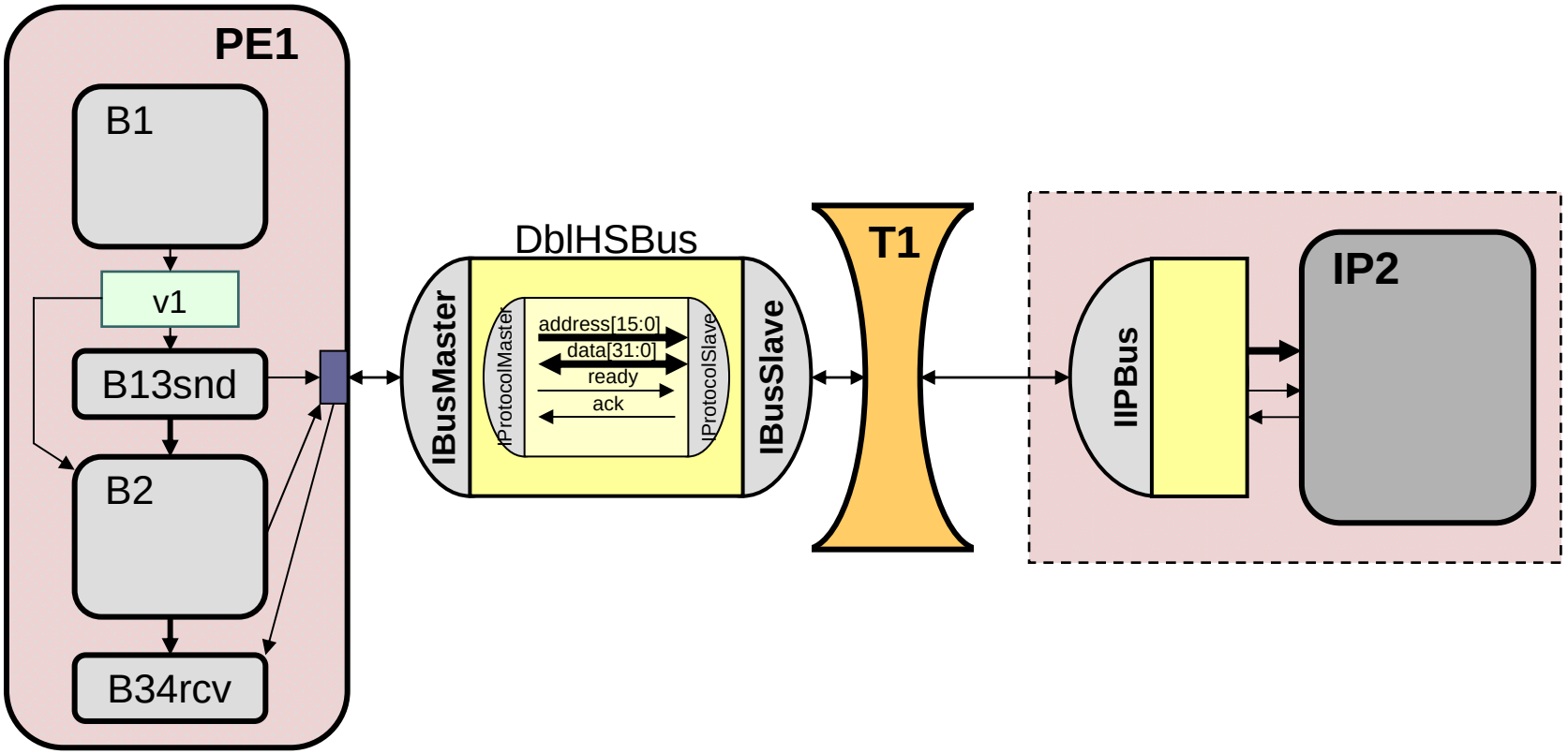
IP Component Model



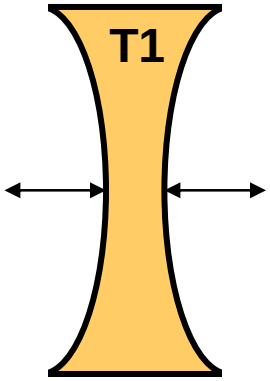
- Component behavior
  - Simulation, synthesis
- Wrapper
  - Encapsulate fixed IP protocol
  - Provide canonical interface

```
1  interface IIPBus
   {
     void start( int v1, … );
     void done( void );
5  };
```

