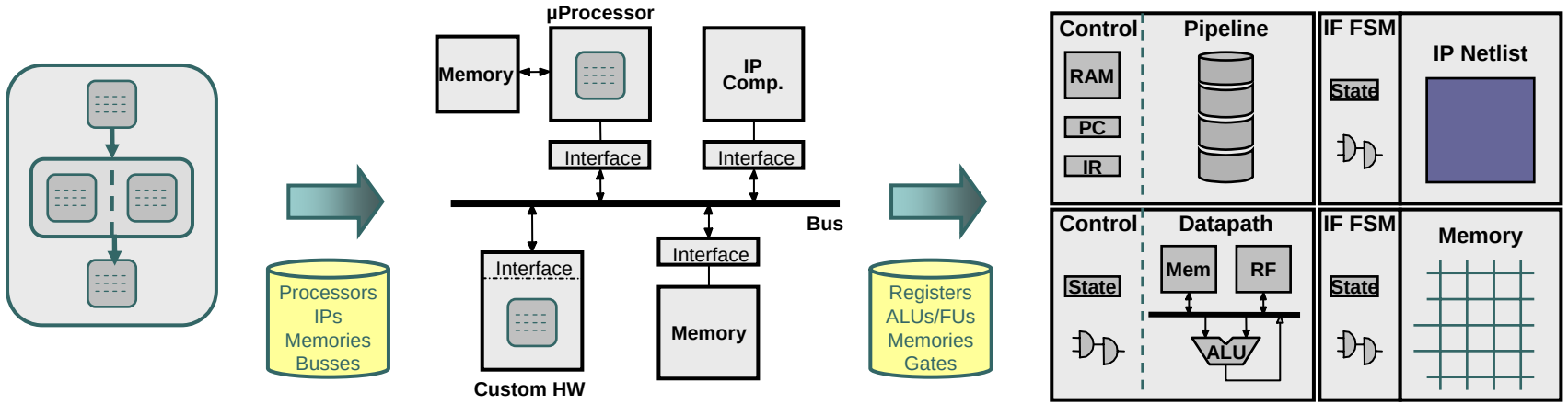
# CprE 588
# Embedded Computer Systems

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

Lecture #5 – System-Level Design with SpecC

# System-On-Chip Design




**Specification + constraints**

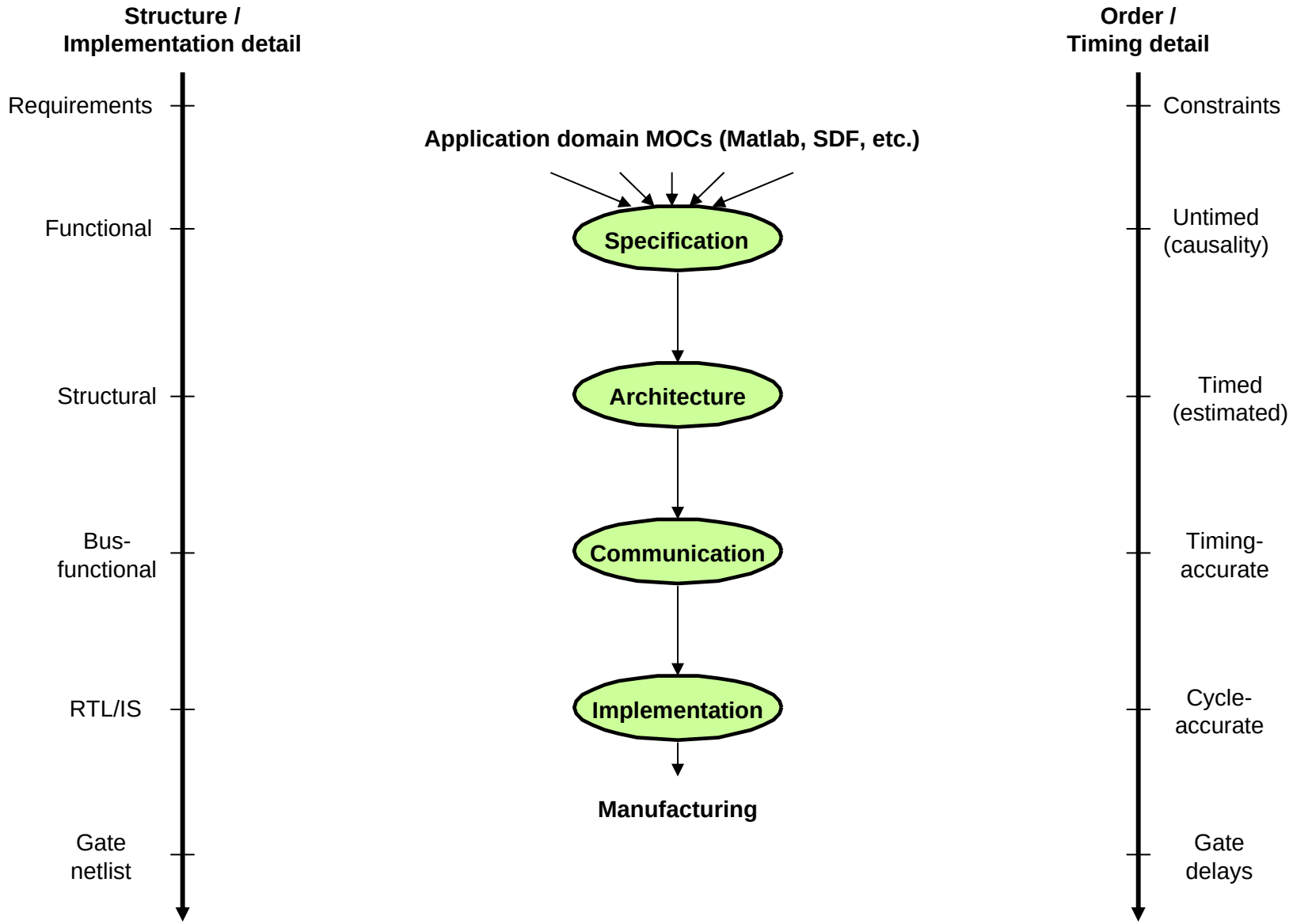
**System architecture + estimates**

**RTL/IS Implementation + results**

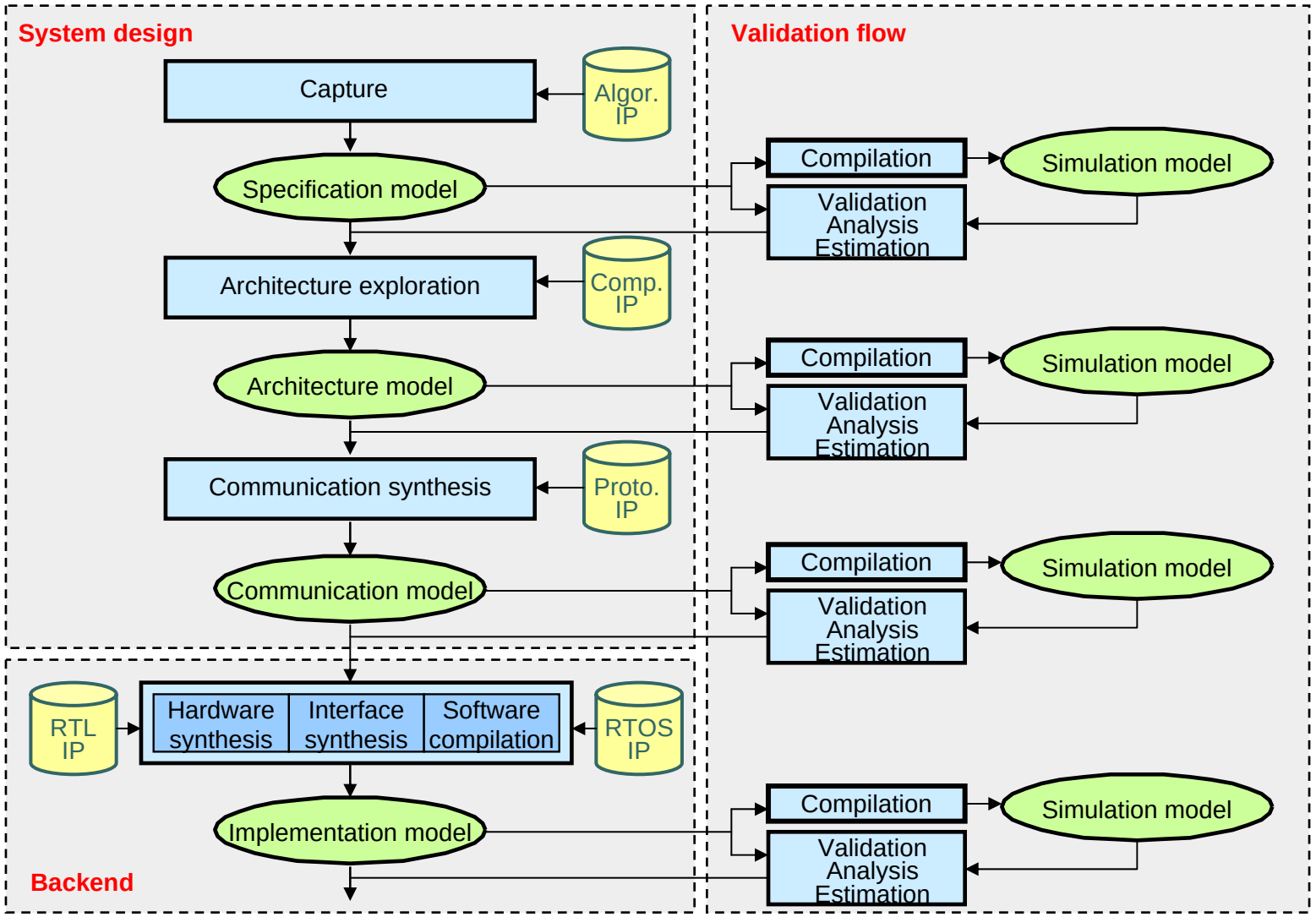
- Specification to architecture to implementation
- Behavior to structure
  1. System level: system specification to system architecture
  2. RT/IS level: component behavior to component microarchitecture

