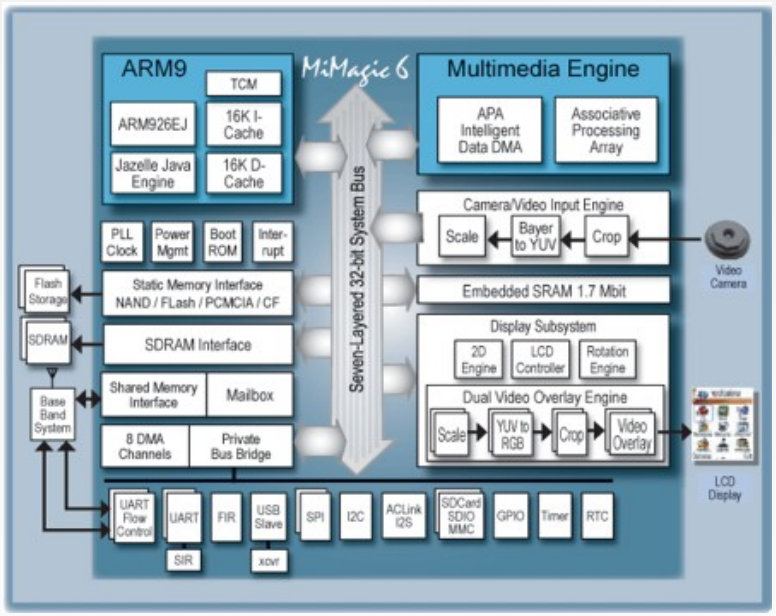
Formal and Executable Contracts for Transaction-Level Modeling in SystemC

Tayeb Bouhadiba & Florence Maraninchi & Giovanni Funchal

EMSOFT Conference
Grenoble, OCT 12-16 2009

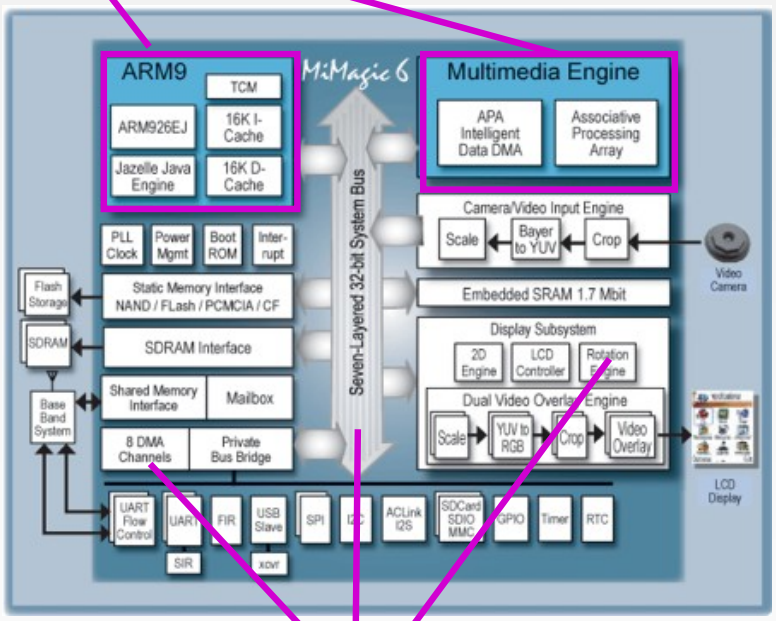
Systems-on-a-Chip



Hardware

Systems-on-a-Chip

Processors

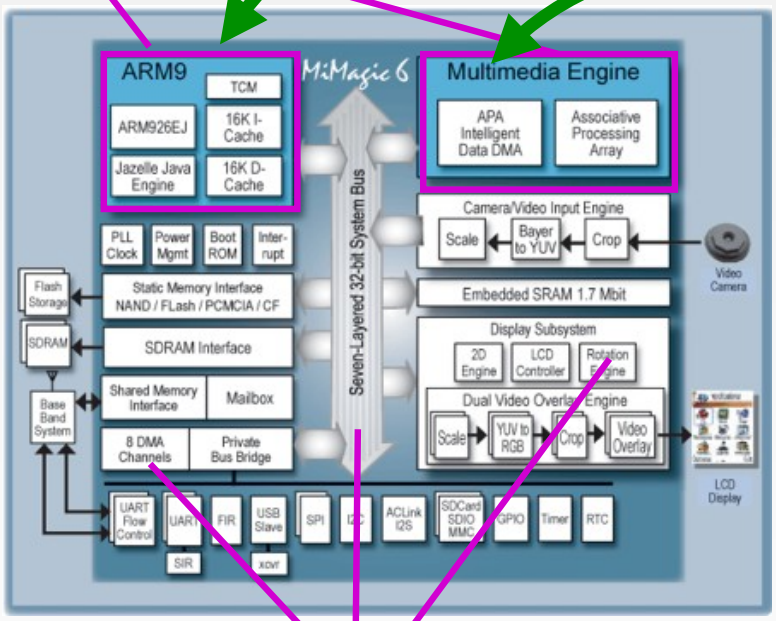


Hardware IP's (components)

Hardware

Systems-on-a-Chip

Processors



Software

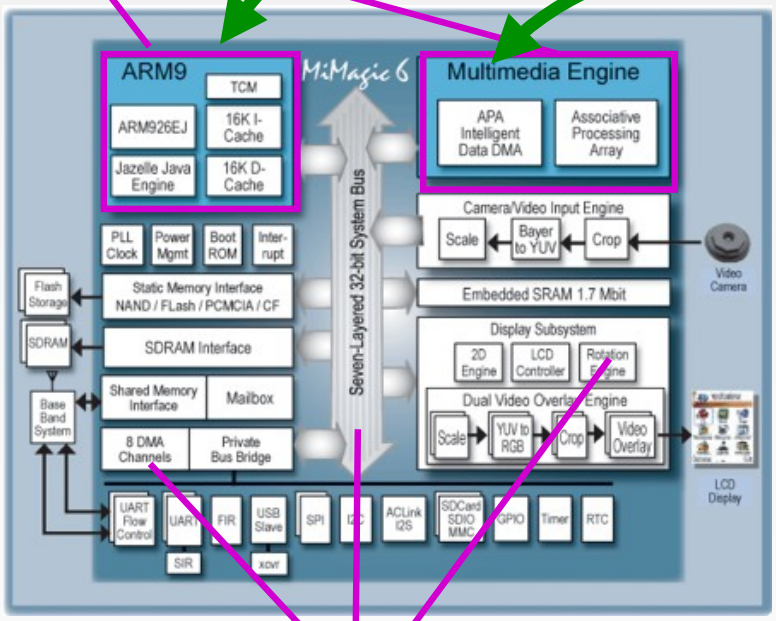
+

Hardware

Hardware IP's
(components)

Systems-on-a-Chip

Processors



Hardware IP's (components)

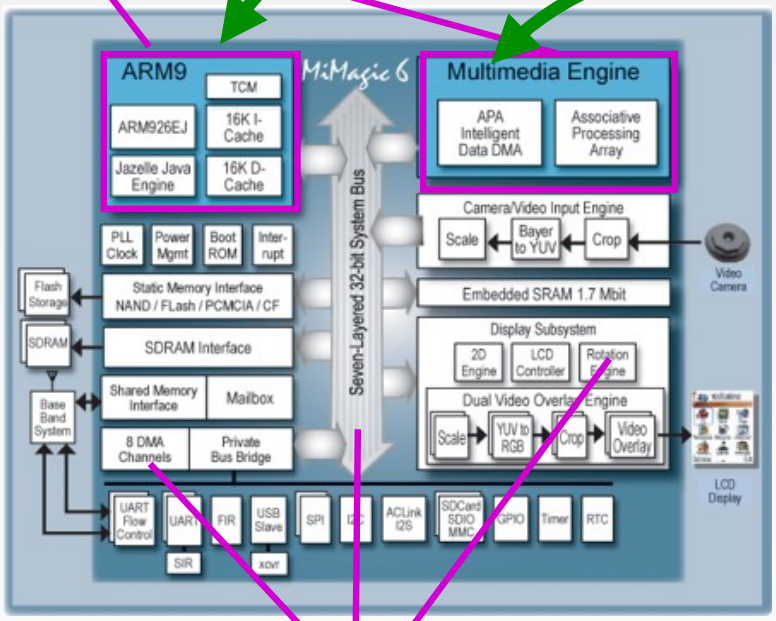
Software

+

Dedicated Hardware

Systems-on-a-Chip

Processors



Software

+

Dedicated Hardware

Hardware IP's (components)

⇒ Need for Virtual Prototypes of the Hardware

(Virtual Prototypes == Executable Models)

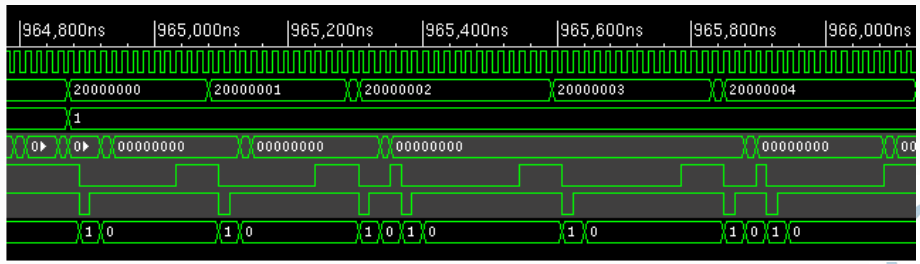
Virtual Prototyping for SoCs

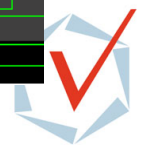
Register Transfer Level (RTL)

Synthesizable+

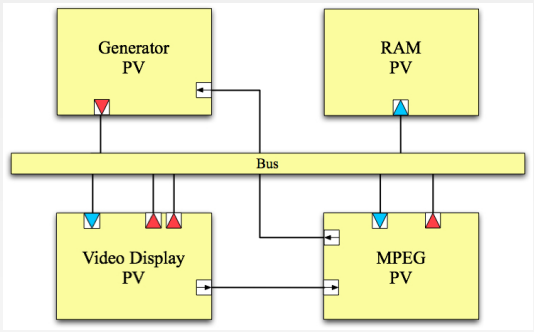
Cycle and Data Accurate+

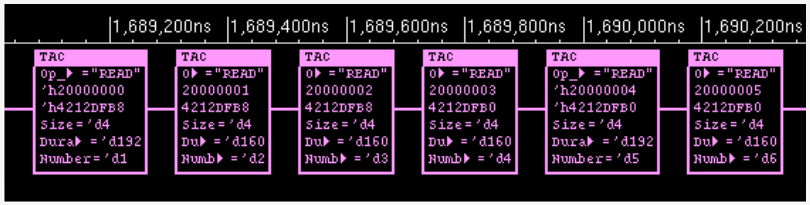
Slow Simulations–





Virtual Prototyping for SoCs





Transaction Level Modeling (TLM)

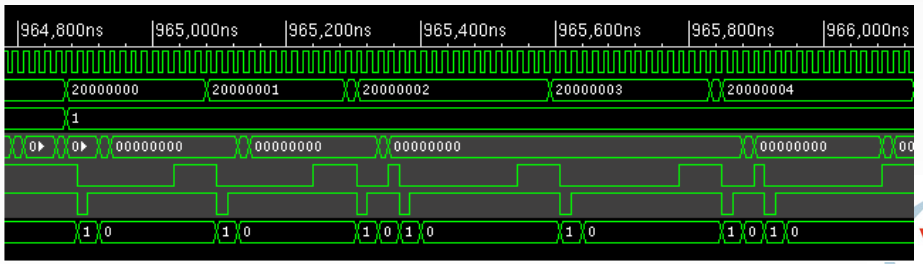
+ Early Available
+ Fast Simulations
– Not synthesizable

Register Transfer Level (RTL)

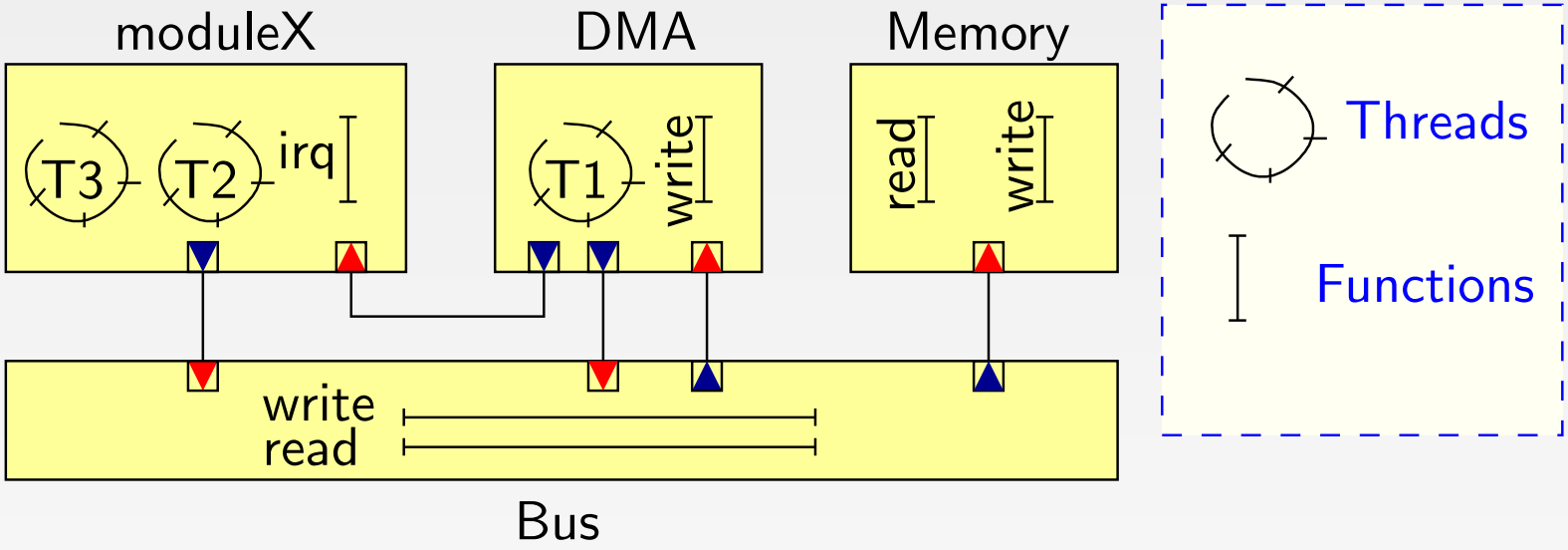
Synthesizable+

Cycle and Data Accurate+

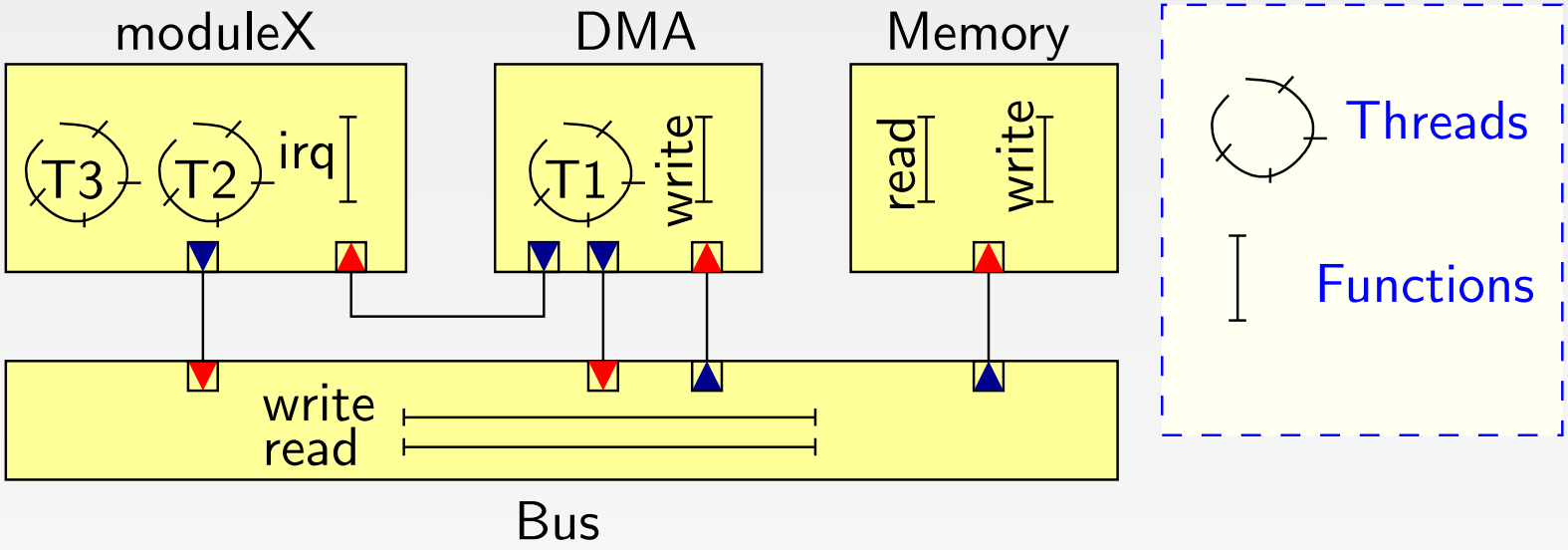
Slow Simulations–



TLM in practice (SystemC)