Transducer Insertion



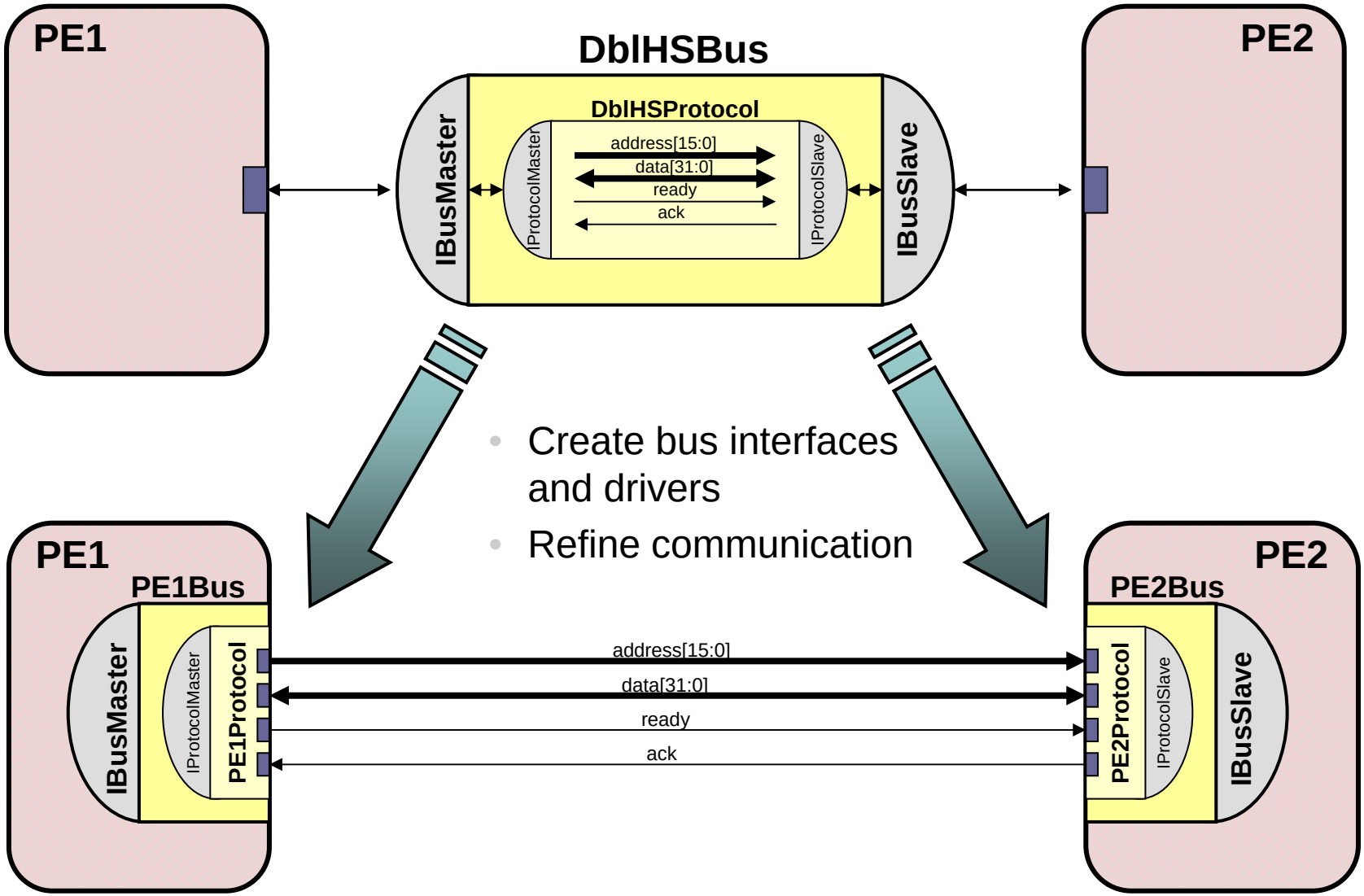
Transducer



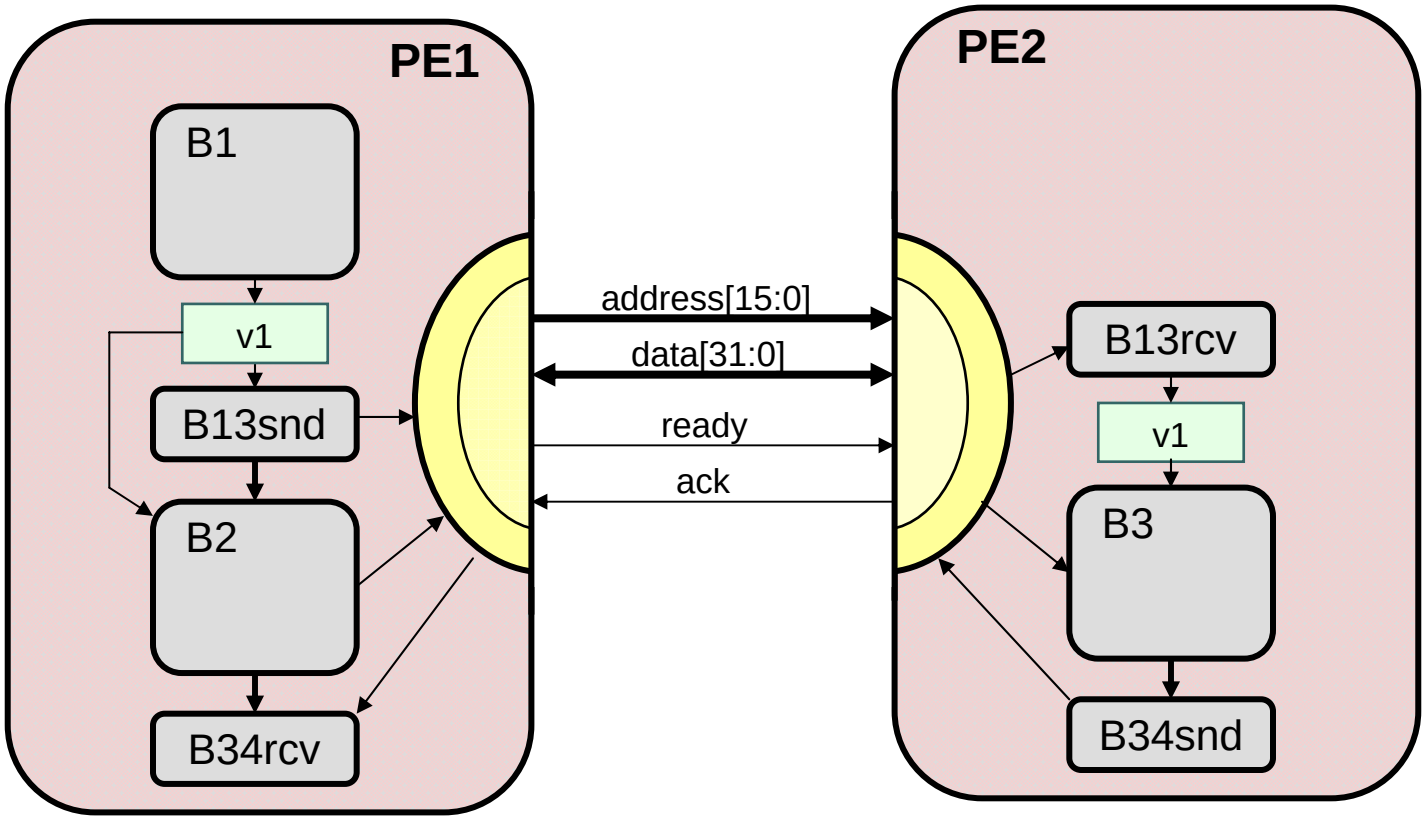

```
1  behavior T1( IBusSlave bus,
                IIPbus ip )
   {
     int v1, … ;
5
     void main(void)
     {
       bus.slaveReceive( CB13, &v1, sizeof(v1) );
       bus.slaveReceive( C2, … );
10     ip.start( v1, … );
       ip.done();
       bus.slaveSend( CB34, 0, 0 );
     }
   };
```

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

Inlining

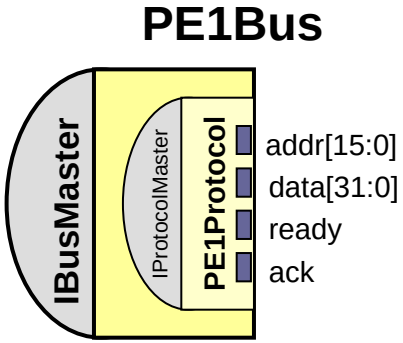

PE1
DblHSBus
DblHSProtocol
IBusMaster
IProtocolMaster
address[15:0]
data[31:0]
ready
ack
IProtocolSlave
IBusSlave
PE2
Create bus interfaces and drivers
Refine communication
PE1
PE1Bus
IBusMaster
IProtocolMaster
PE1Protocol
address[15:0]
data[31:0]
ready
ack
PE2
PE2Bus
PE2Protocol
IProtocolSlave
IBusSlave

Model after Inlining



Model after Inlining (cont.)

- PE1 bus driver



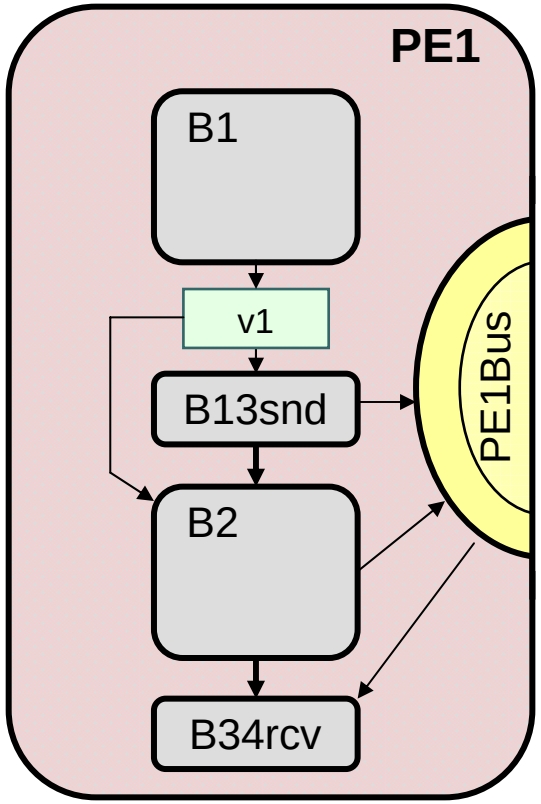
Application layer:

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

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

Protocol layer:

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

Model after Inlining (cont.)




```
1  behavior PE1( out   bit[15:0] addr,
                 inout bit[31:0] data,
                 OSignal         ready,
                 ISignal         ack      )
5  {
     PE1Bus bus1( addr, data, ready, ack );

     int v1;

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

15   void main(void) {
       b1.main();
       b13snd.main();
       b2.main();
       b34rcv.main();
20   }
};
```

Model after Inlining (cont.)