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

# Abstraction Levels

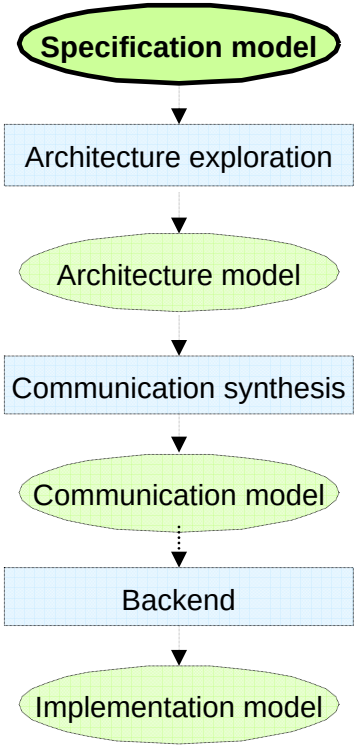


# SpecC Methodology

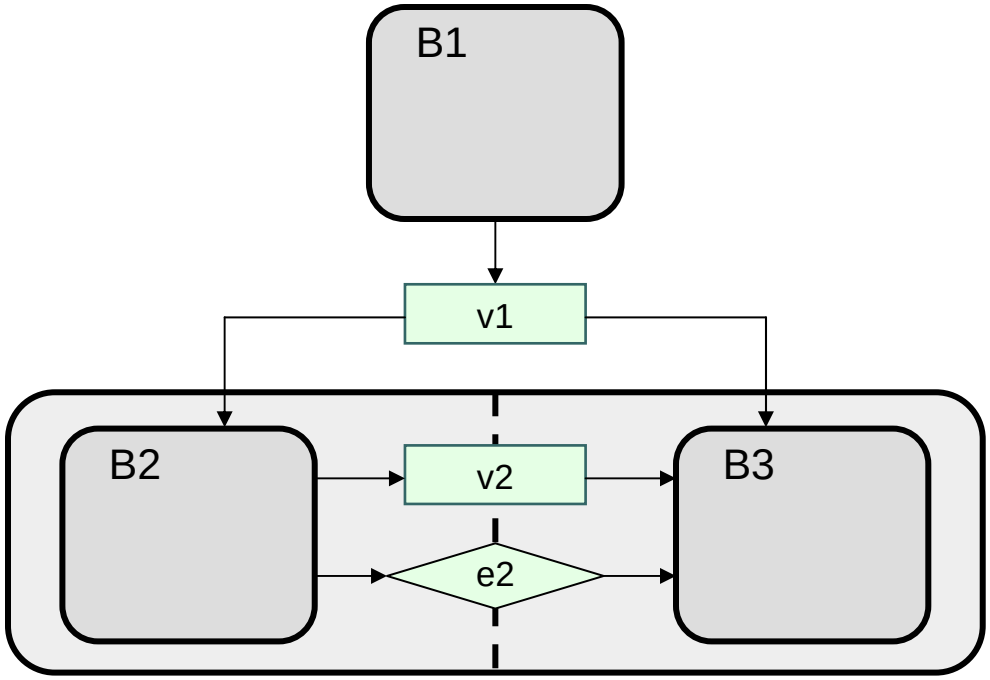


# Specification Model

- 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

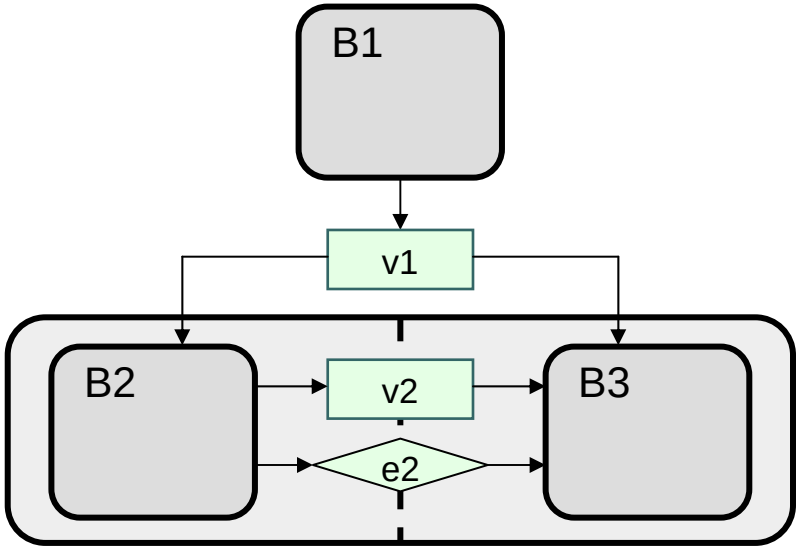


# Specification Model Example




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

# Specification Model Example (cont.)

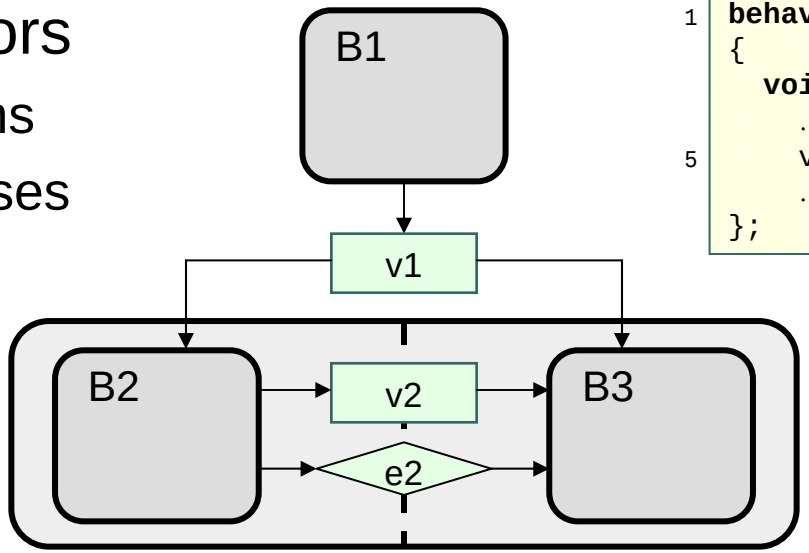



SpecC design hierarchy:

```
1  behavior B2B3( in int v1 )
   {
     int   v2;              // variables
     event e2;
5
     B2 b2( v1, v2, e2 );  // children
     B3 b3( v1, v2, e2 );

     void main(void) {
10     par {
         b2.main();
         b3.main();
       }
   };
15
   behavior Design()
   {
     int v1;                // variables

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

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

# Specification Model Example (cont.)

- Leaf behaviors
  - C algorithms
  - Port accesses



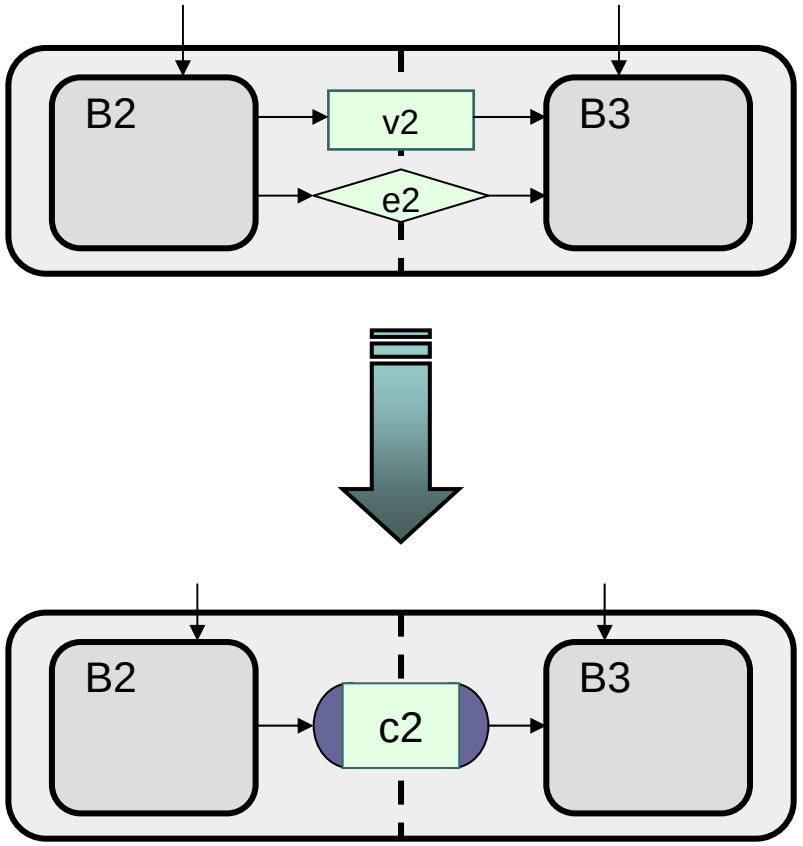

```
1  behavior B1( out int v1 )
   {
     void main(void) {
       …
5      v1 = …    // write v1
       …
   };
```

```
1  behavior B2( in  int   v1,
                out int   v2,
                out event e2 )
   {