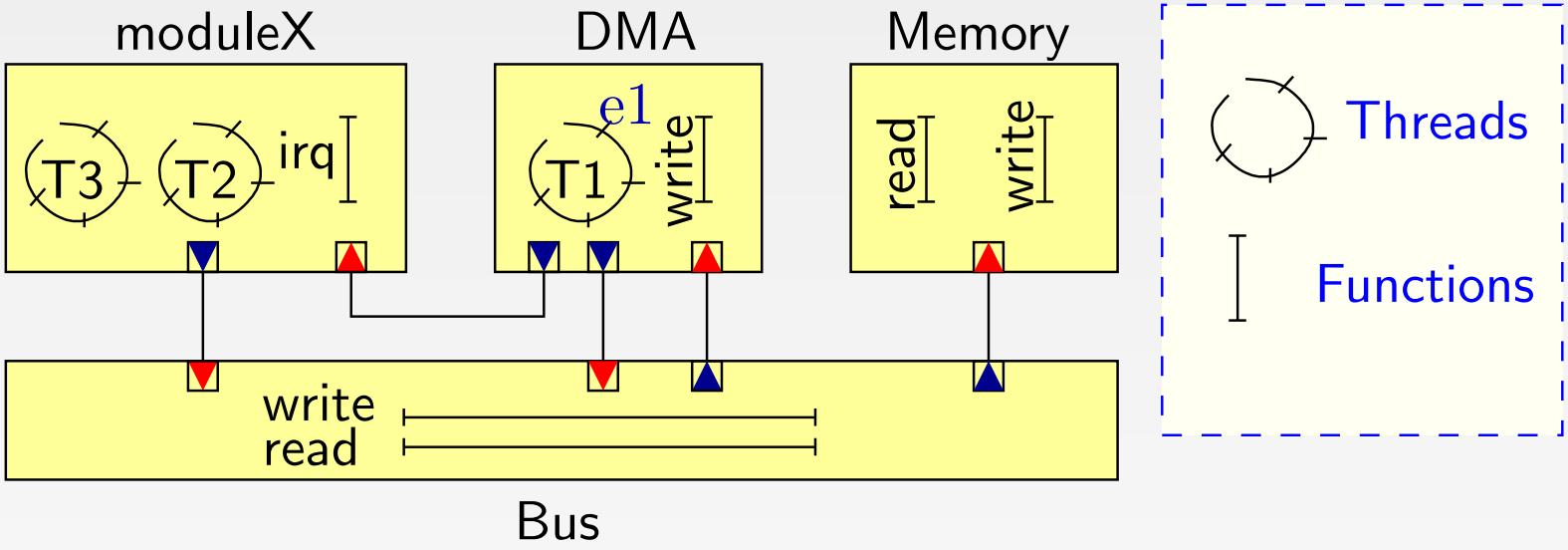


TLM in practice (SystemC)



Some Guidelines:

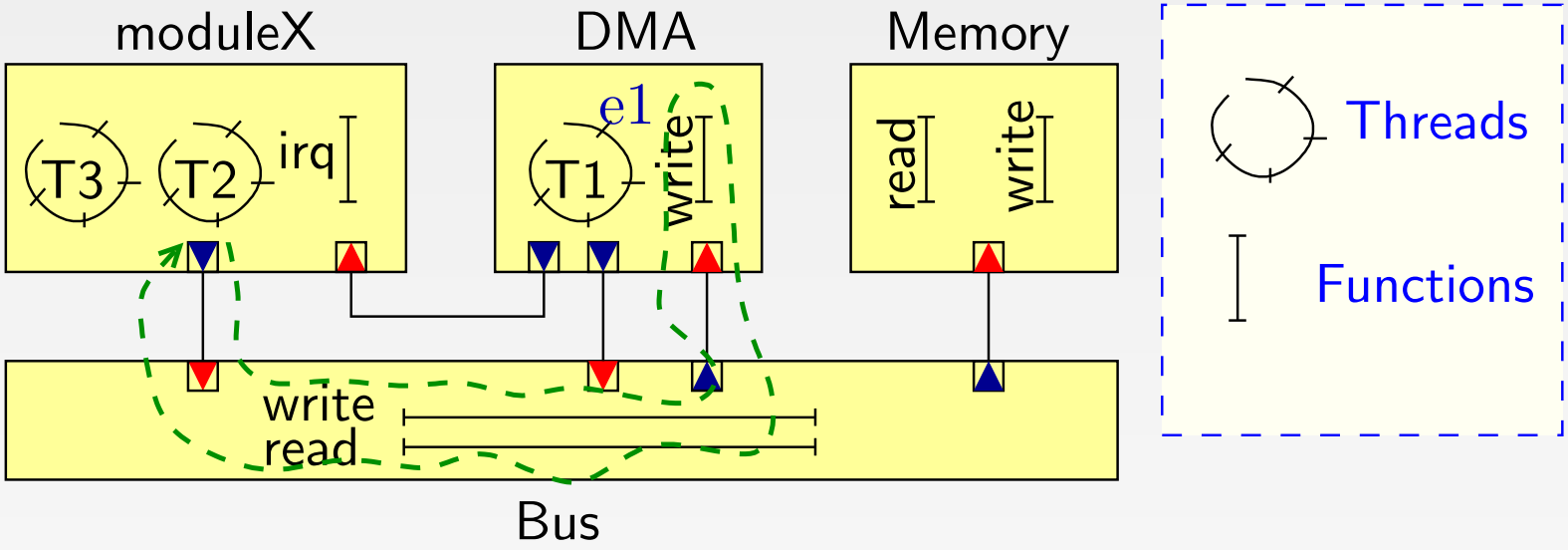
TLM in practice (SystemC)



Some Guidelines:

- Inside a module:
  - Shared variables
  - Events (wait, notify)

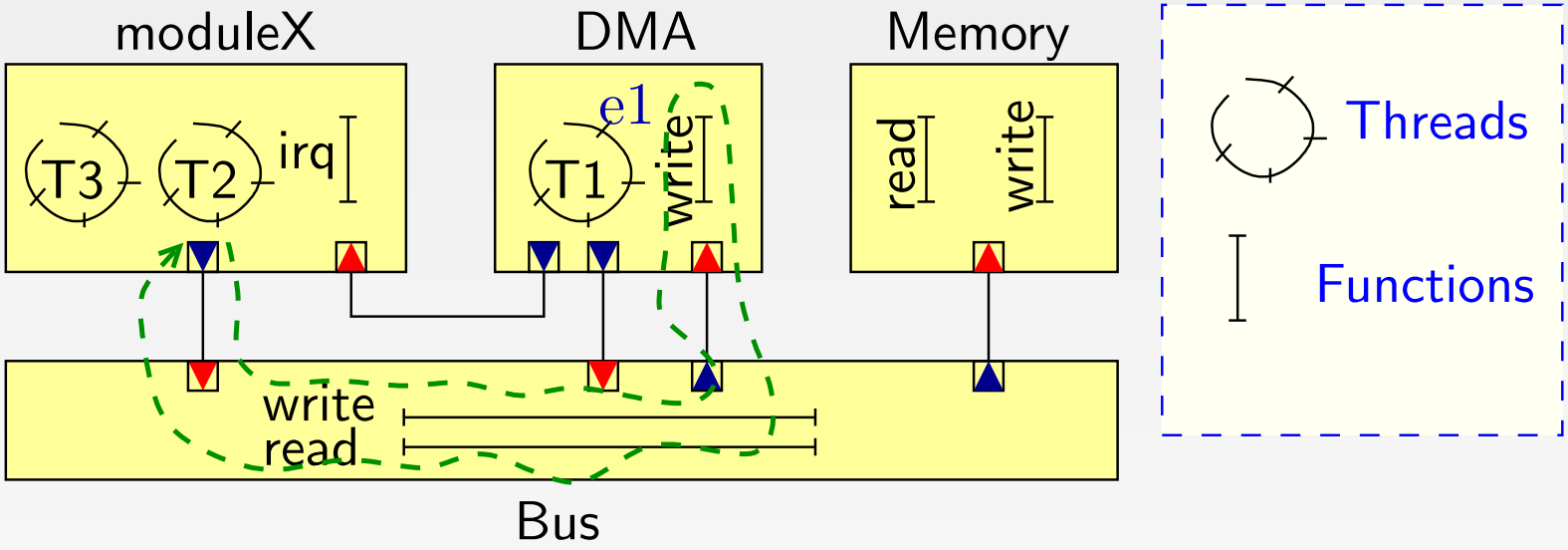
TLM in practice (SystemC)



Some Guidelines:

- Inside a module:
  - Shared variables
  - Events (wait, notify)
- Between modules:
  - Function calls through ports
  - ...

TLM in practice (SystemC)

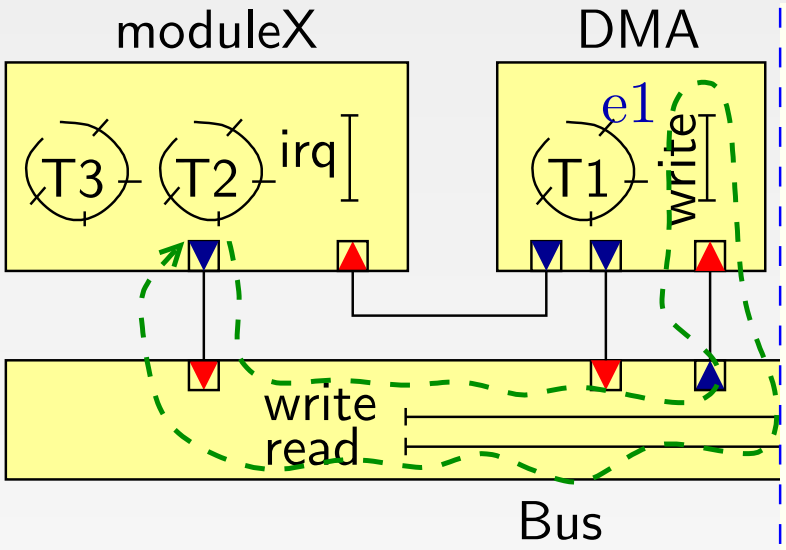


Some Guidelines:

- Inside a module:
  - Shared variables
  - Events (wait, notify)
- Between modules:
  - Function calls through ports
  - ...

A non-preemptive scheduler manages the execution of threads

TLM in practice (SystemC)




```
moduleX::T2(){
  while(true){
    x = 42;
    while (x > 0){
      y = p.read(addrM + x)^0xFF;
      p.write(addrM , x);
    }
    p.write(addrD+0x0, addrM);
    p.write(addrD+0x1, 42);
    p.write(addrD+0x4, 0x01);
    wait(e0);
    ...
    e2.notify();
  }
  ...
}

moduleX::irq(int n){
  ...
  e0.notify();
  ...
  wait(e2);
}
```

Example Implementation of *moduleX*

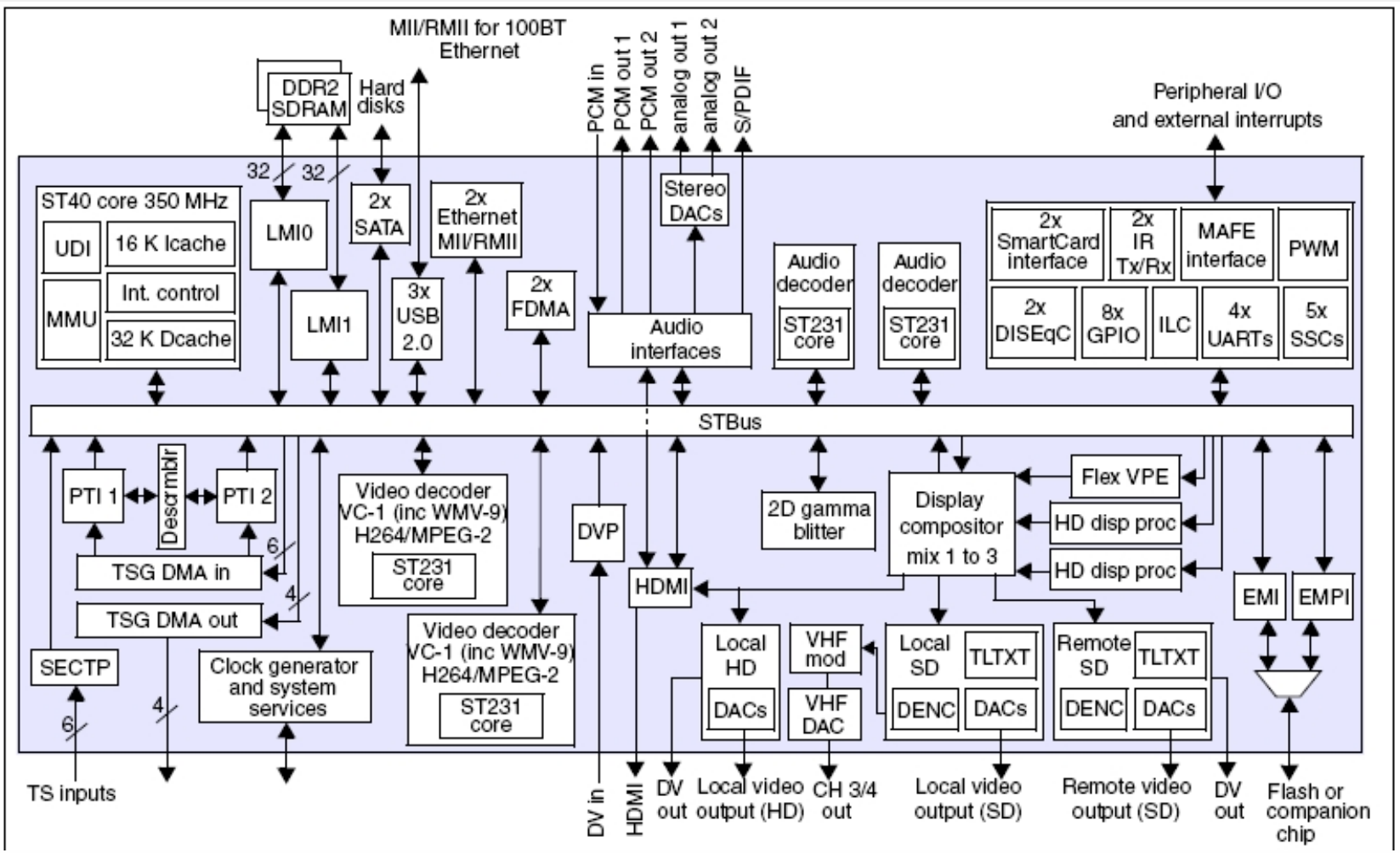
Some Guidelines:

- Inside a module:
  - Shared variables
  - Events (wait, notify)
- Between modules:
  - Function calls through ports
  - ...