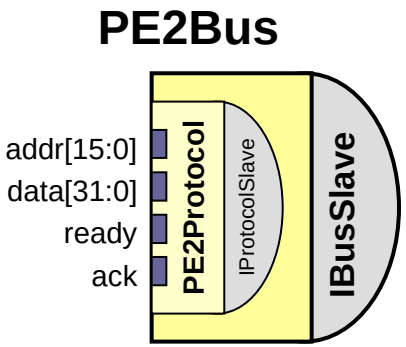
- PE2 bus interface

Application layer:

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

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

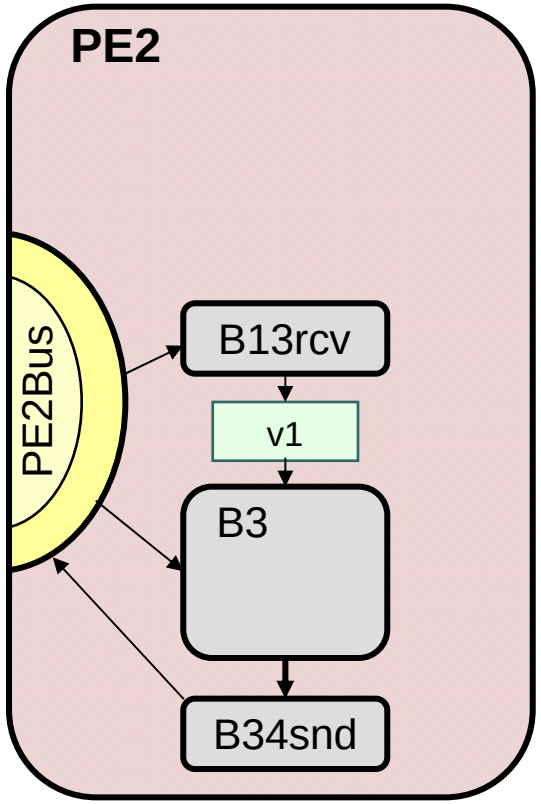
Protocol layer:

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

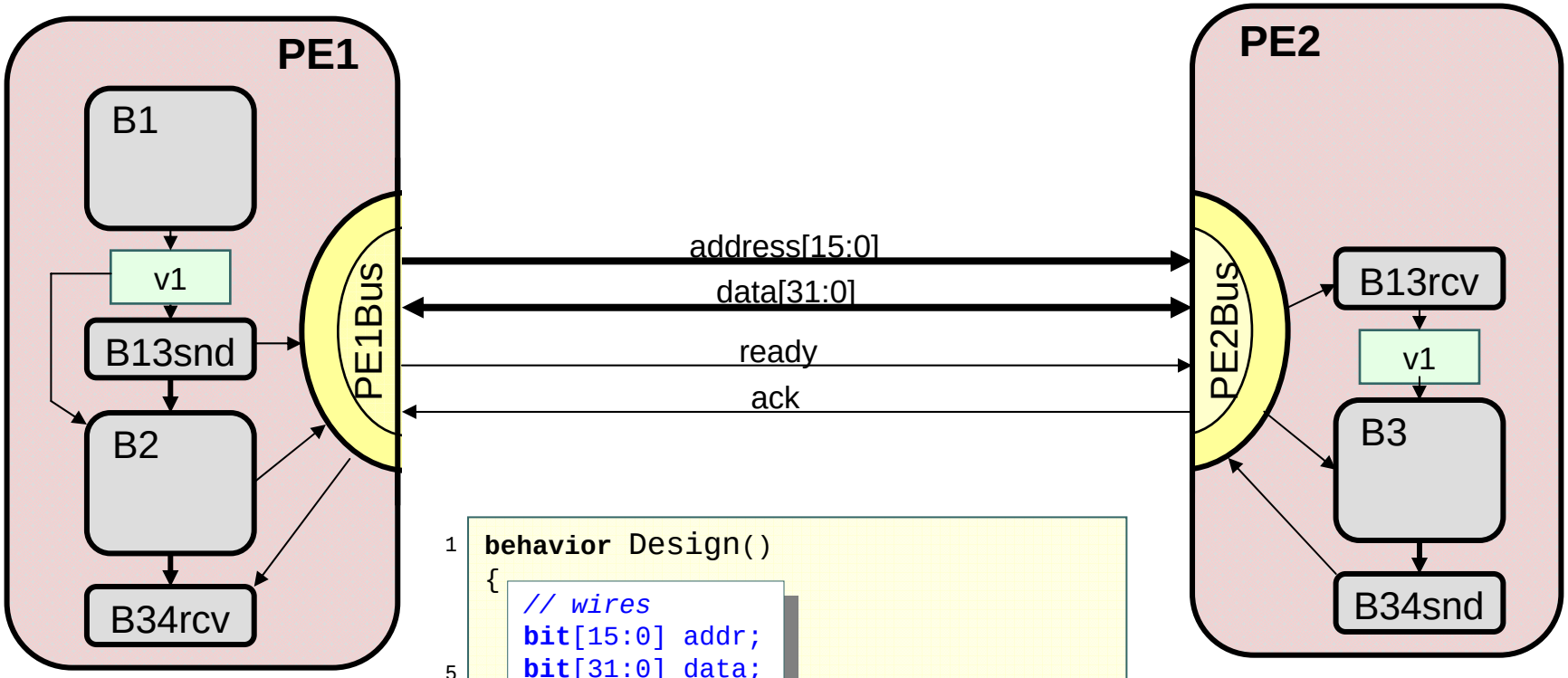


Model after Inlining (cont.)

```
1  behavior PE2(  in    bit[15:0] addr,
                  inout bit[31:0] data,
                  ISignal         ready,
                  OSignal         ack      )
5  {
     PE2Bus bus1( addr, data, ready, ack );

     int v1;

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

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



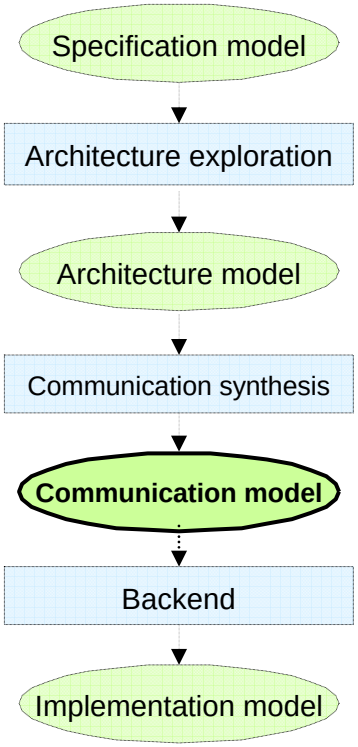
Model after Inlining (cont.)




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

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

     void main(void) {
       par { pe1.main(); pe2.main(); }
15   }
   };
```