5     void main(void) {     // read v1
         …
         v2 = f2( v1, … ); // produce v2
         notify( e2 );     // synchronize
         …
10     }
   };
```

```
1  behavior B3( in int   v1,
                in int   v2,
                in event e2 )
   {
5     void main(void) {    // read v1
         …
         wait( e2 );       // wait for sync
         f3( v1, v2, … ); // consume v2
         …
10     }
   };
```

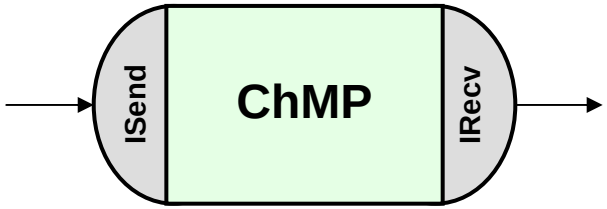
# Communication

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



# Message-Passing Channel

- Blocking, unbuffered message-passing



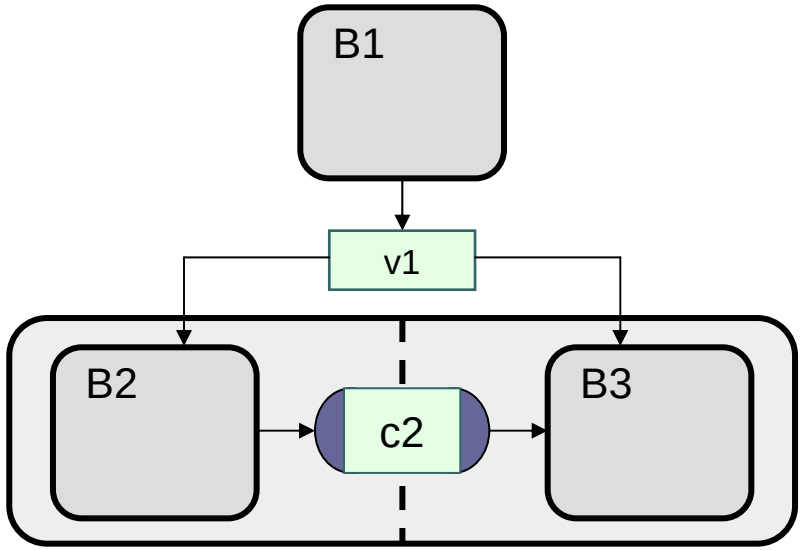

Channel interfaces:

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

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

Simulation model:

```
1  channel ChMP() implements ISend, IRecv
   {
     void  *buf = 0;
     event eReady, eAck;
5
     void send( void *d, int size ) {
       buf = malloc( size );
       memcpy( buf, d, size );
       notify( eReady );
10     wait( eAck );
       free( buf );
       buf = 0;
     }

15   void recv( void *d, int size ) {
       if( !buf ) wait( eReady );
       memcpy( d, buf, size );
       notify( eAck );
     }
20 };
```

# Message-Passing Specification



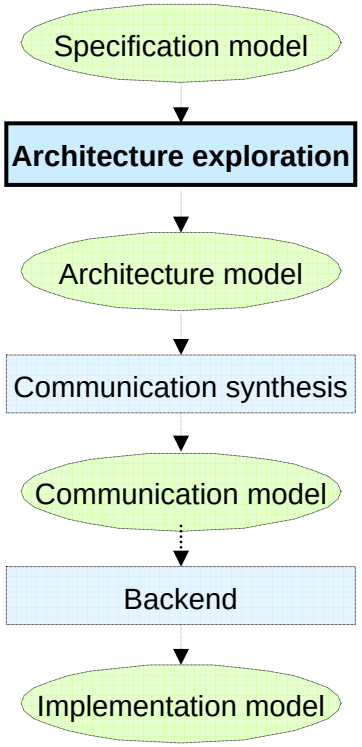