A non
ages th

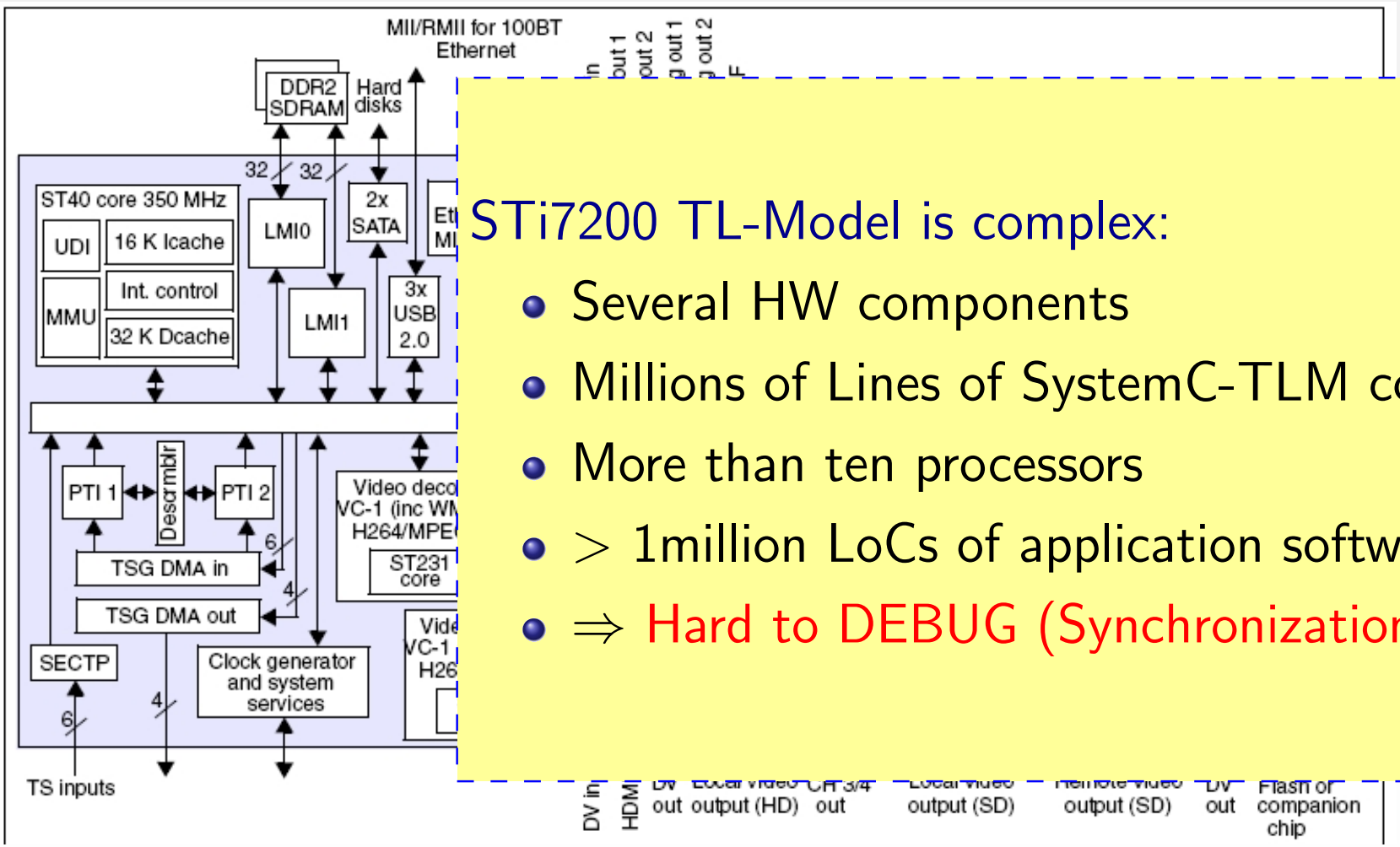
Example System-on-Chip

Multimedia SoC STi7200 (Set-top Boxes, HD television,...) (STMicroelectronics)



Example System-on-Chip

Multimedia SoC STi7200 (Set-top Boxes, HD television,...) (STMicroelectronics)



STi7200 TL-Model is complex:

- Several HW components
- Millions of Lines of SystemC-TLM code
- More than ten processors
- $>$ 1million LoCs of application software
- $\Rightarrow$ Hard to DEBUG (Synchronizations)!!

TLM + a Formal component model (42)

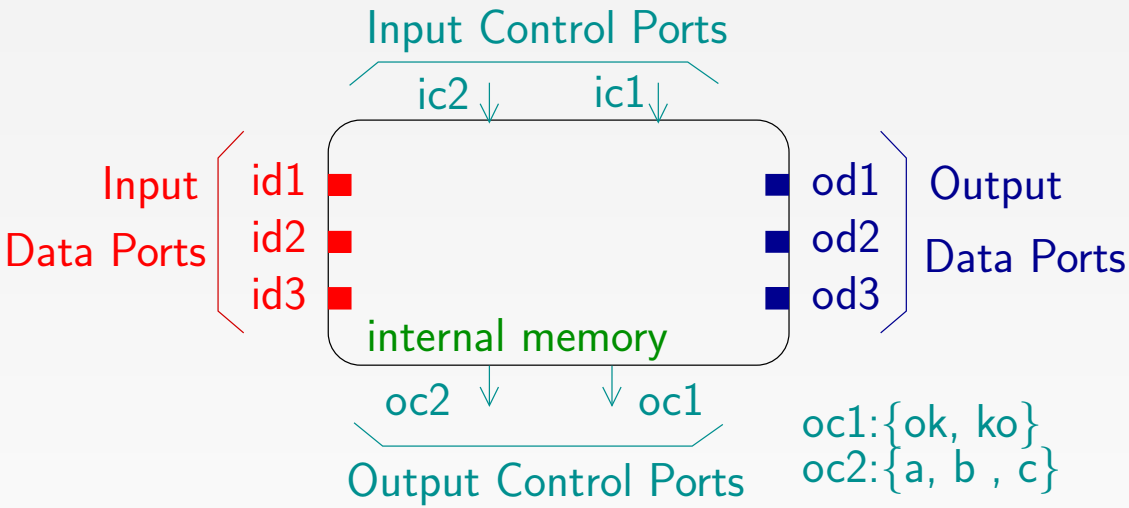
- Motivation
  - Simplify the simulation
  - Focus on Control Flow and Synchronizations
  - Find bugs earlier
- How?
  - Joint use of SystemC-TLM and 42
  - Using 42 Control Contracts

TLM + a Formal component model (42)

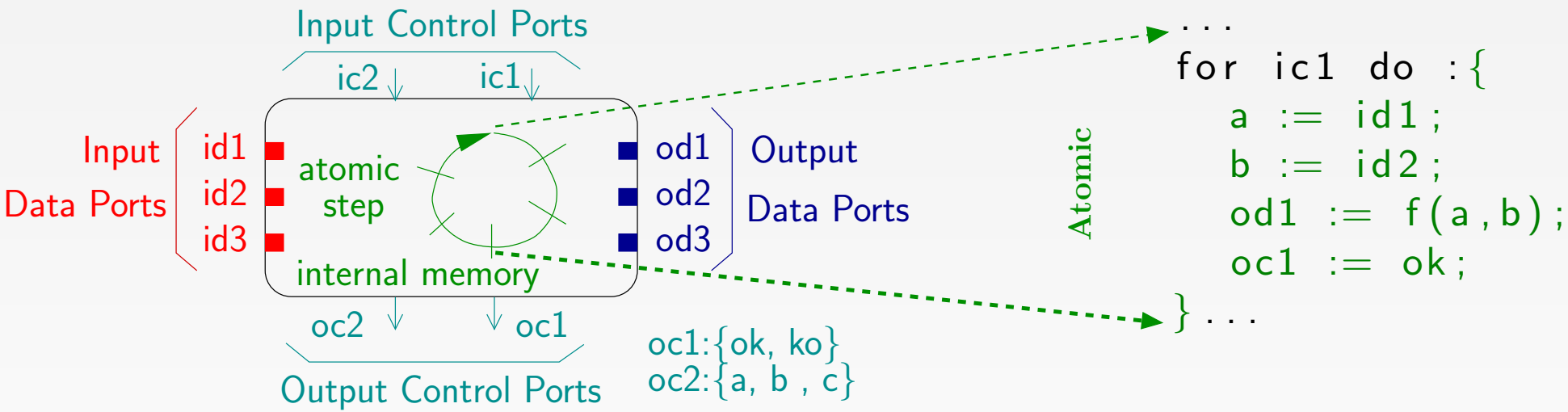
- TLM = Transaction-Level Modeling
  - + Widely used
  - + Powerful (large systems, heterogeneous models, ...)
  - + Intrinsically component-based
  - − Very informal, models are mixed with the simulation mechanics
- 42 = a Formal component-based model for embedded systems
  - − Not flexible enough for large systems
  - + Formal, clear distinction between the semantics of the model, and the simulation mechanisms
  - + Able to represent various MoCCs (*à la* Ptolemy)
  - + Formal interface and contract definitions for components
  - + Connection to various verification tools
  - + An execution mode for contracts only

Content

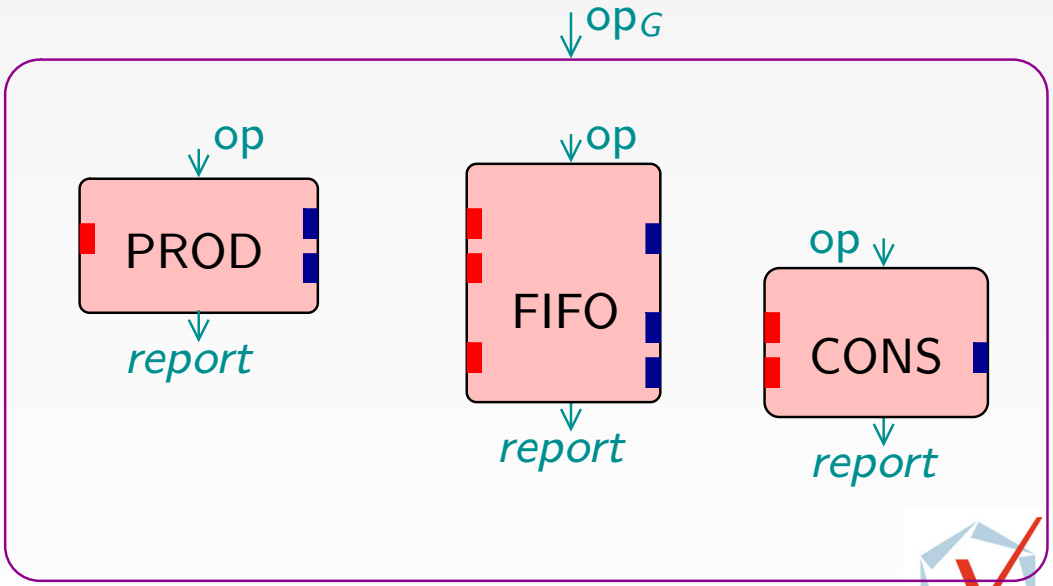
42 in a Nutshell: Basic Components



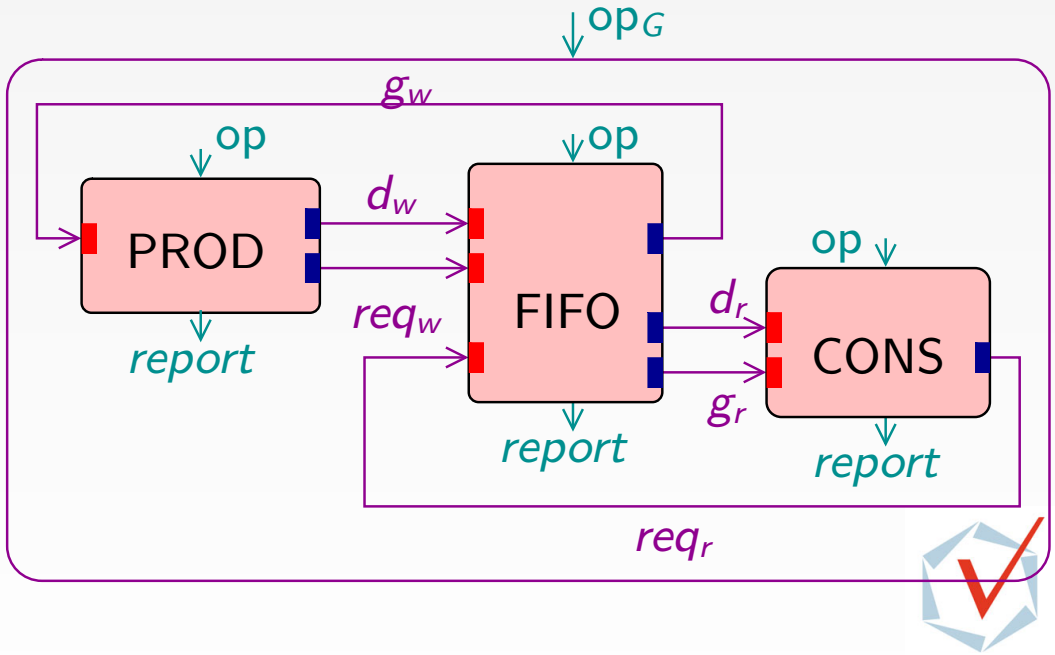
42 in a Nutshell: Basic Components



42 in a Nutshell: Assembling Components

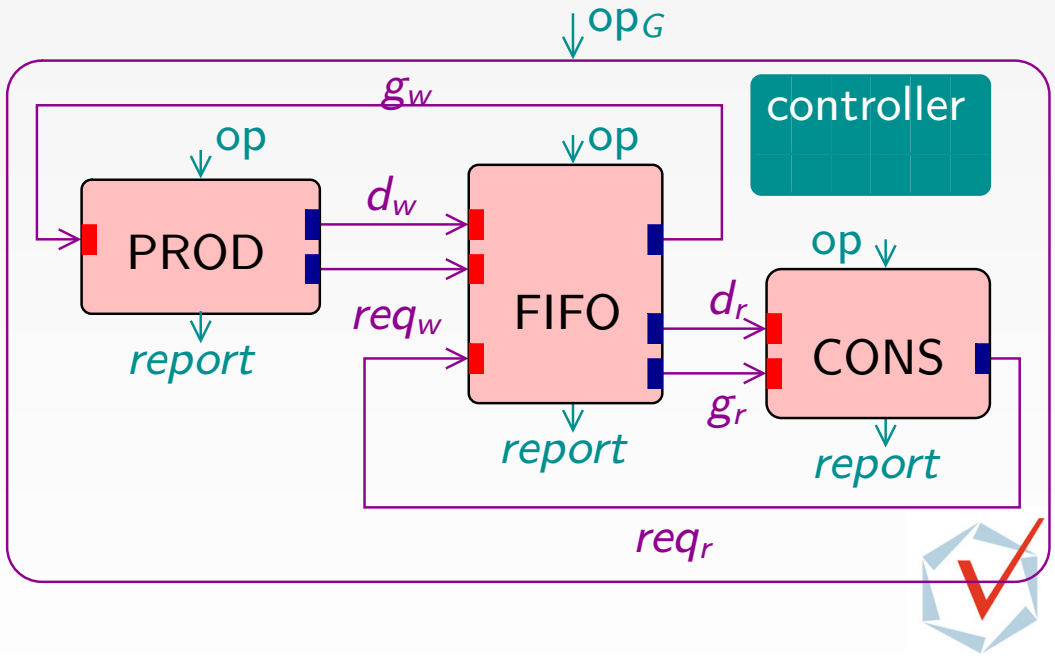


42 in a Nutshell: Assembling Components



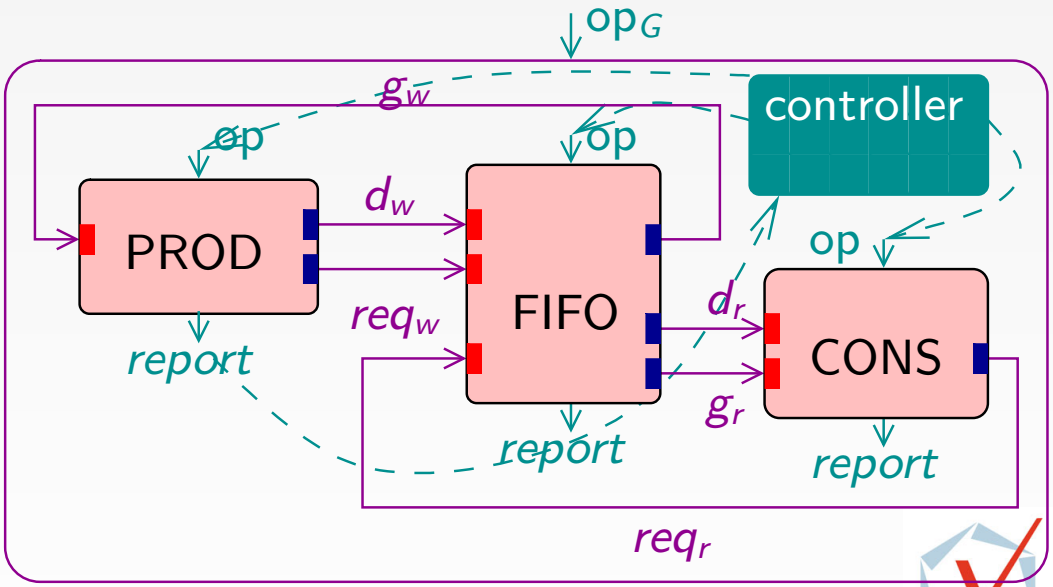
42 in a Nutshell: Assembling Components

For each OP_G the controller:



42 in a Nutshell: Assembling Components

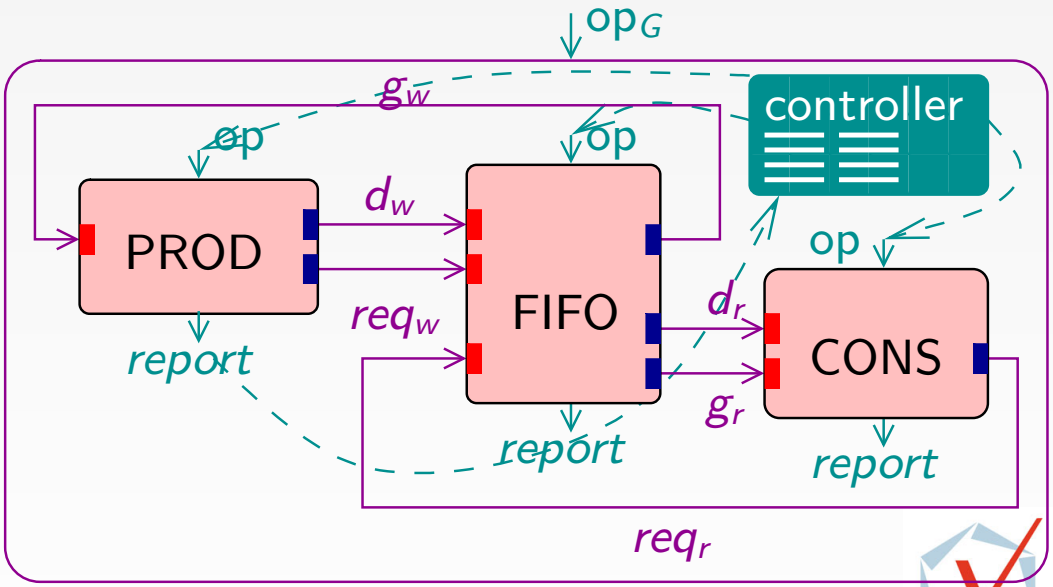
For each OP_G the controller:
- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)



42 in a Nutshell: Assembling Components

For each OP_G the controller:

- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr, reqw...)

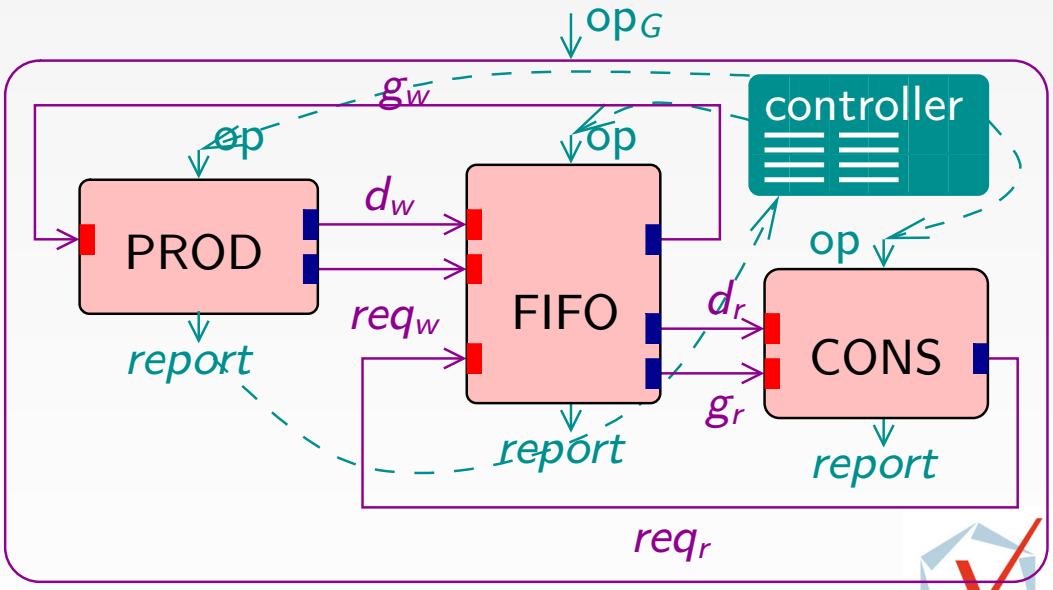


42 in a Nutshell: Assembling Components

```
Controller is {
 var M : bool;
 for OP_G do :{ /*defines opg.
  dw, dr, reqw,...: fifo(1,int);
  M := random();
  if (M) {
   PROD.op ; reqw.put; reqw.get;
   FIFO.op ; gw.put; gw.get;
   a := FIFO.report; /*reads oc.
   if(a==ok){ /*output control
     PROD.op;dw.put; dw.get;
     FIFO.op;/*activates FIFO.
       ...
   }
   ...
  }else{
   CONS.op; reqr.put; reqr.get;
   FIFO.op ; gr.put; gr.get;
   a := FIFO.report;
   ...
  }
 }
}
```

For each OP_G the controller:

- Activates PROD, CONS, FIFO through op
- Reads their output control ports (report)
- Manages a temporary memory (reqr, dr, reqw...)

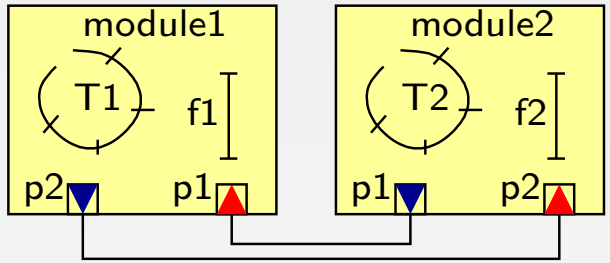


Contents

Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

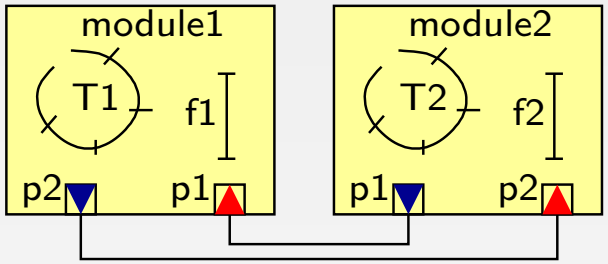
```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

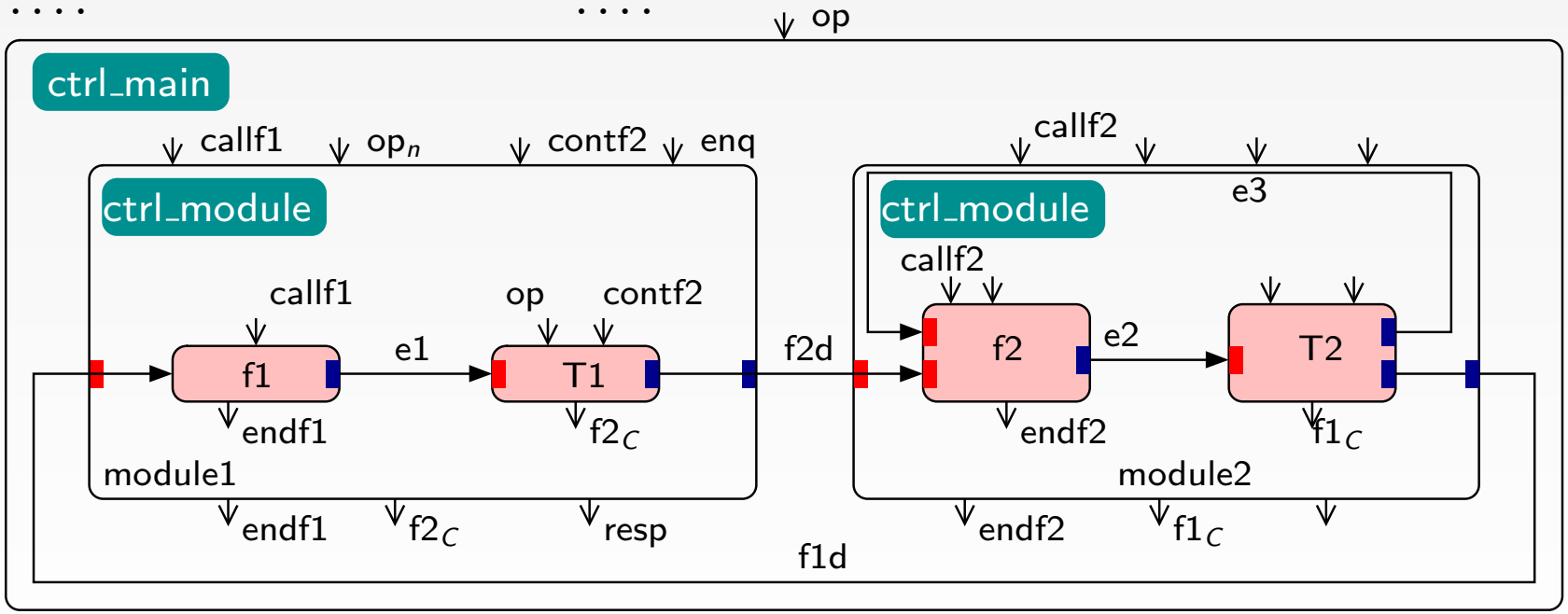


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

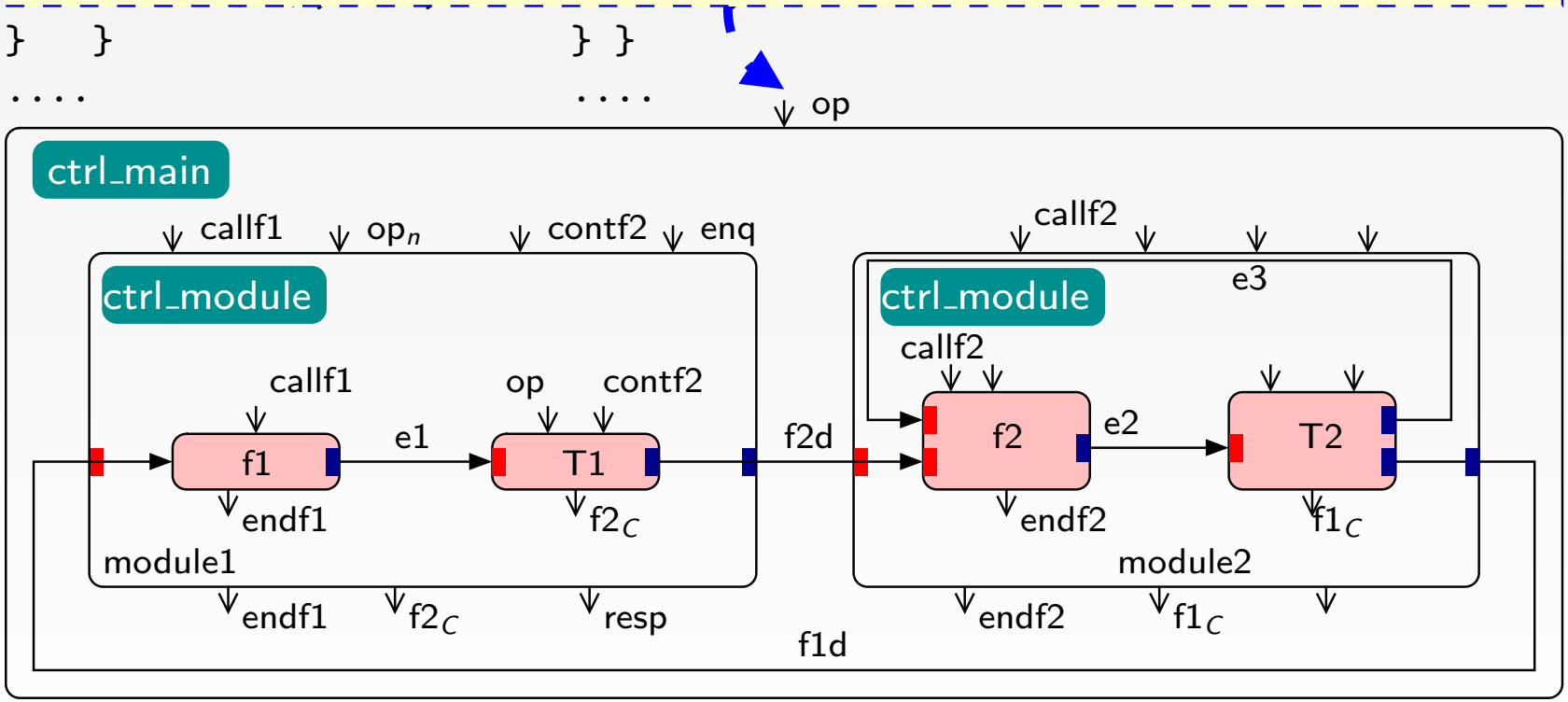
```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```





Structural Correspondence

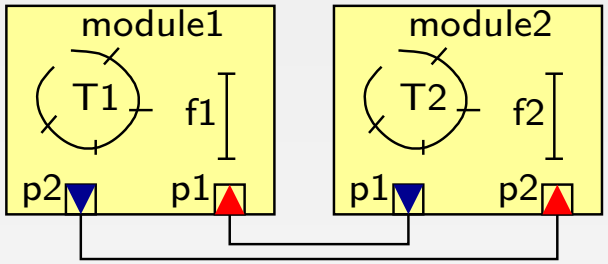
An activation of the main component with OP corresponds to what happens in the SystemC-TLM model when the scheduler elects a new thread to execute.