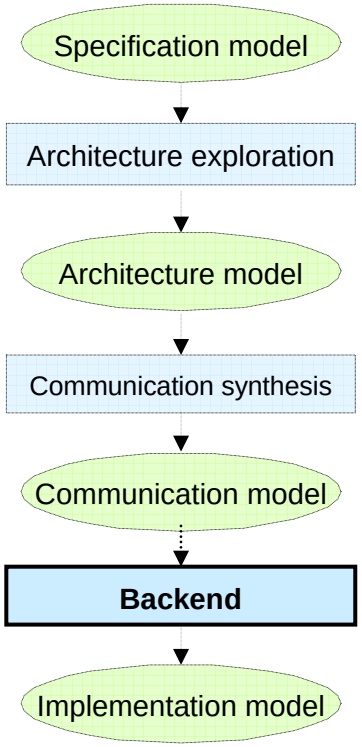
Communication Model

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

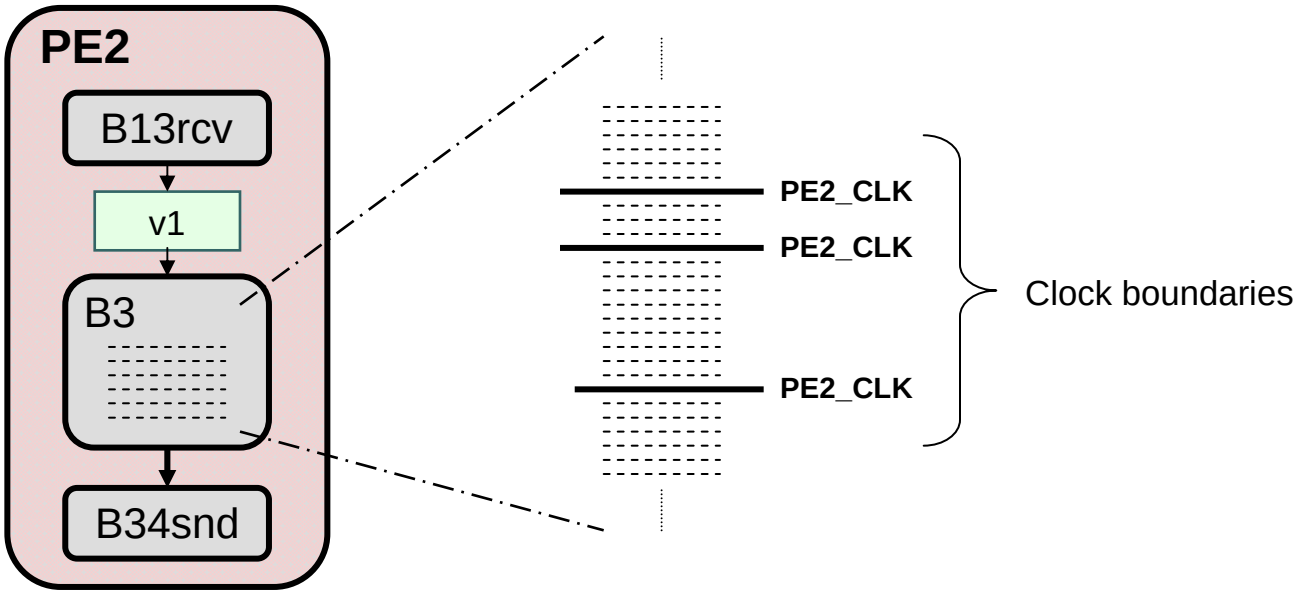


Backend

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



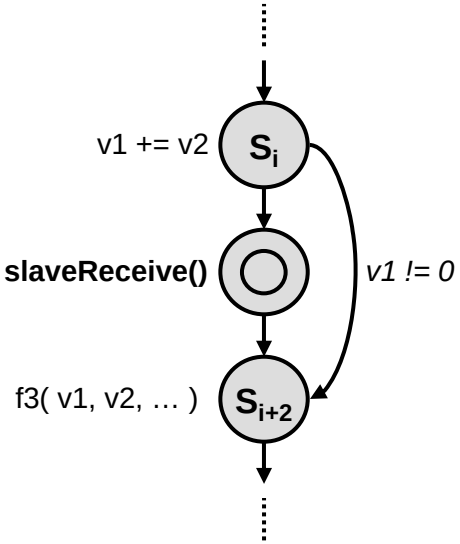
Hardware Synthesis



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

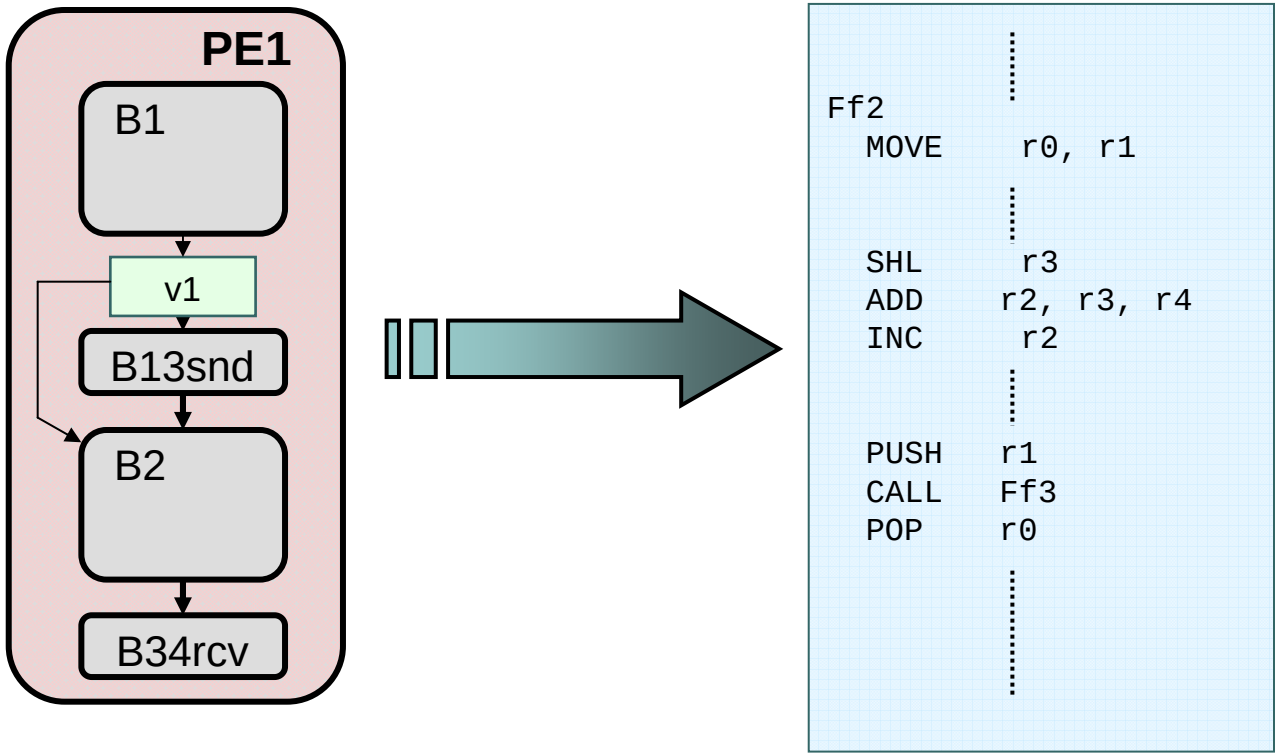
Scheduled Hardware Model

Hierarchical FSMD:



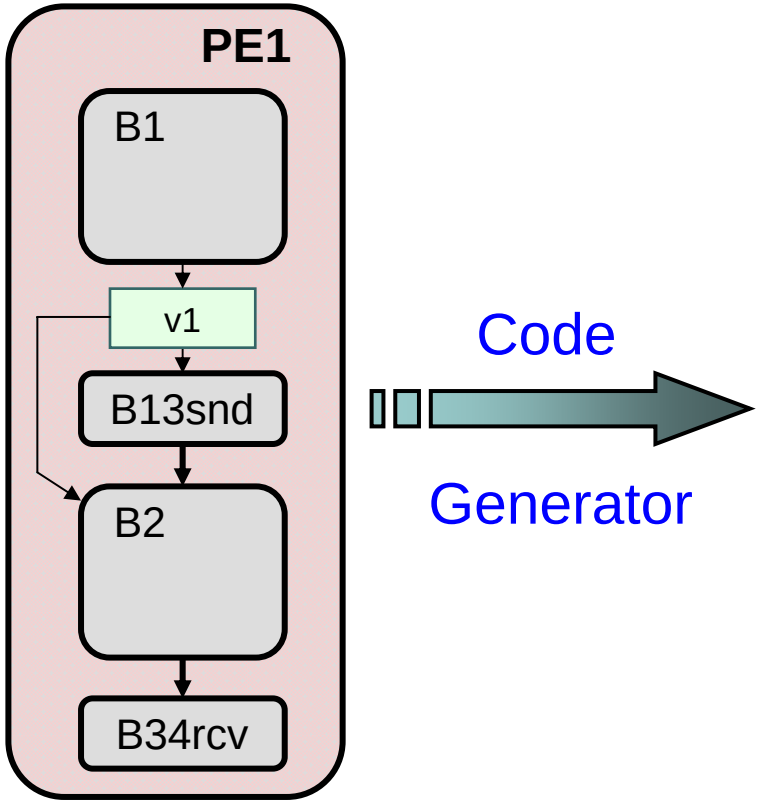

```
1  behavior B3( in int v1, IBusSlave bus )
   {
     void main(void) {
       enum { S0, S1, S2, …, Sn } state = S0;
5
       while( state != Sn )
       {
         waitfor( PE2_CLK );  // wait for clock period

10       switch( state ) {
           …
           case Si:
             v1 += v2;  // data-path operations
             if( v1 )
15             state = Si+1;
             else
               state = Si+2;
             break;
           case Si+1:  // receive message
20           bus.slaveReceive( C2, … );
             state = Si+2;
             break;
           case Si+2:
             f3( v1, v2, … );
25           state = Si+3;
             break;
           …
       }}
     }
30 };
```

Software Synthesis



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

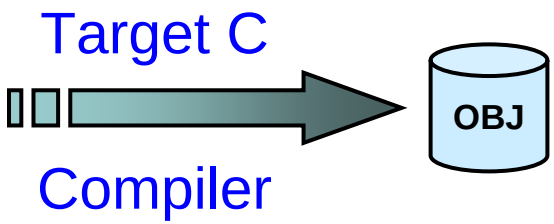
Code Generation



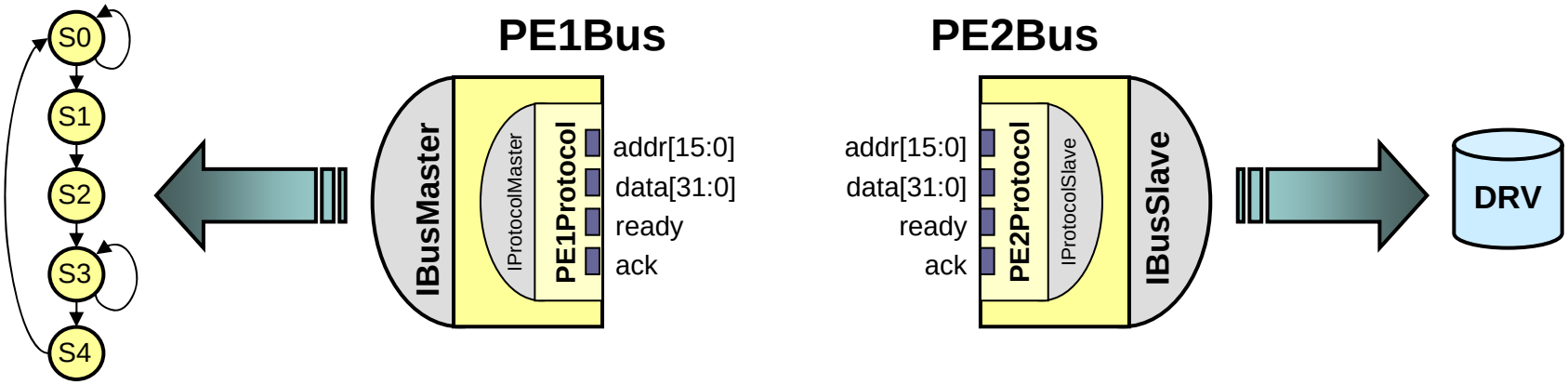

```
1  void B1( int *v1 ) {
     …
   }

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

   void B2( int v1 ) {
10   …
     masterSend( C2, );
     …
   }

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

   void main(void)
20 {
     int v1;

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

Compilation

```
1  void b1( int *v1 ) {
     …
   }
   void b13snd( int v1 ) {
5    masterSend( CB13, &v1,
                 sizeof(v1) );
   }
   void b2( int v1 ) {
     …
10   masterSend( C2, );
     …
   }
   void b34rcv() {
     masterReceive( CB34, 0, 0 );
15 }