```
1  behavior B2B3( in int v1 )
   {
     ChMP c2;

5    B2   b2( v1, c2 );

     B3   b3( v1, c2 );

     void main(void) {
10     par {
         b2.main();
         b3.main();
       }
     }
15 };
```

```
1  behavior B2( in int v1, ISend c2 )
   {
     void main(void) {
       …
5      v2 = f2( v1, … );
       c2.send( &v2, sizeof(v2) );
       …
     }
10 };
```

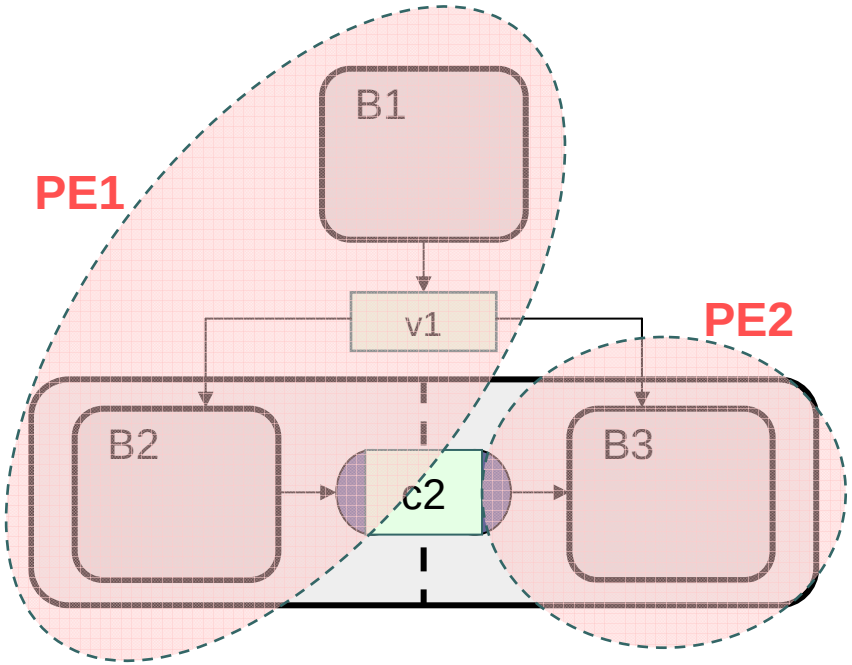
```
1  behavior B3( in int v1, IRecv c2 )
   {
     void main(void) {
       …
5      c2.recv( &v2, sizeof(v2) );
       f3( v1, v2, …);
       …
     }
10 };
```

# Architecture Exploration

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



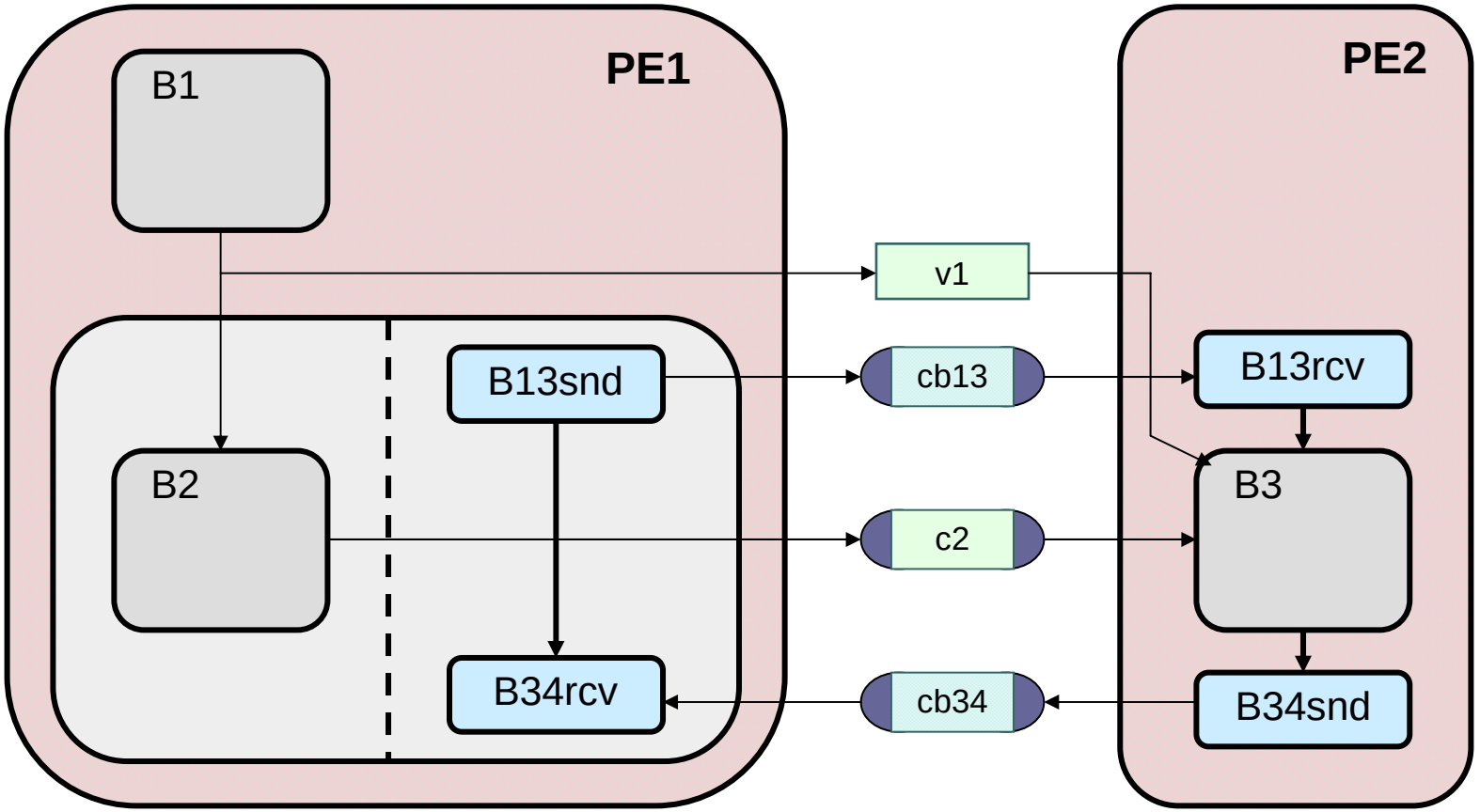
# Allocation, Behavior Partitioning



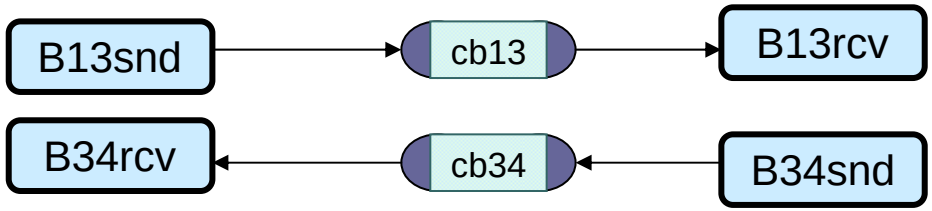

- Allocate PEs
- Partition behaviors
- Globalize communication

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

# Model after Behavior Partitioning



# Synchronization



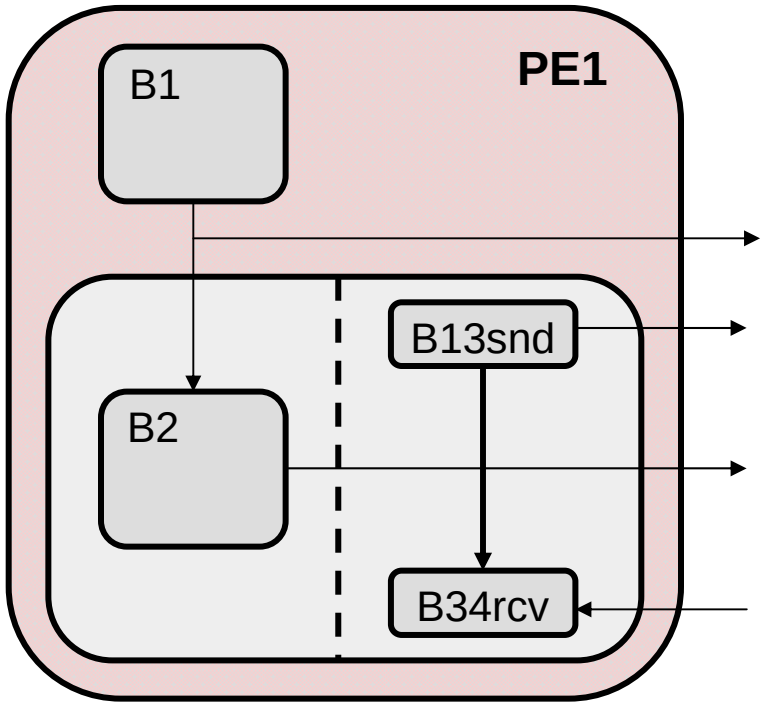

- For each component-crossing transition
  - Synchronization behavior pair
  - Synchronize over blocking message-passing channel