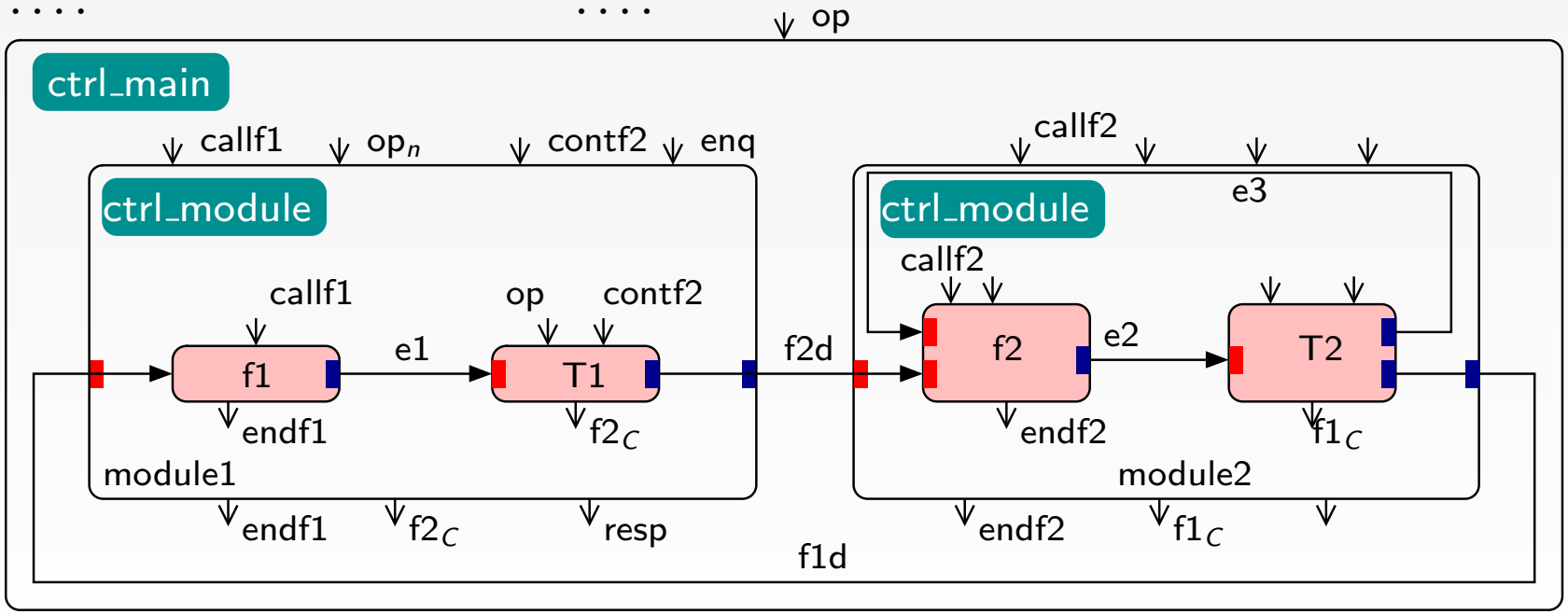


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

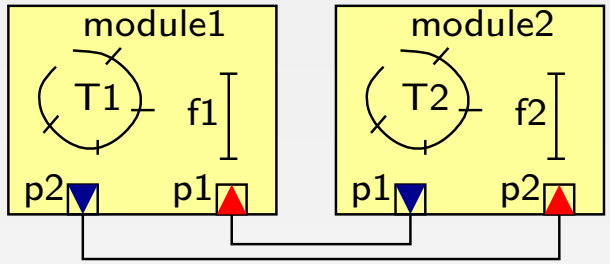


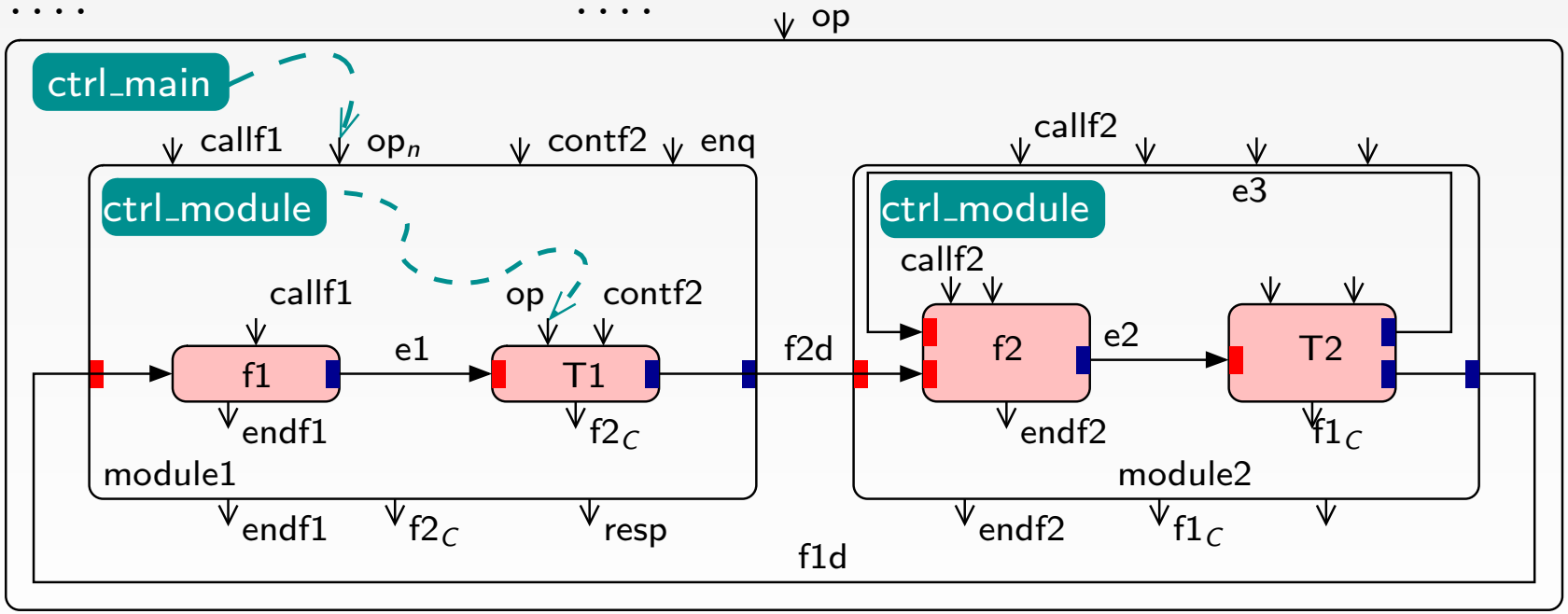


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

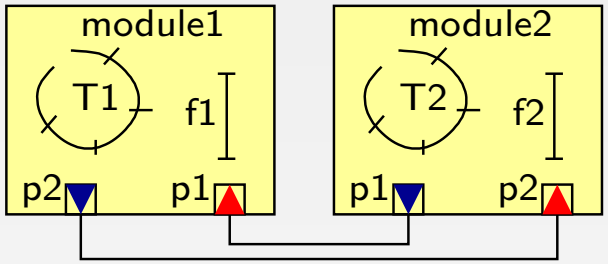


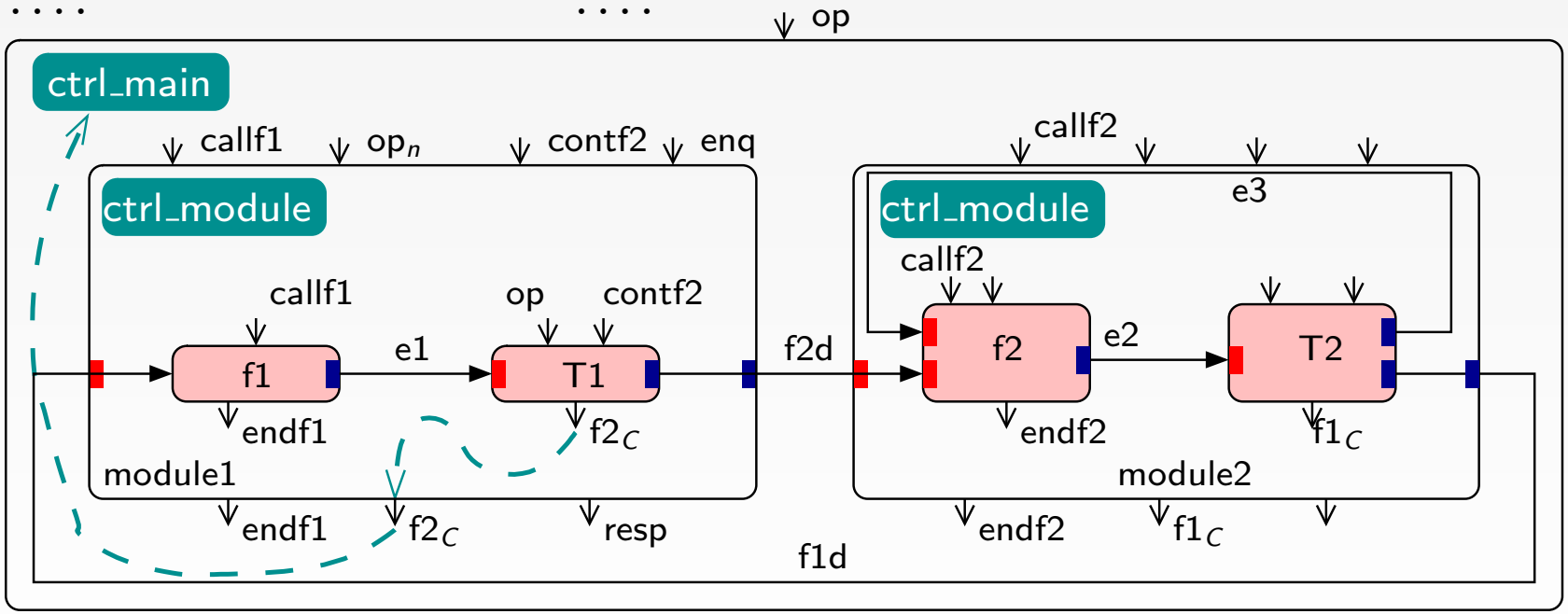


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

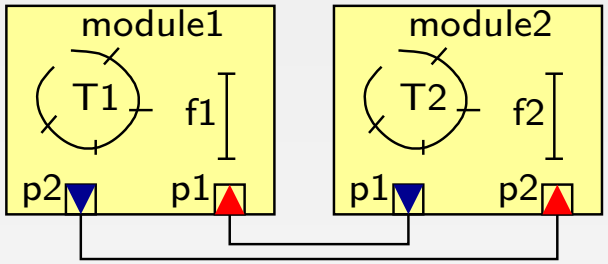


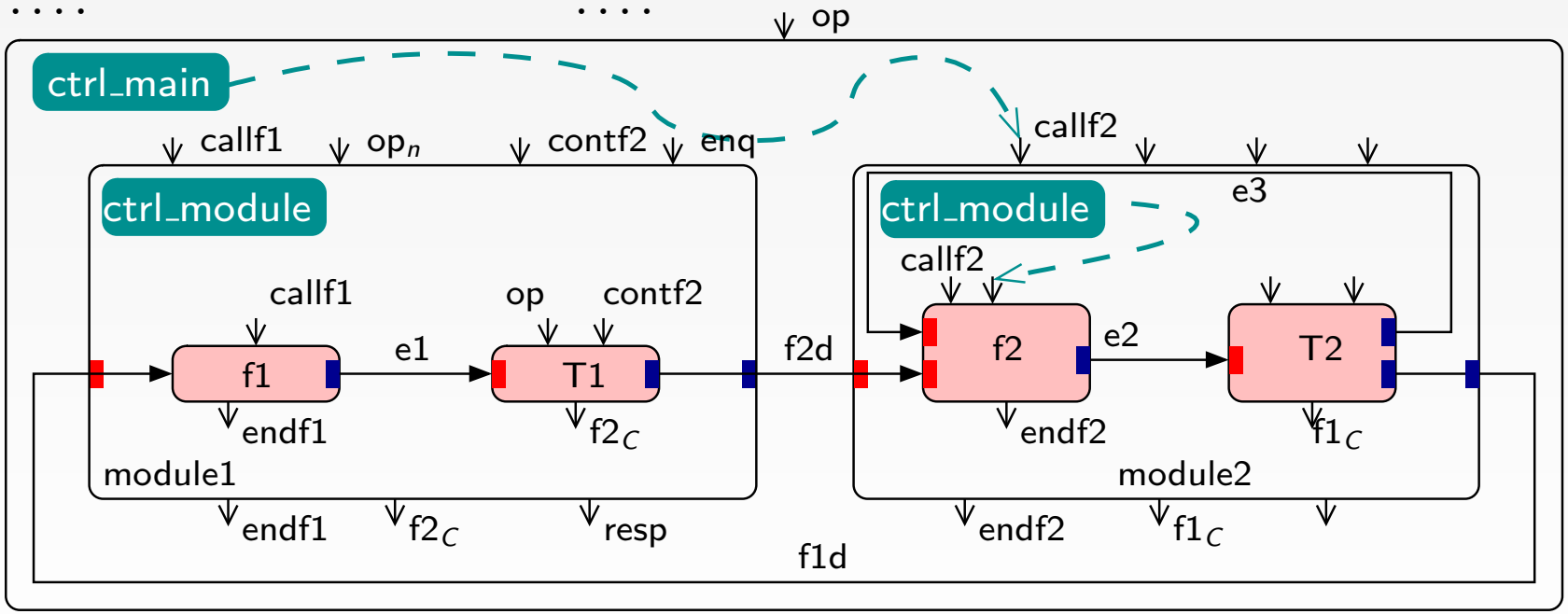


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

```
module2::f2(int x){
      ...;
      y = x * 42;
      ...;
      notify(e2);
      ...;
      ...;
} }
....
```