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



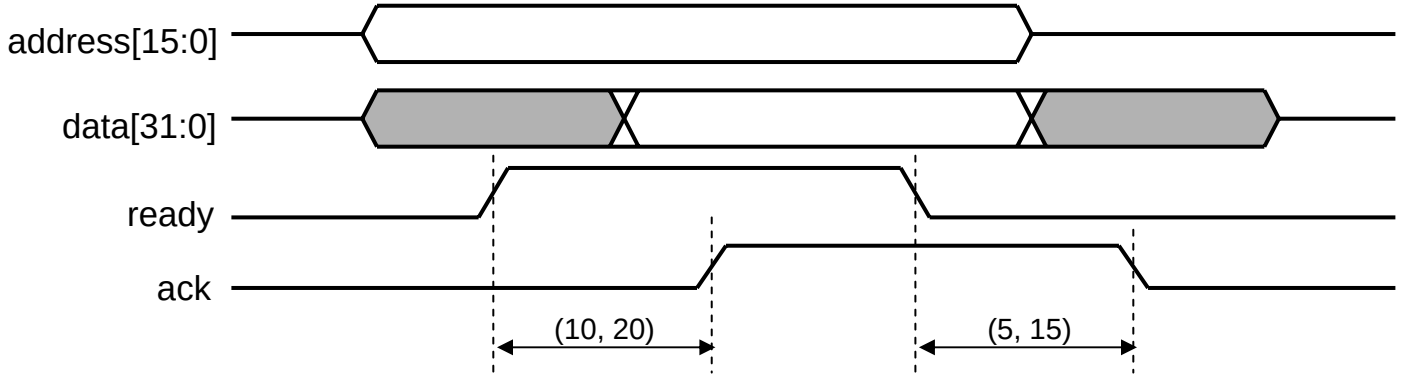
Interface Synthesis



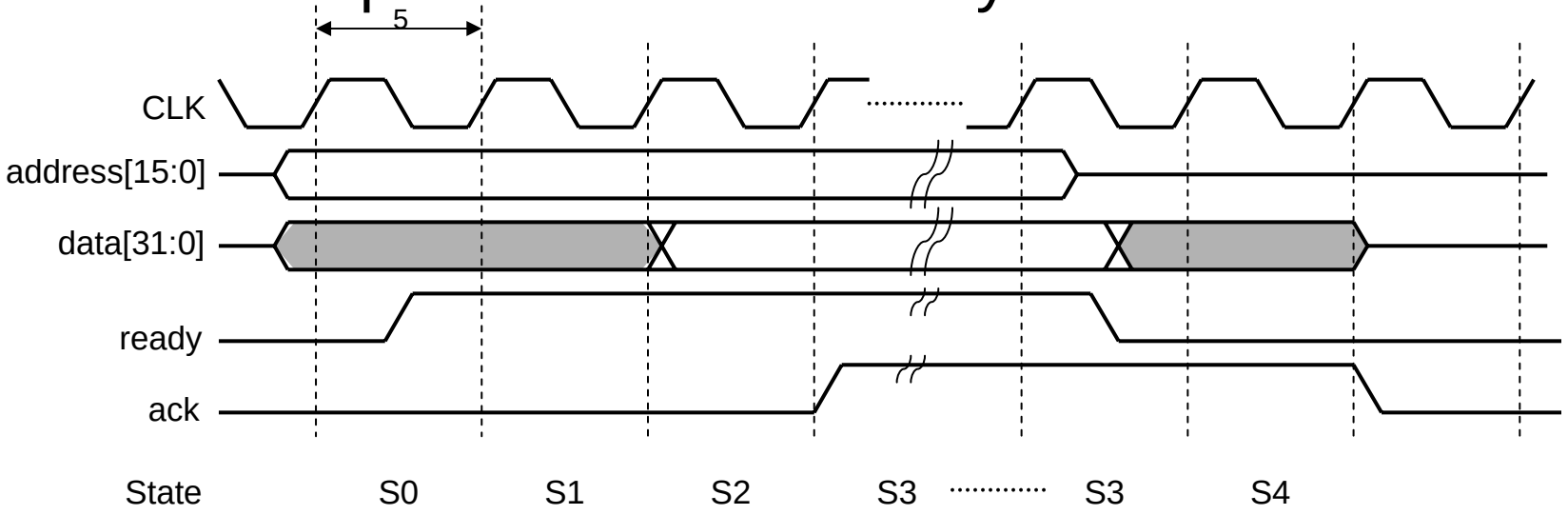
- Implement communication on components
  - Hardware bus interface logic
  - Software bus drivers

Hardware Interface Synthesis

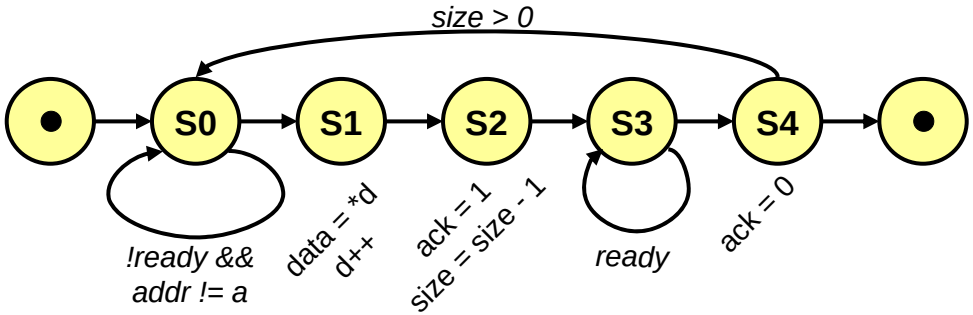
- Specification: timing diagram / constraints

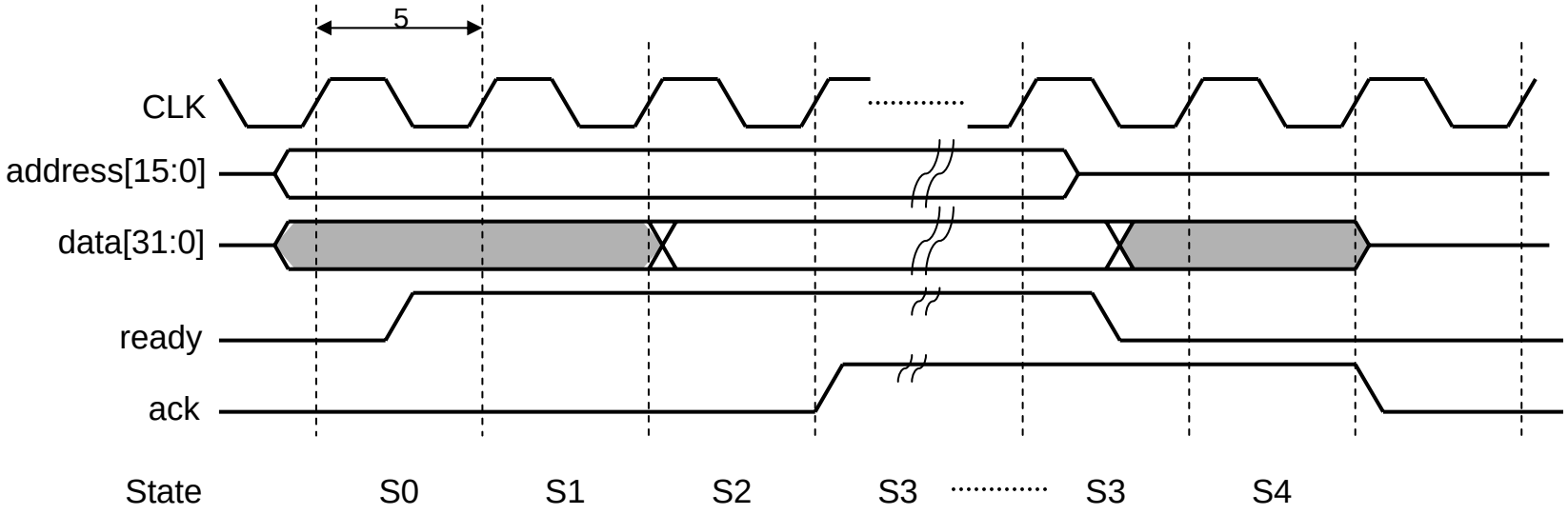


- FSMD implementation: clock cycles



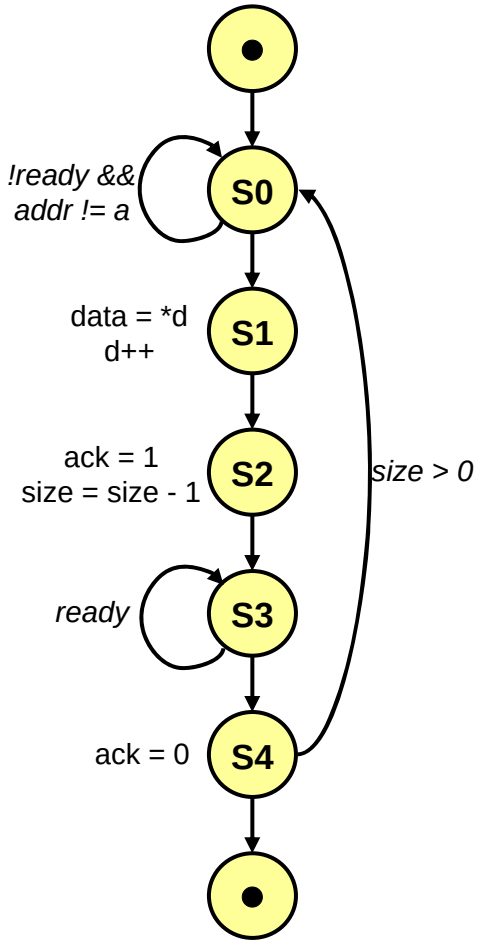
Hardware Interface FSMD





Hardware Interface FSMD (cont.)