```
1  behavior BSnd( ISend ch )
   {
     void main(void) {
       ch.send( 0, 0 );
5    }
   };
```

```
1  behavior BRcv( IRecv ch )
   {
     void main(void) {
       ch.recv( 0, 0 );
5    }
   };
```

➢ **Preserve execution semantics**

# After Behavior Partitioning



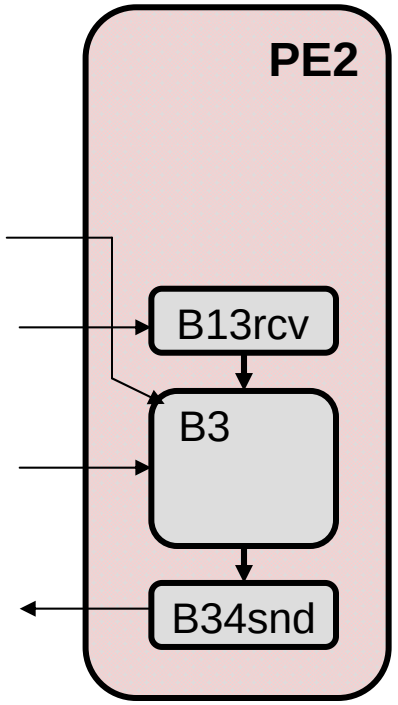

```
1  behavior B3Stub( ISend cb13, IRecv cb34 )
   {
     BSnd b13snd( cb13 );
     BRcv b34rcv( cb34 );
5
     void main(void) {
       b13snd.main();
       b34rcv.main();
     }
10 };
```

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

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

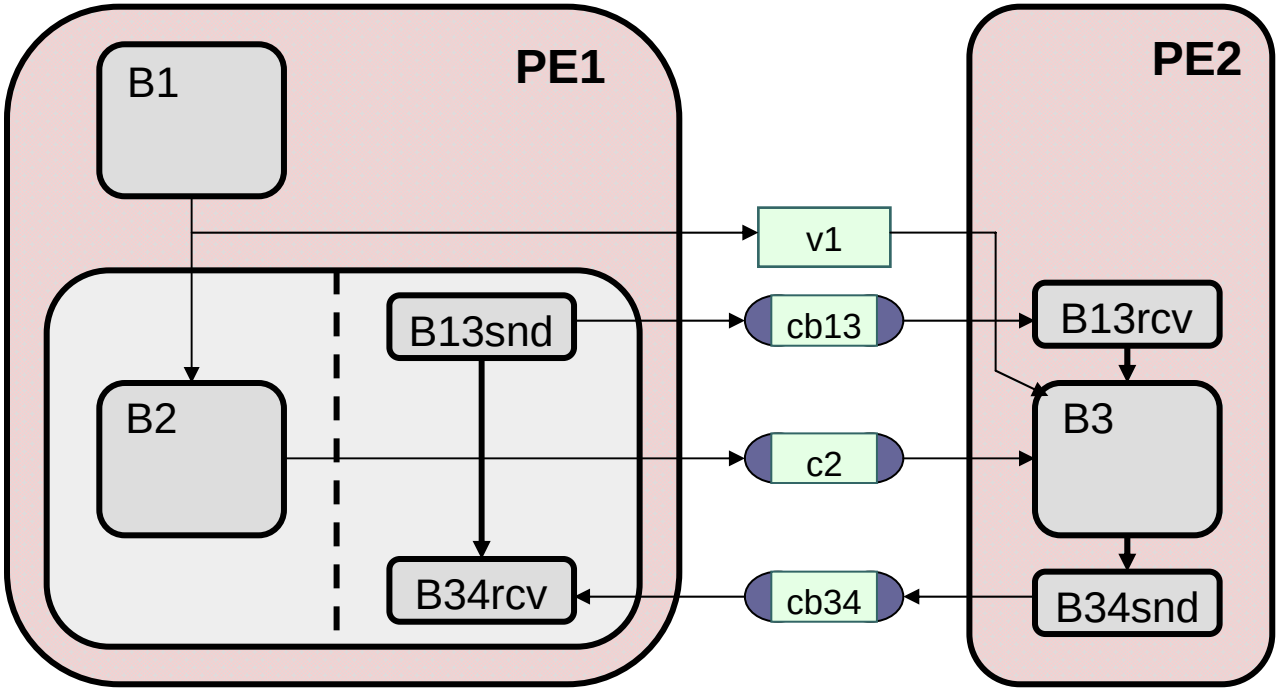
```
1  behavior B2B3( in int v1,
                  ISend cb13,
                  ISend c2,
                  IRecv cb34 )
   {
5    B2     b2    ( v1, c2 );
     B3Stub b3stub( cb13, cb34 );

     void main(void) {
10     par {
         b2.main();
         b3stub.main();
       }
     }
15 };
```

# After Behavior Partitioning (cont.)