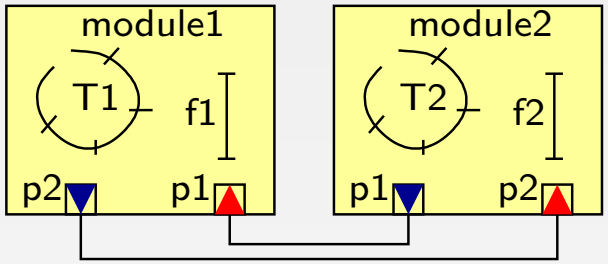


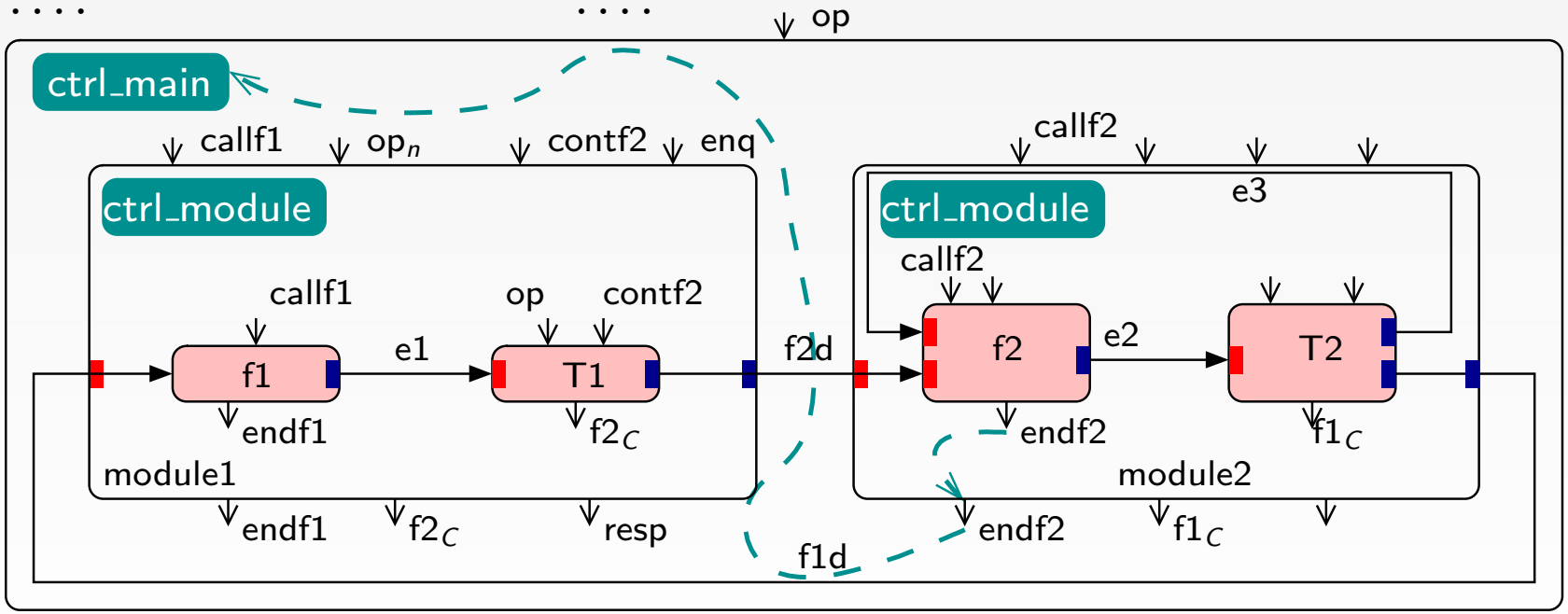


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

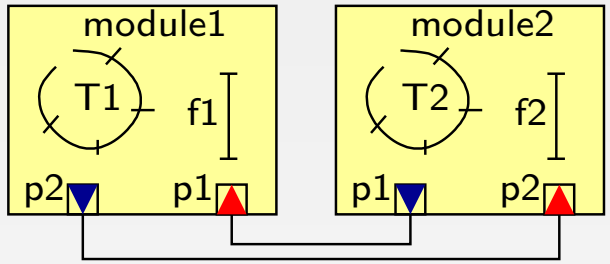


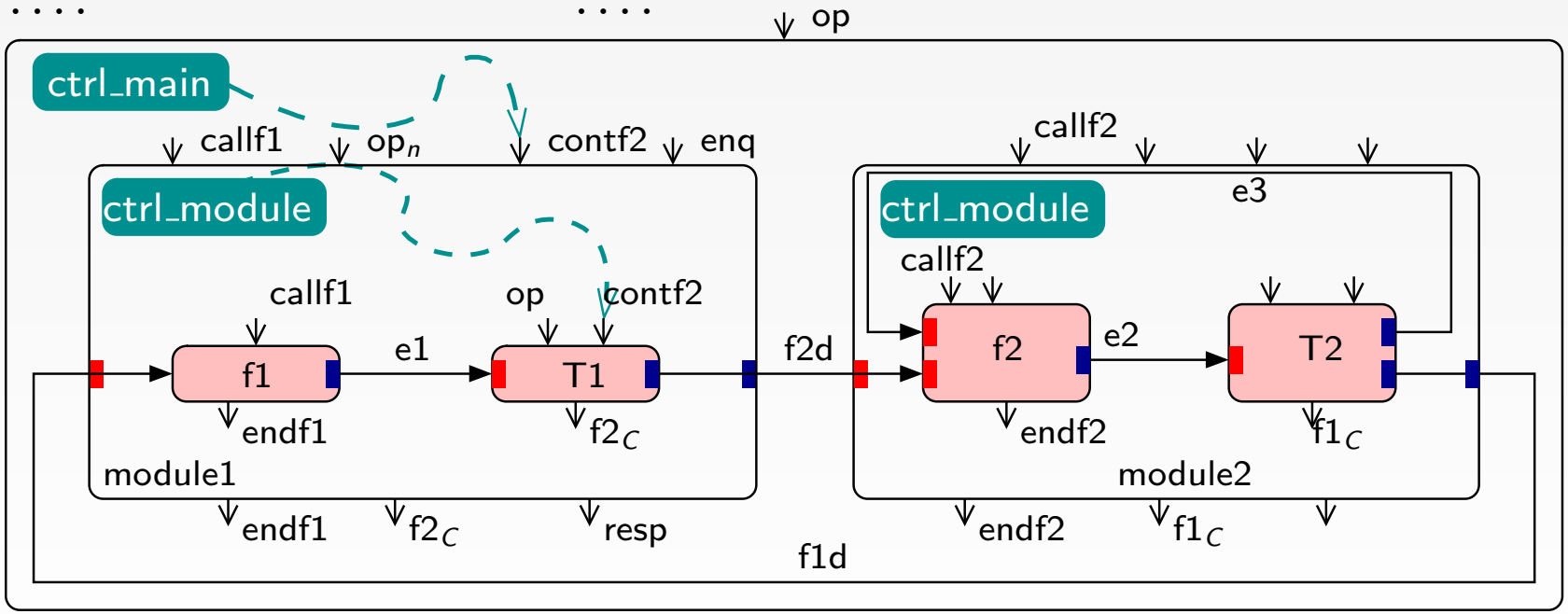


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

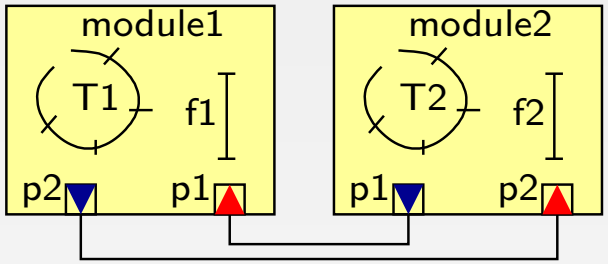


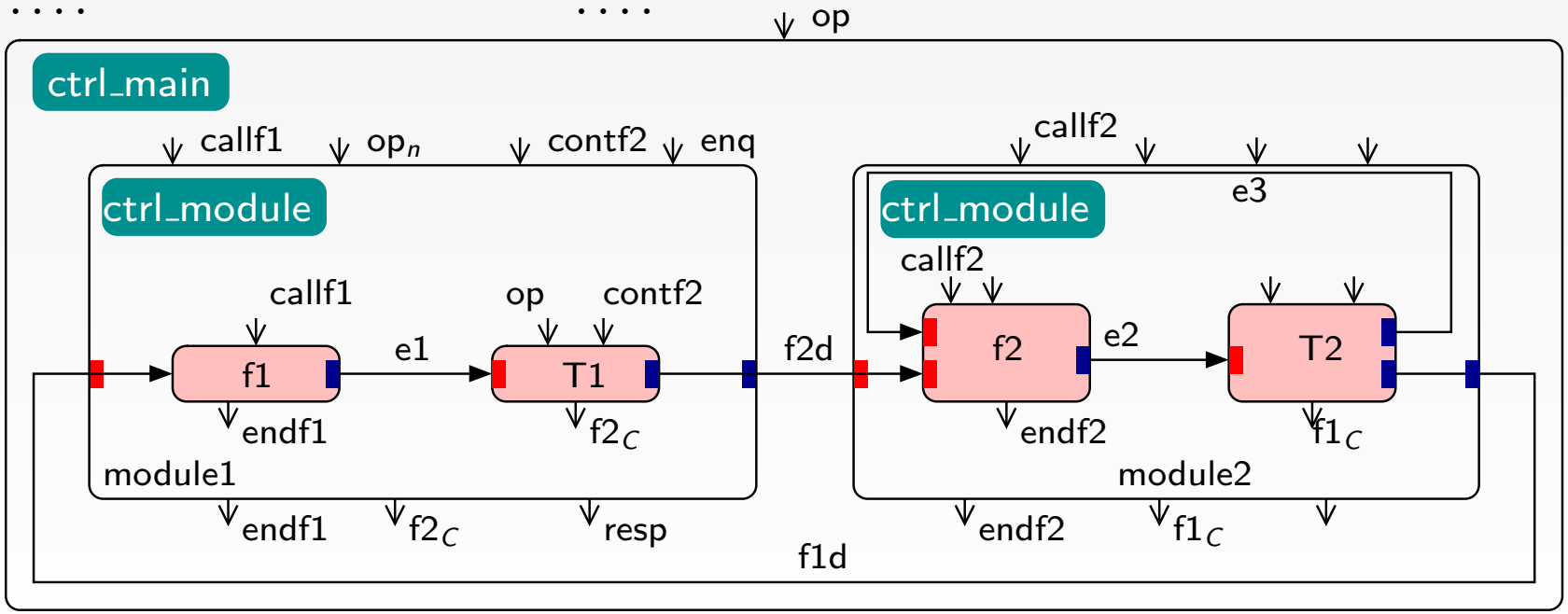


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```

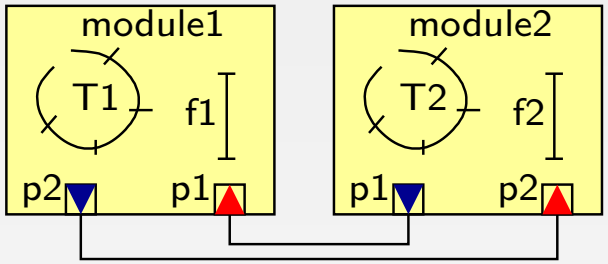


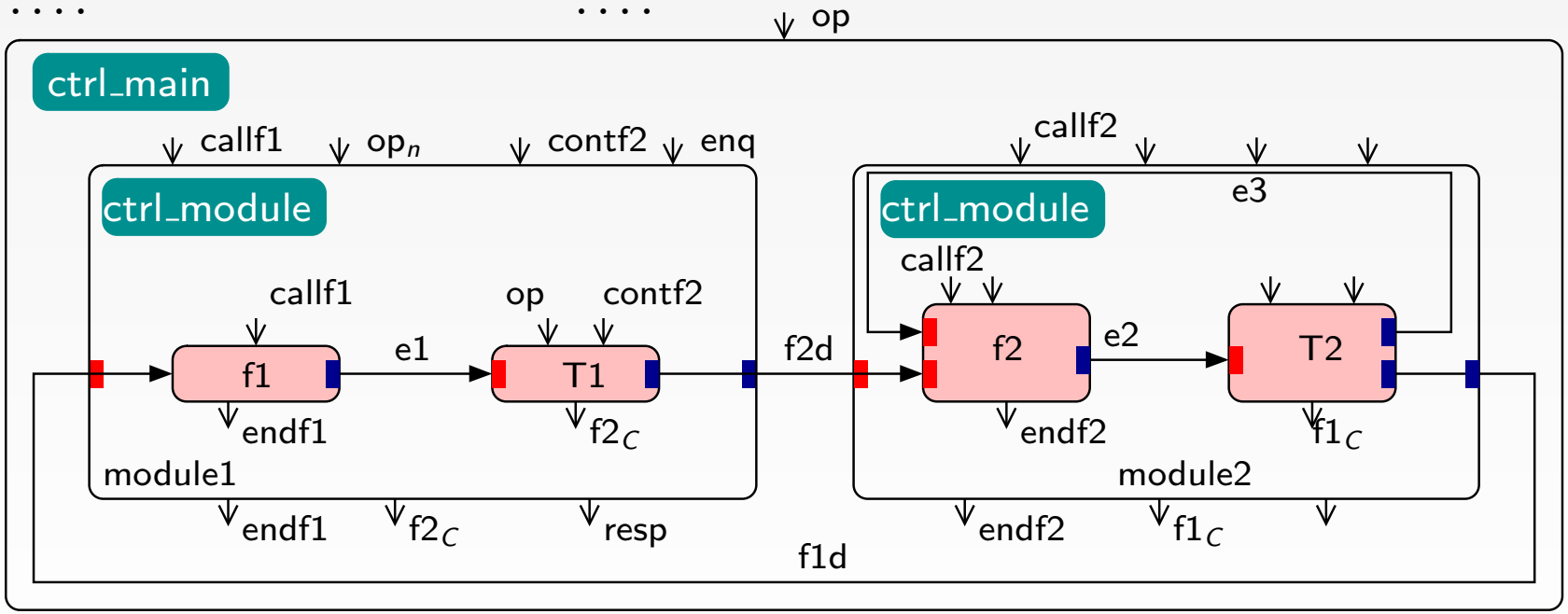


Structural Correspondence

```
module1::T1(){
    while(true){
      x++;
      ...;
      p2.f2(x);
      ...;
      wait(e1); ...;
}   }
....
```

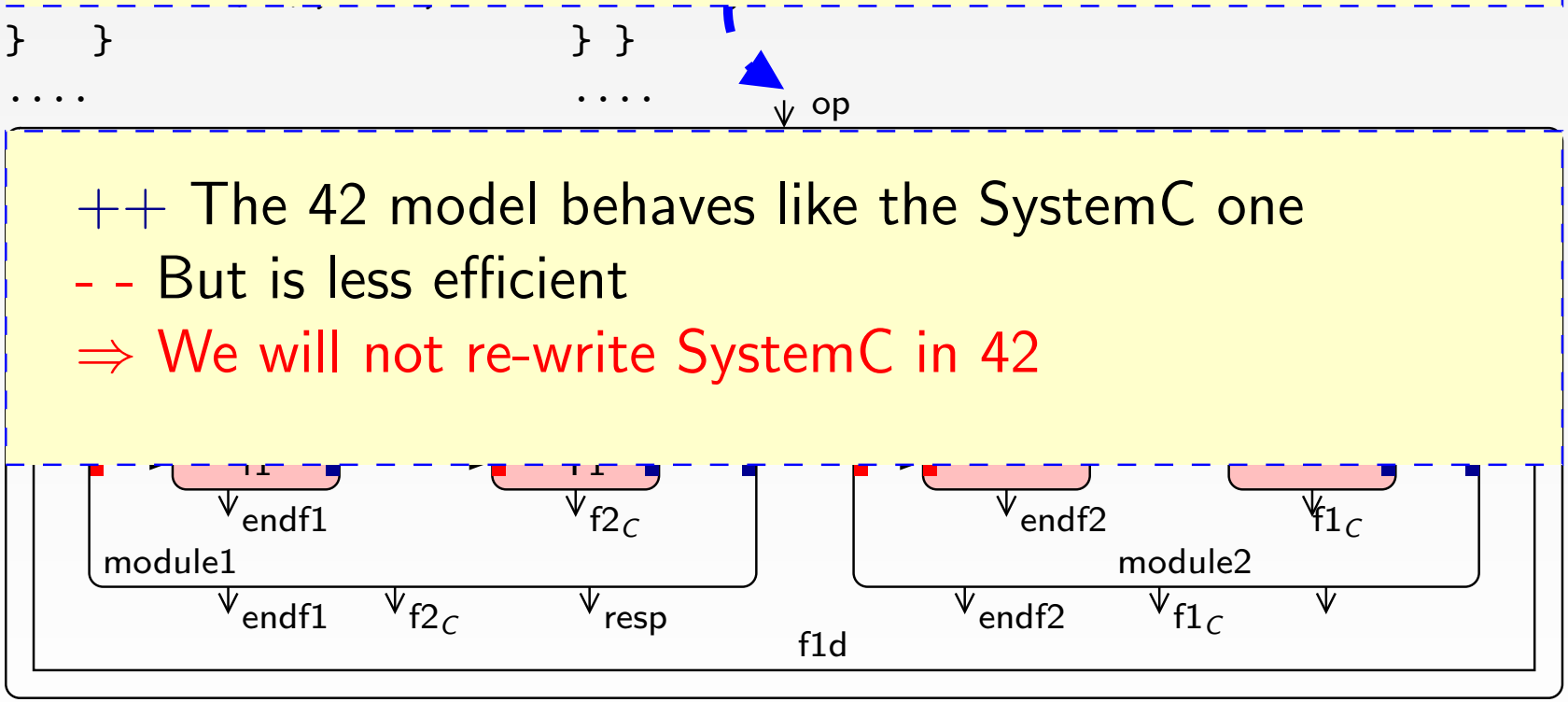
```
module2::f2(int x){
    ...;
    y = x * 42;
    ...;
    notify(e2);
    ...;
    ...;
} }
....
```





Structural Correspondence

An activation of the main component with OP corresponds to what happens in the SystemC-TLM model when the scheduler elects a new thread to execute.



++ The 42 model behaves like the SystemC one

- - But is less efficient

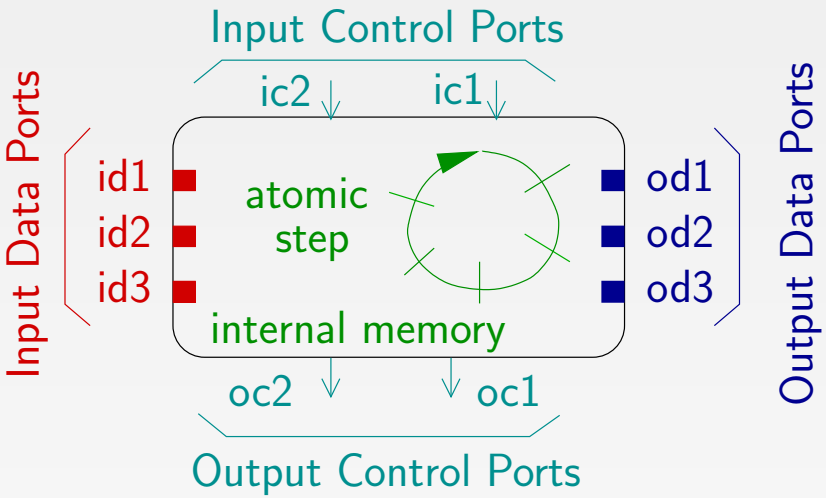
⇒ We will not re-write SystemC in 42

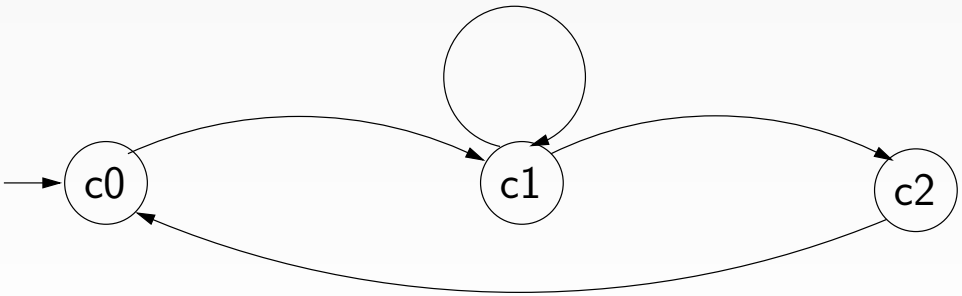
Interesting Uses of the Correspondence

- Motivation
  - Simplify the simulation
  - Focus on Control Flow and Synchronizations
  - Find bugs earlier
- How?
  - Joint use of SystemC-TLM and 42
  - Using *42 Control Contracts*