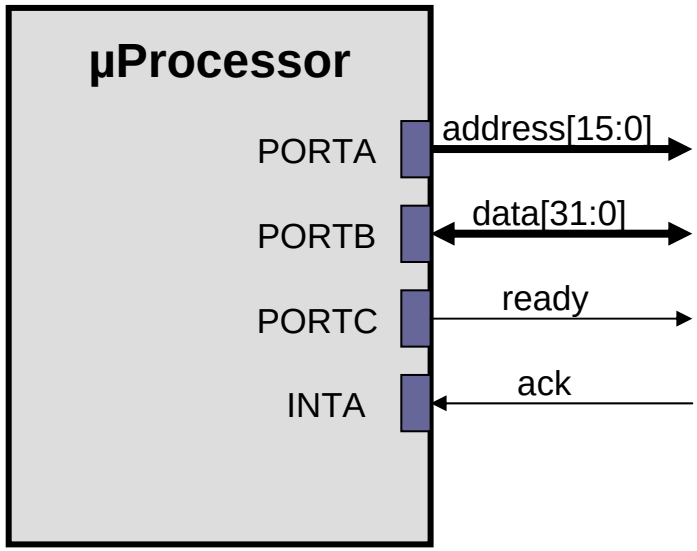
```
1  channel PE2Bus( in bit[15:0] addr, inout bit[31:0] data,
                   ISignal ready, OSignal ack )
     implements IBusSlave
   {
5    void slaveSend( int a, void* d, int size ) {
       enum { S0, S1, S2, S3, S4, S5 } state = S0;
       while( state != S5 ) {
         waitfor( PE2_CLK );        // wait for clock period
         switch( state ) {
10           case S0:  // sample ready, address
               if( (ready.val() == 1) && (addr == a) ) state = S1;
               break;
             case S1:  // read memory, drive data bus
               data = *( ((long*)d)++ );
15             state = S2;
               break;
             case S2: // raise ack, advance counter
               ack.set( 1 );
               size--;
20             state = S3;
               break;
             case S3: // sample ready
               if( ready.val() == 0 ) state = S4;
               break;
             case S4: // reset ack, loop condition
25             ack.set( 0 );
               if( size != 0 ) state = S0 else state = S5;
       }}
     }
30   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



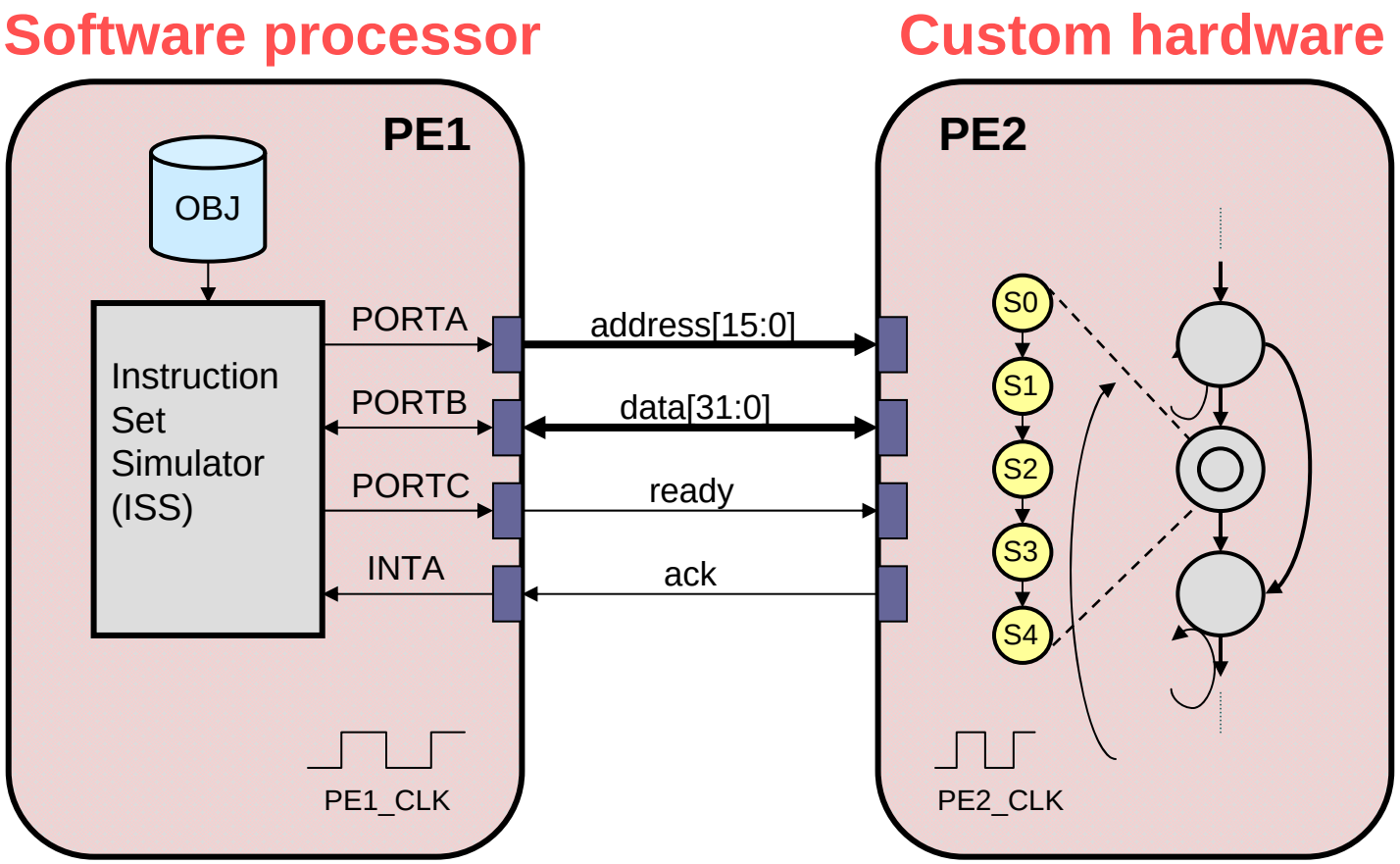
Bus driver:

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

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

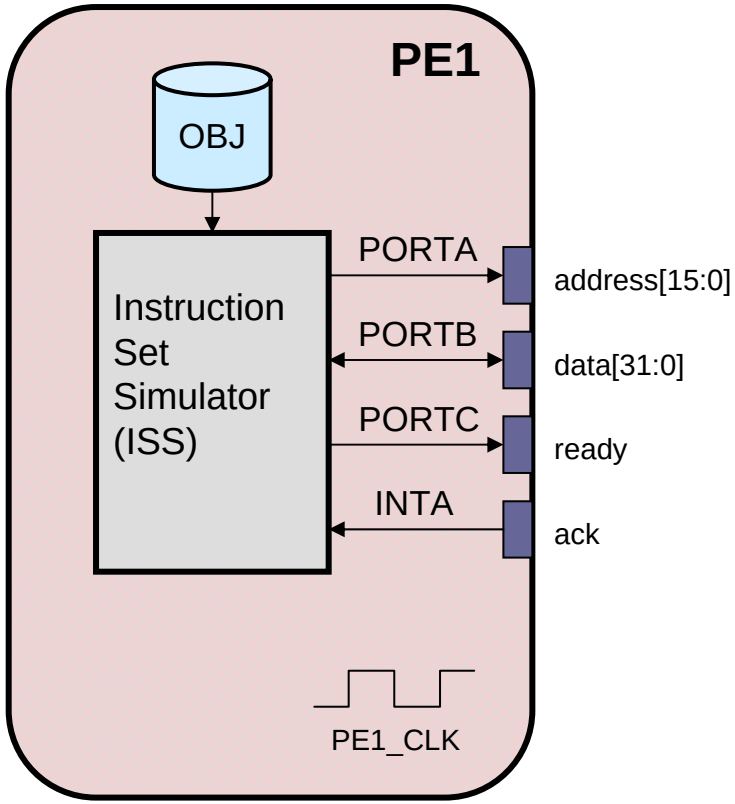
Interrupt handler:

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

Implementation Model



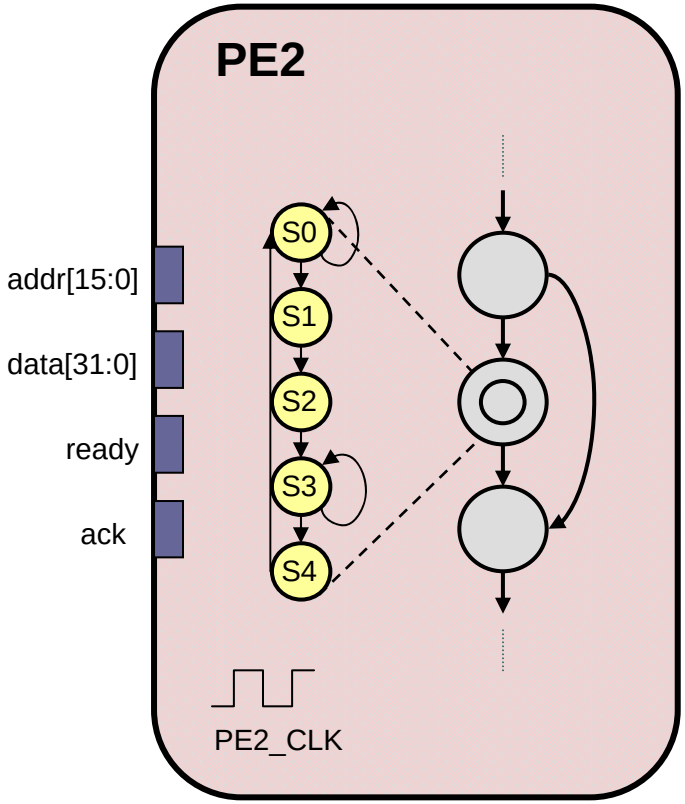
Implementation Model (cont.)



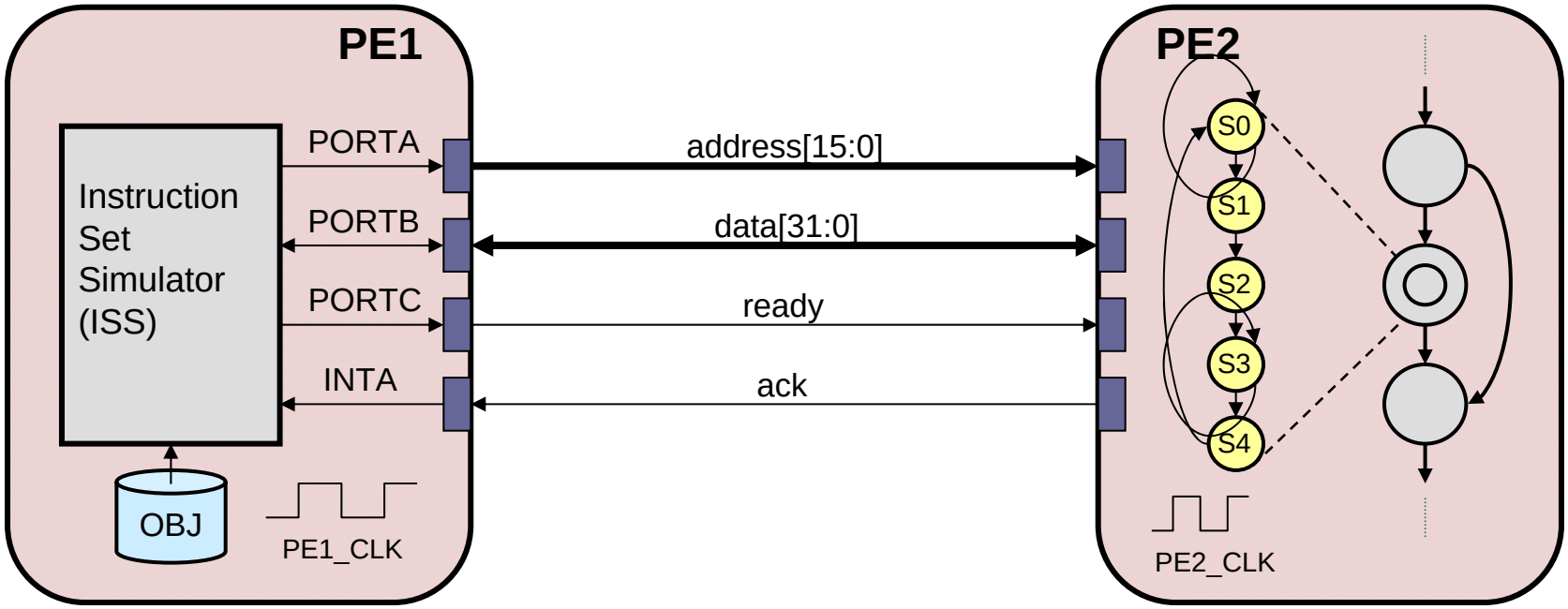