```
1  behavior PE2( in int v1,
                 IRecv cb13,
                 IRecv c2,
                 ISend cb34)
5  {
     BRcv b13rcv( cb13 );
     B3   b3    ( v1, c2 );
     BSnd b34snd( cb34 );

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



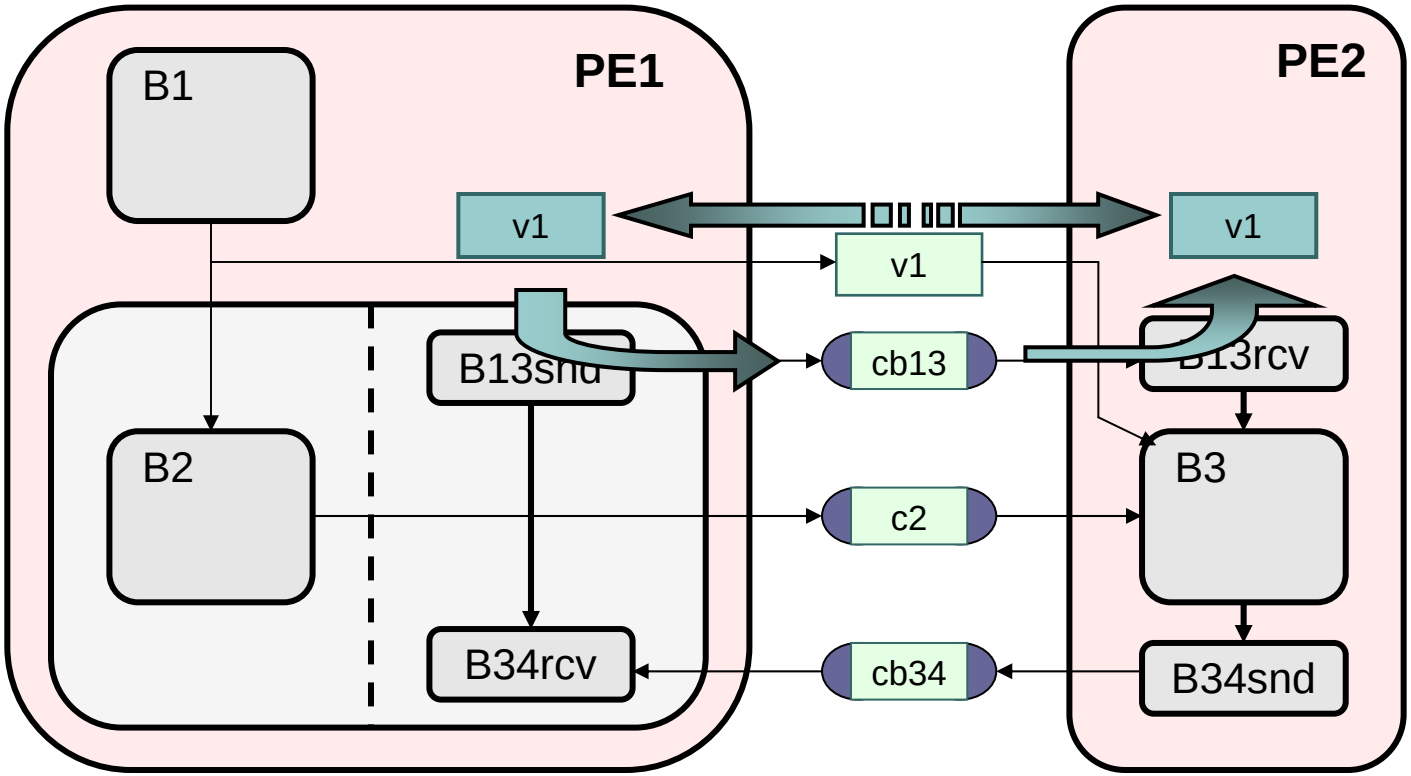
# After Behavior Partitioning (cont.)



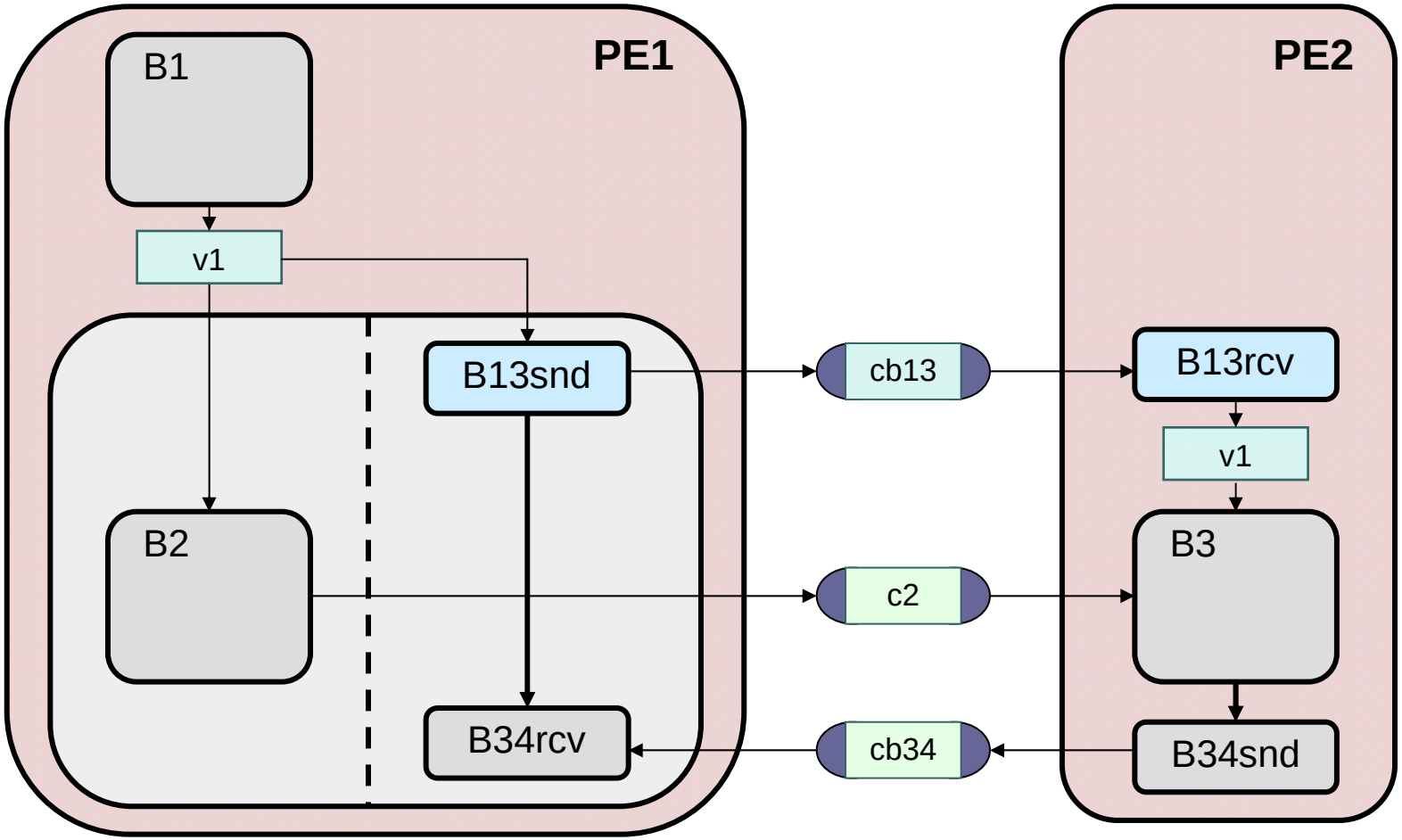

```
1  behavior Design() {
     int v1;
     ChMP cb13, c2, cb34;

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

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

# Variable Partitioning

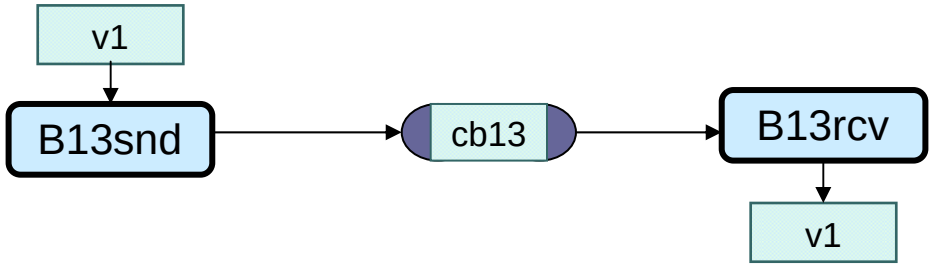
- Shared memory vs. message passing implementation
  - Map global variables to local memories
  - Communicate data over message-passing channels



# Message-Passing Model



# Message-Passing Communication



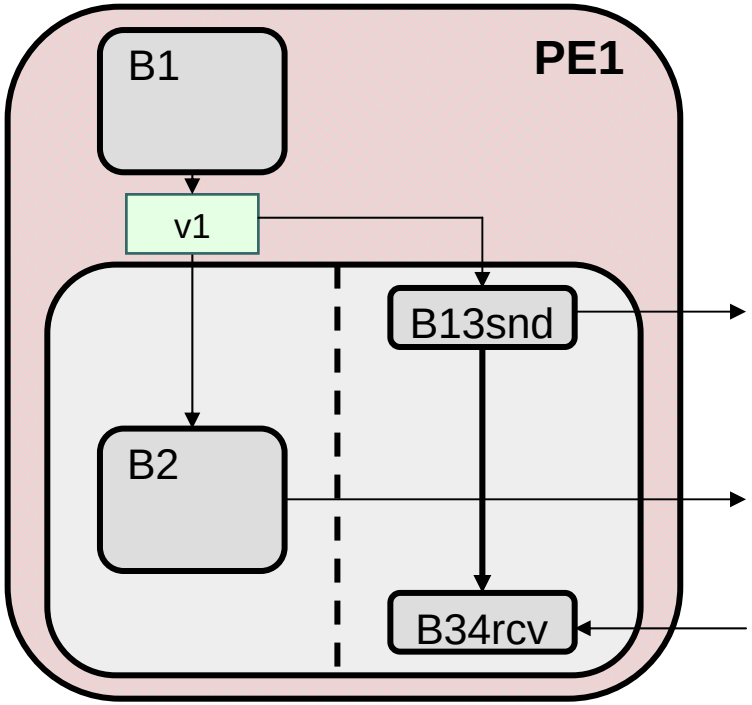

- Keep local variable copies in sync
  - Communicate updated values at synchronization points
  - Transfer control & data over message-passing channel