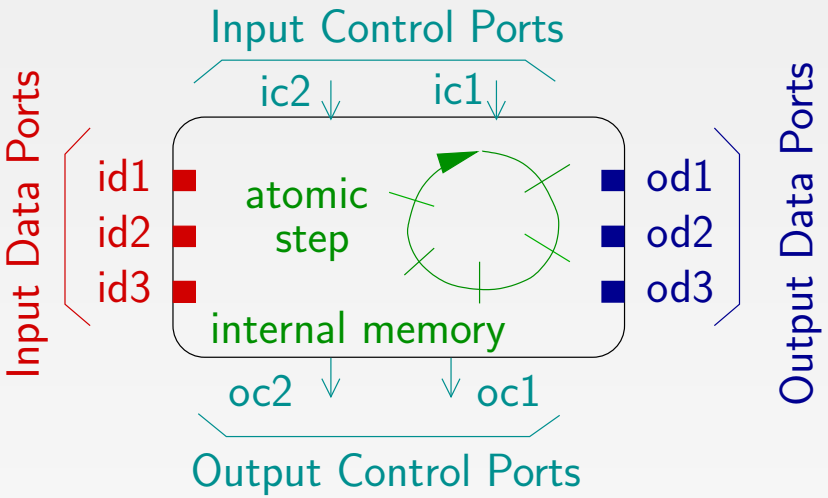
Contents

Control Contracts for 42 Components

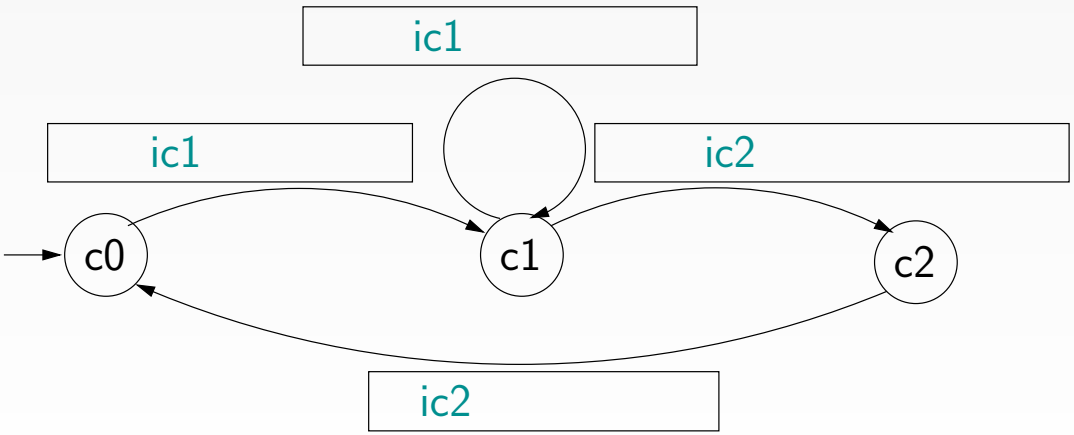




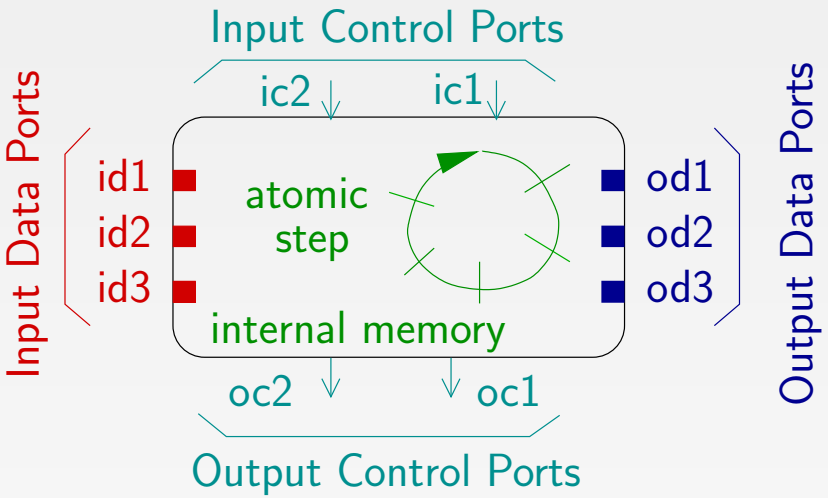
Control Contracts for 42 Components



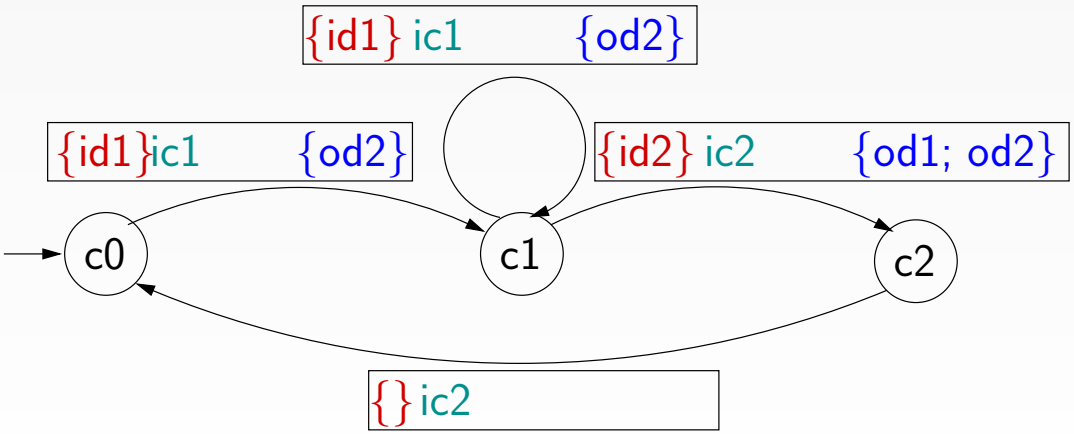
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state



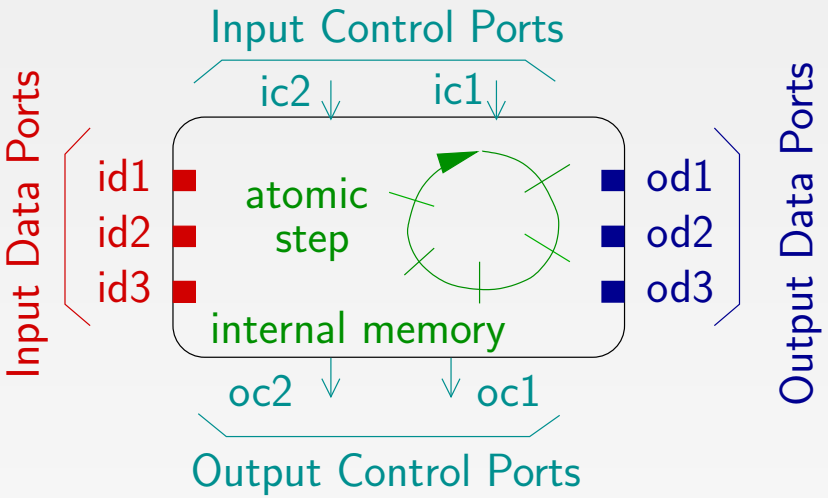
Control Contracts for 42 Components



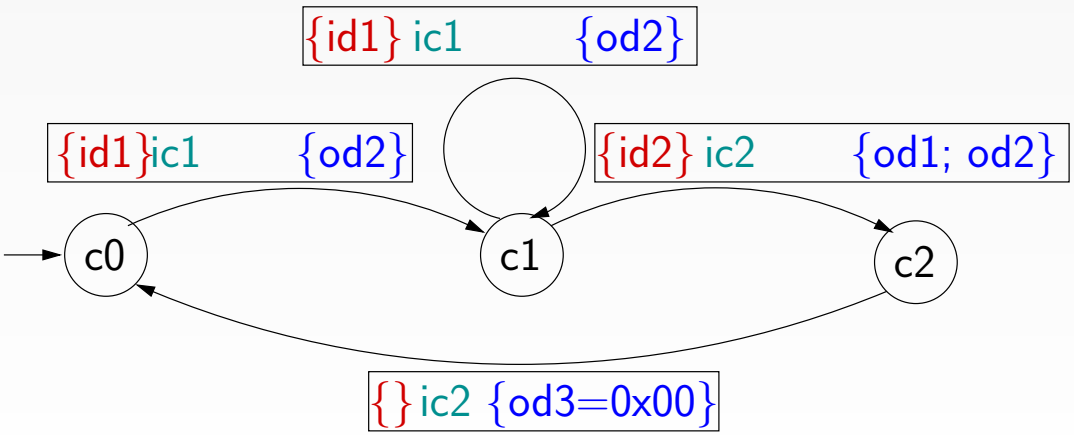
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state
- Data dependencies (Required, Provided)



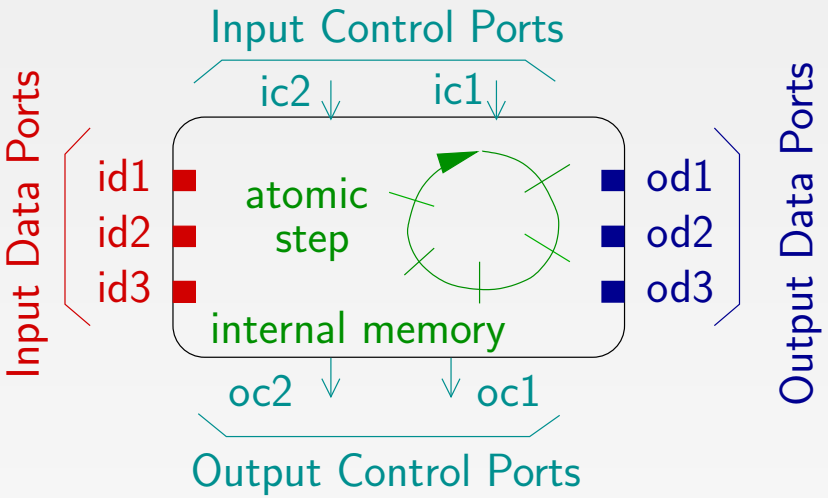
Control Contracts for 42 Components



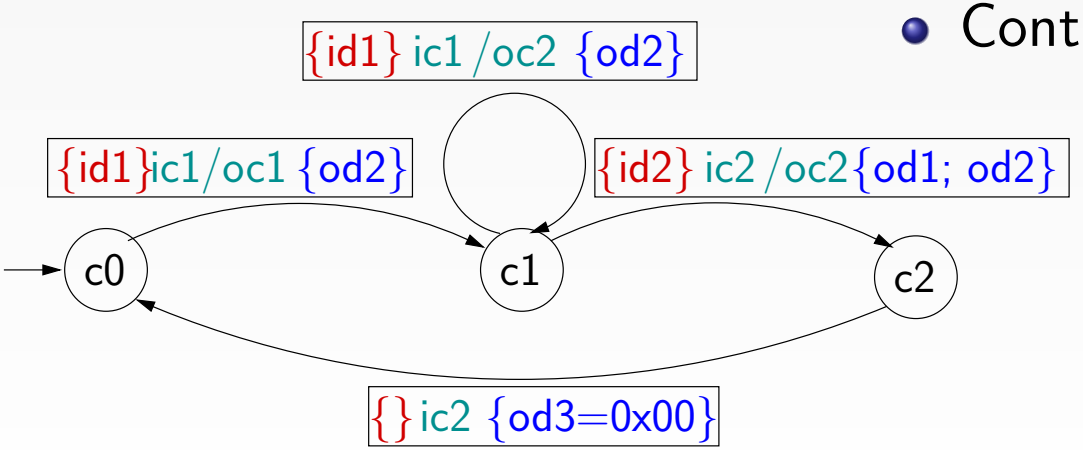
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state
- Data dependencies (Required, Provided)
  - Don't care about values except for particular data ports



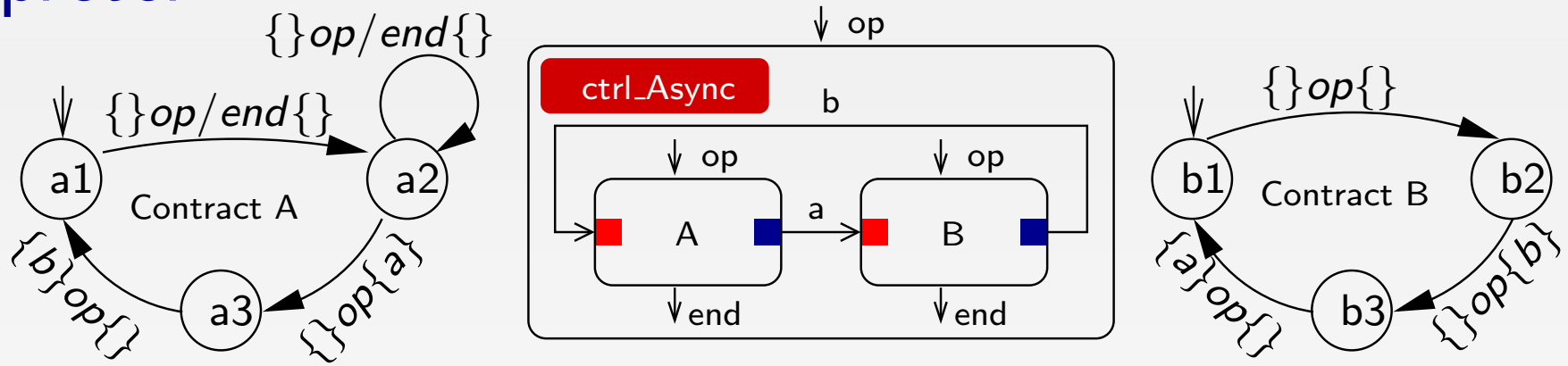
Control Contracts for 42 Components



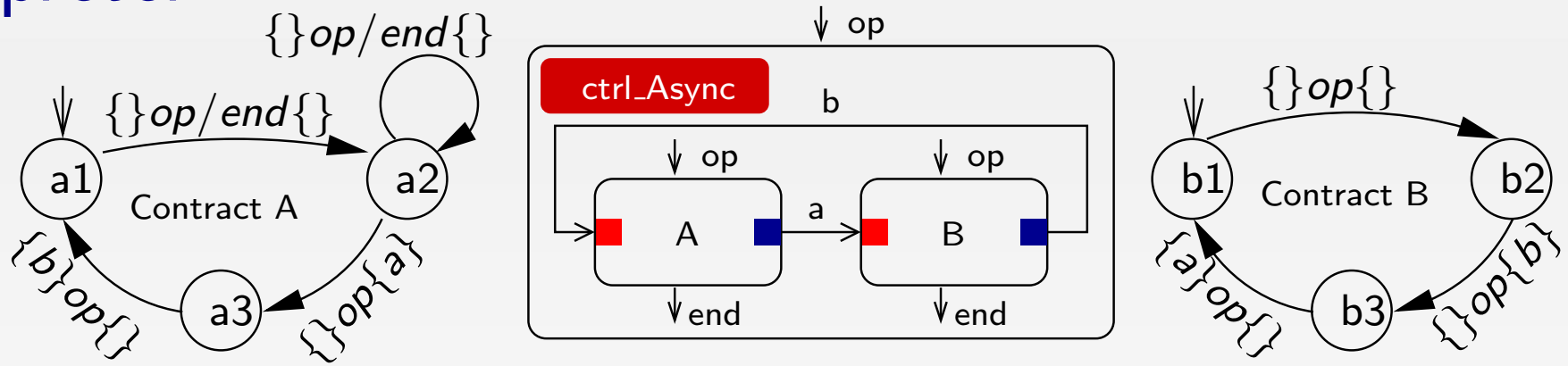
- Allowed activation sequences
  - It recognizes the correct sequence of activations since the first activation
  - Each state is an accepting state
- Data dependencies (Required, Provided)
  - Don't care about values except for particular data ports
- Control information