```
1  #include <iss.h>    // ISS API

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

       // run simulation
15     for( ; ; ) {
         // drive inputs
         iss.porta = addr;
         iss.portb = data;
         iss.inta  = ack.val();
20
         // run processor cycle
         iss.exec();
         waitfor( PE1_CLK );

25       // update outputs
         data = iss.portb;
         ready.set( iss.portc & 0x01 );
       }
     }
30 };
```

Implementation Model (cont.)

```
1  behavior PE2( in    bit[15:0] addr,
                 inout bit[31:0] data,
                 ISignal         ready,
                 OSignal         ack )
5  {
     // Interface FSM
     PE2Bus bus1( addr, data, ready, ack );

     int v1;                     // memory
10
     B13Rcv b13rcv( bus1, v1 );  // FSMDs
     B3     b3    ( v1, bus1 );
     B34Snd b34snd( bus1 );

15   void main(void)
     {
       b13rcv.main();
       b3.main();
       b34snd.main();
20   }
   };
```



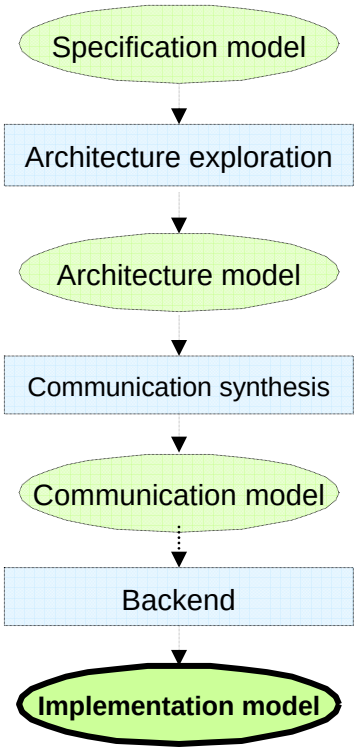
Implementation Model (cont.)




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

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

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

Implementation Model (cont.)

- 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



Summary and Conclusions

- SpecC system-level design methodology & language
  - Four levels of abstraction
    - Specification model: untimed, functional
    - Architecture model: estimated, structural
    - Communication model: timed, bus-functional
    - Implementation model: cycle-accurate, RTL/IS
  - Specification to RTL
    - System synthesis
      1. Architecture exploration (map computation to components)
      2. Communication synthesis (map communication to busses)
    - Backend
      3. hardware synthesis, software compilation, interface synthesis
  - Well-defined, formal models & transformations
    - Automatic, gradual refinement
    - Executable models, testbench re-use
    - Simple verification