```
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**

# Message-Passing Model



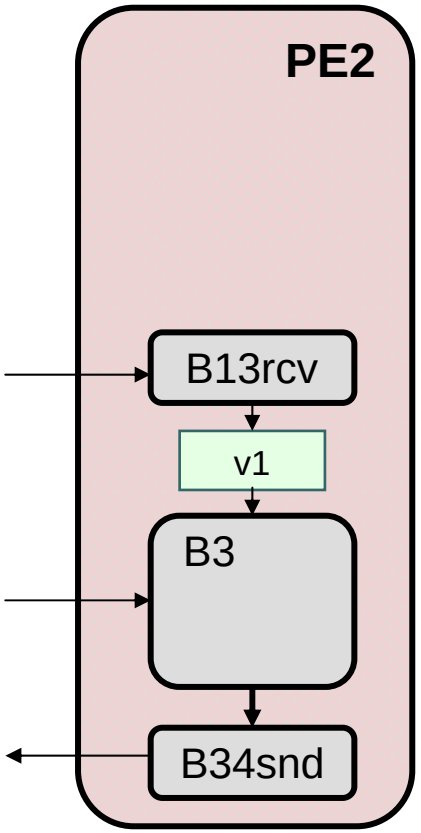

```
1  behavior PE1( ISend cb13,
                 ISend c2,
                 IRecv cb34)
   {
5    int v1;

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

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

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

10   void main(void) {
       par {
         b2.main();
         b3stub.main();
       }
15   }
   };
```

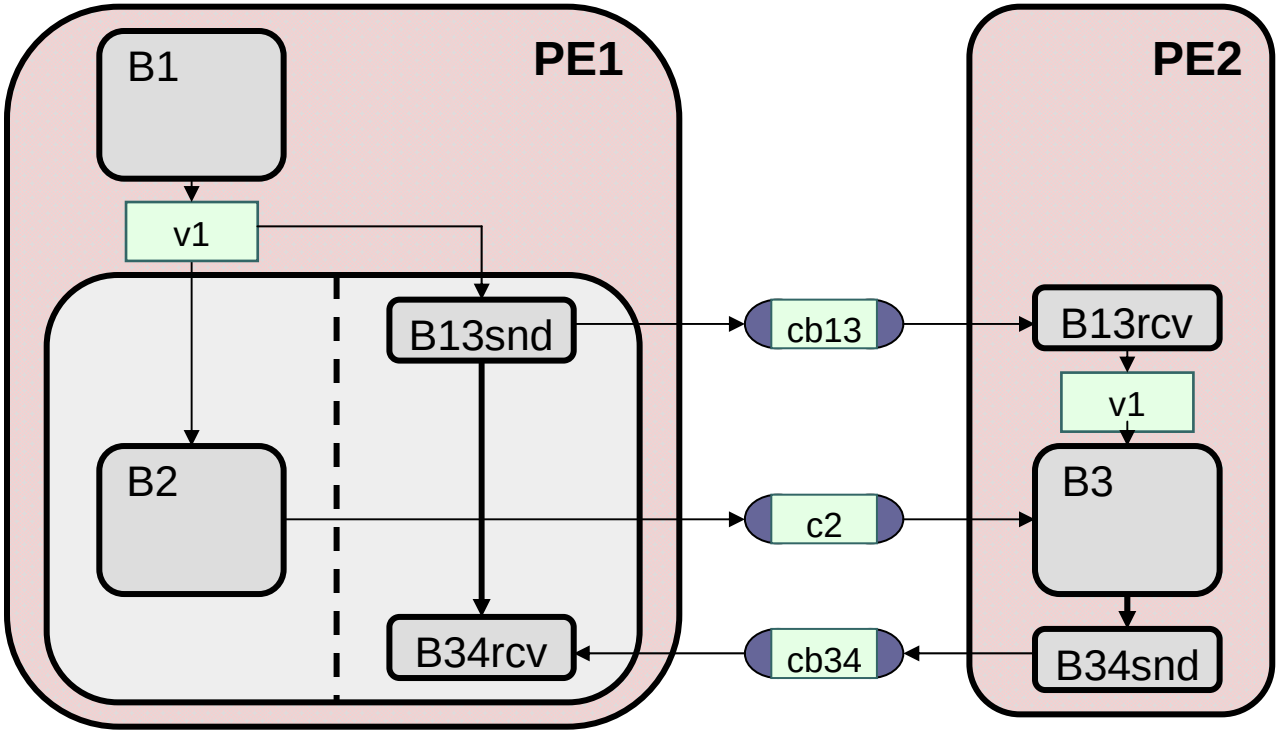
```
1  behavior B3stub( in int v1, ISend cb13,
                    IRecv cb34 ) {
     B13Snd b13snd( v1, cb13 );

5    BRcv   b34rcv( cb34 );

     void main(void) {
       b13snd.main();
       b34rcv.main();
10   }
   };
```

# Message-Passing Model (cont.)

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

     B13Snd b13snd(v1, cb13 );

     B3      b3     ( v1, c2 );
10   BSnd    b34snd( cb34 );

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



# Message-Passing Model (cont.)




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

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

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

# Timed Computation

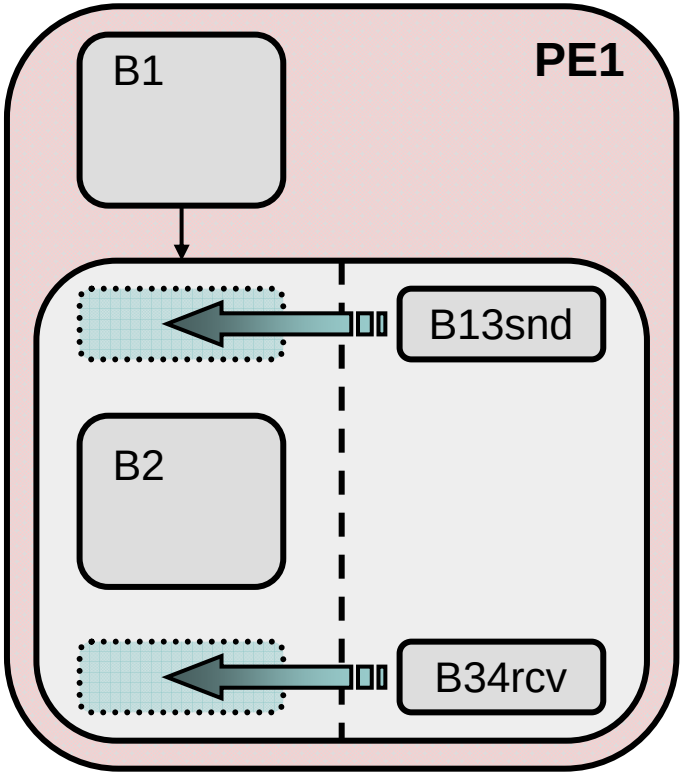
- Execution time of behaviors
  - Estimated target delay / timing budget
- Granularity
  - Behavior level / basic block level

- Annotate behaviors
  - Simulation feedback
  - Synthesis constraints

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

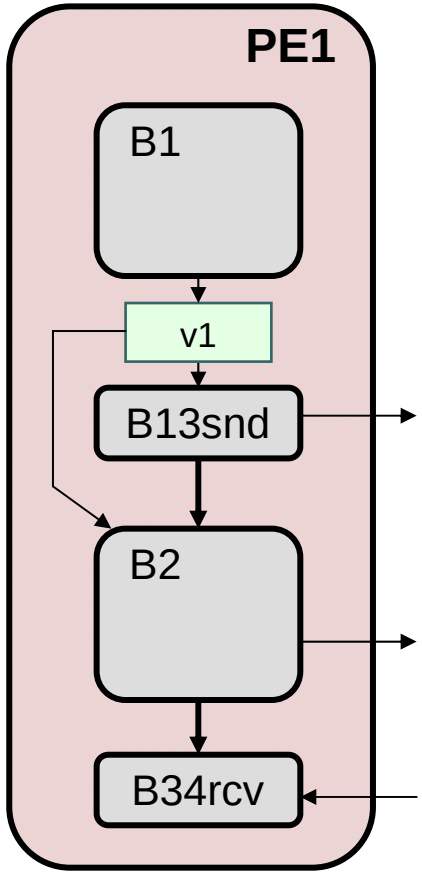
# Scheduling

➢ **Serialize behavior execution on components**




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

# Model after Scheduling



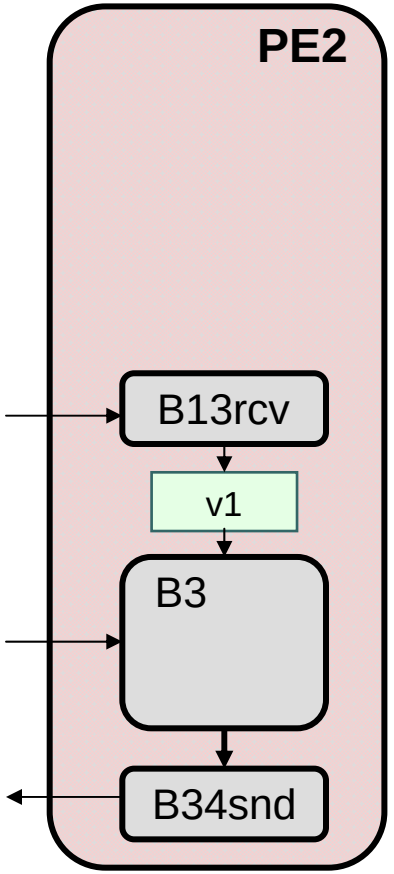

- **Statically scheduled PE1**

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

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

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

     }
20 };
```

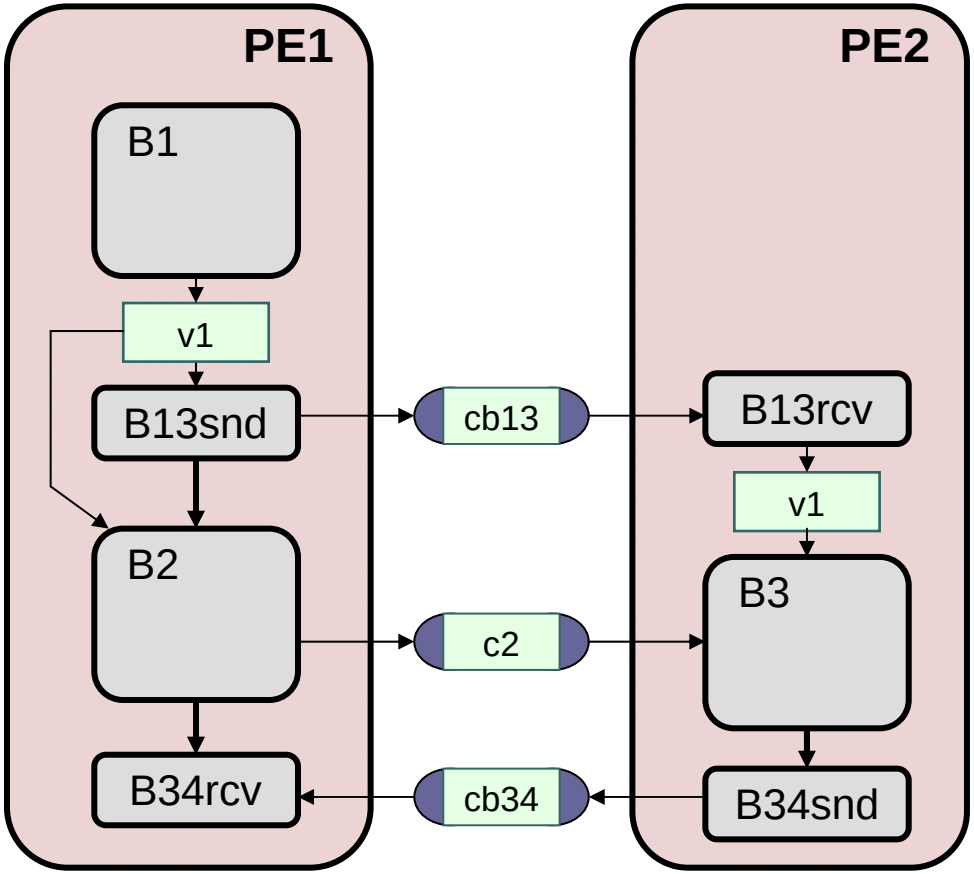
# Model after Scheduling (cont.)

- **No scheduling necessary for PE2**

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

     B13Rcv b13rcv( cb13, v1 );
     B3     b3    ( v1, c2 );
     BSnd   b34snd( cb34 );
10
     void main(void) {
       b13rcv.main();
       b3.main();
       b34snd.main();
15   }
   };
```



# Model after Scheduling (cont.)




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

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

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

# Architecture 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