Asynchronous Simulation MoCC as a Contract Interpreter

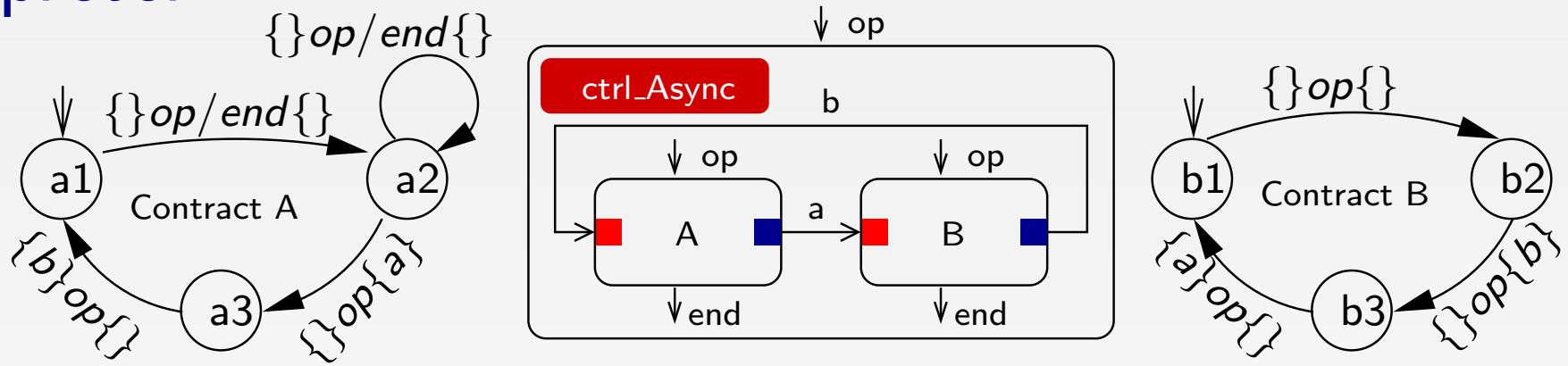


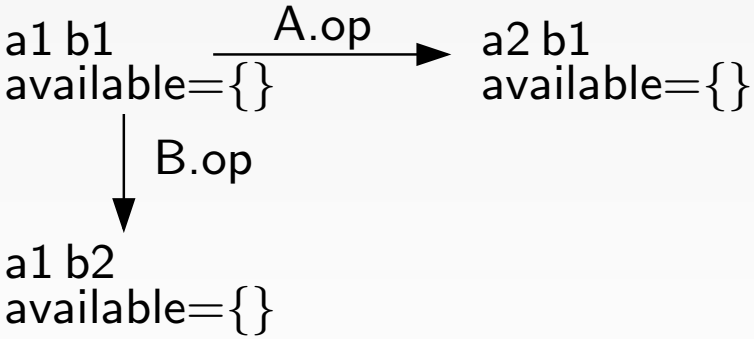
Asynchronous Simulation MoCC as a Contract Interpreter



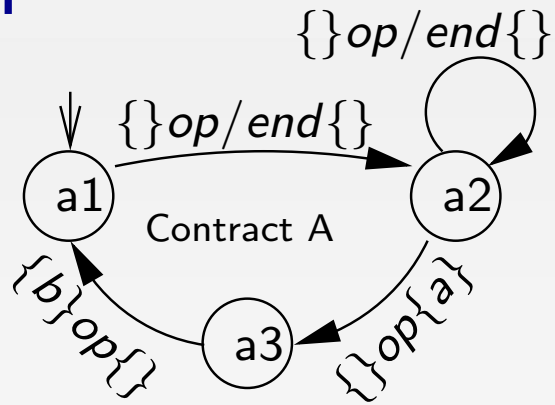
a1 b1
available={}

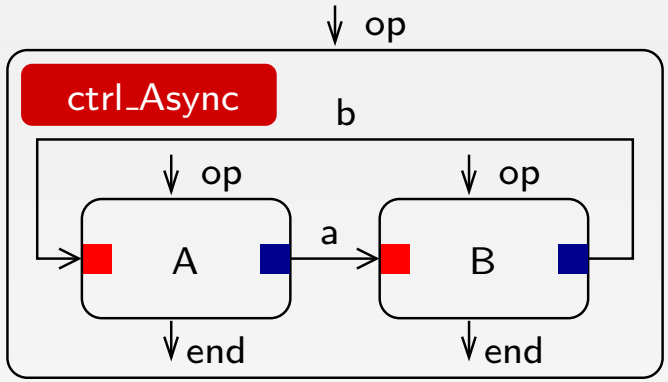
Asynchronous Simulation MoCC as a Contract Interpreter

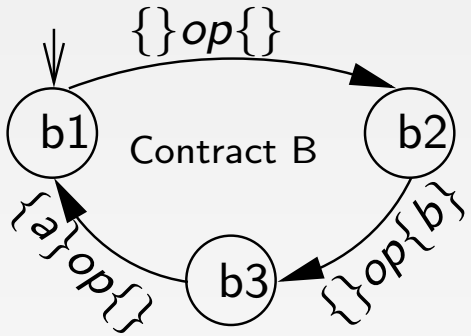


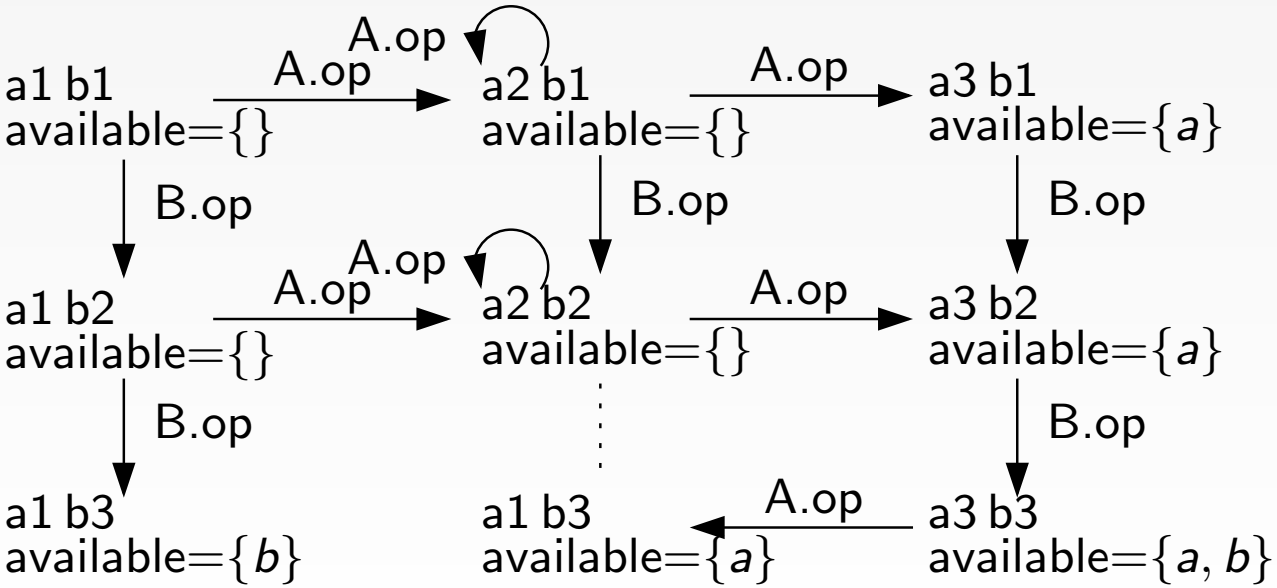


Asynchronous Simulation MoCC as a Contract Interpreter

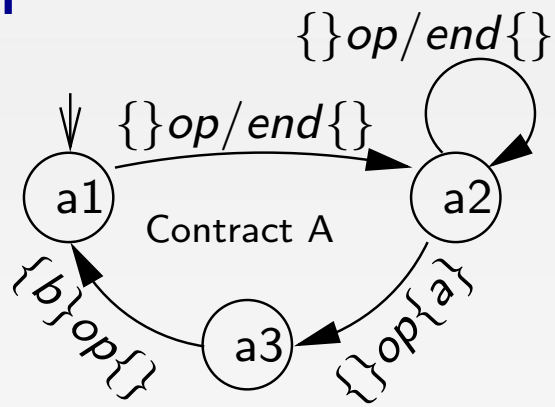


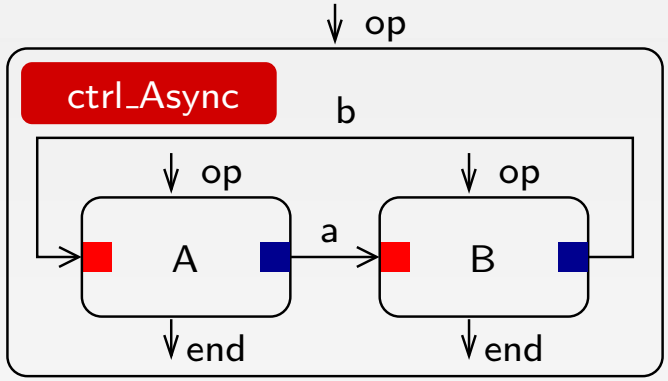


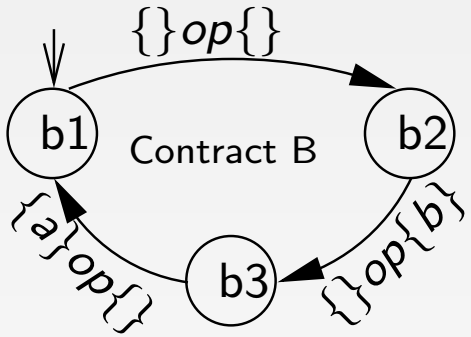


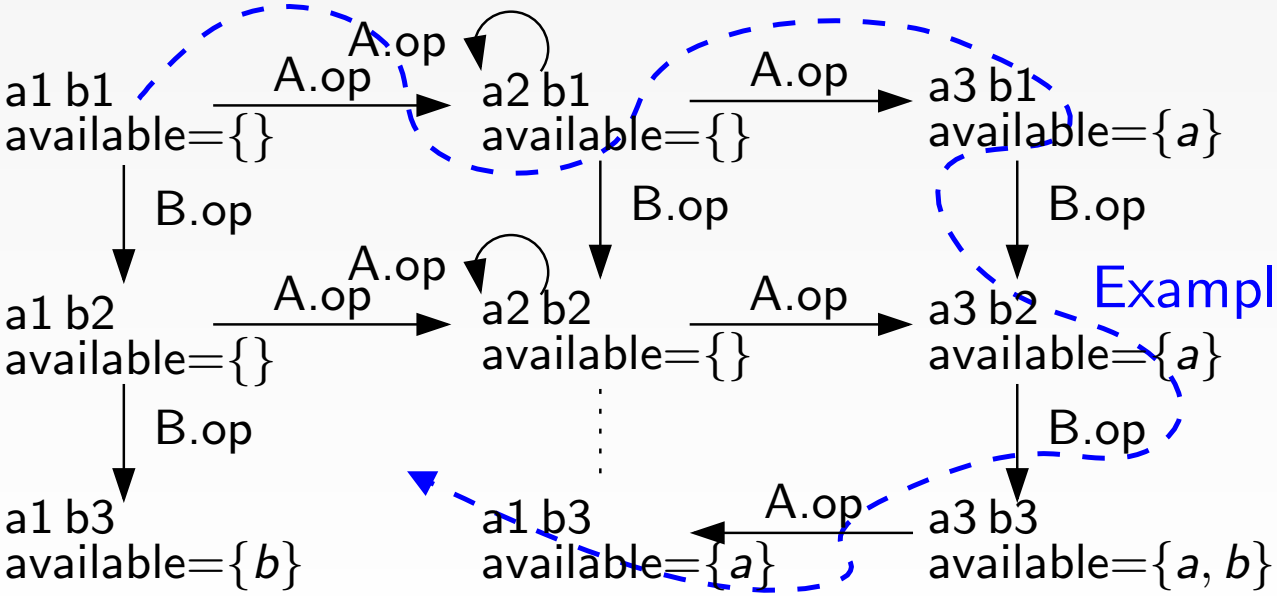


Asynchronous Simulation MoCC as a Contract Interpreter



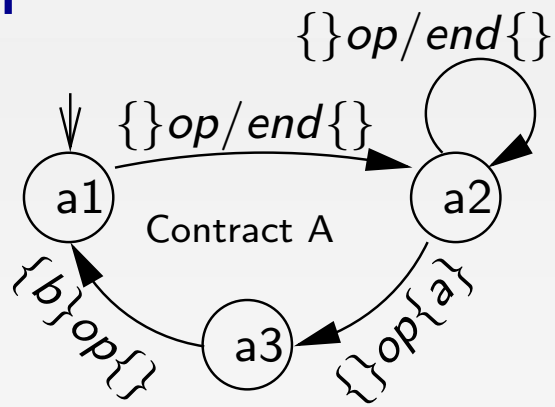


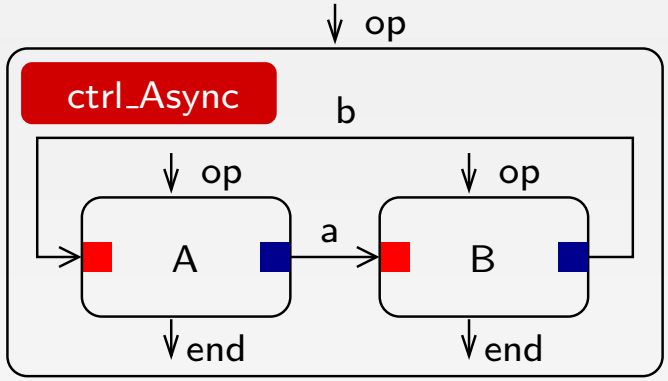


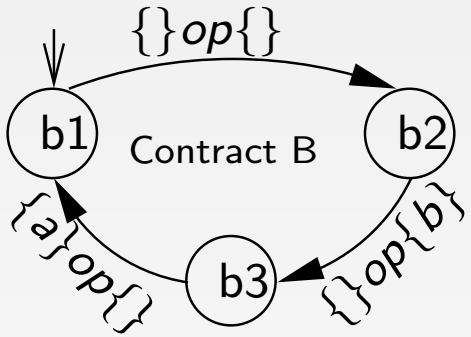


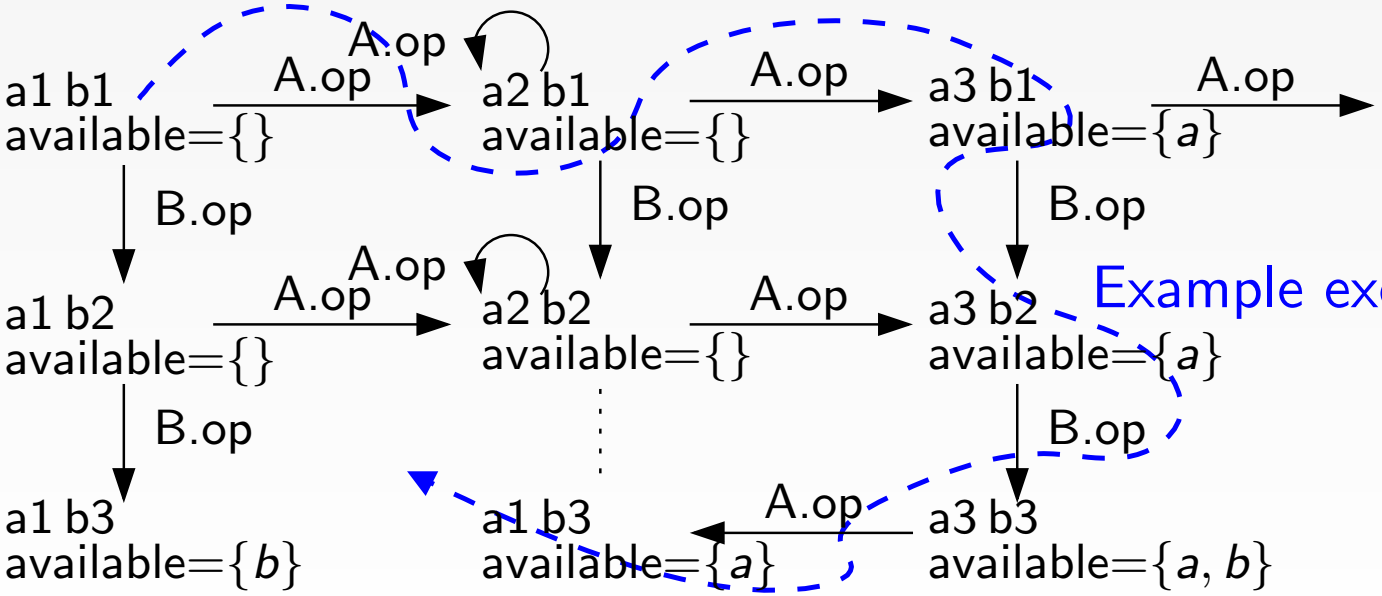
Example execution path

Asynchronous Simulation MoCC as a Contract Interpreter



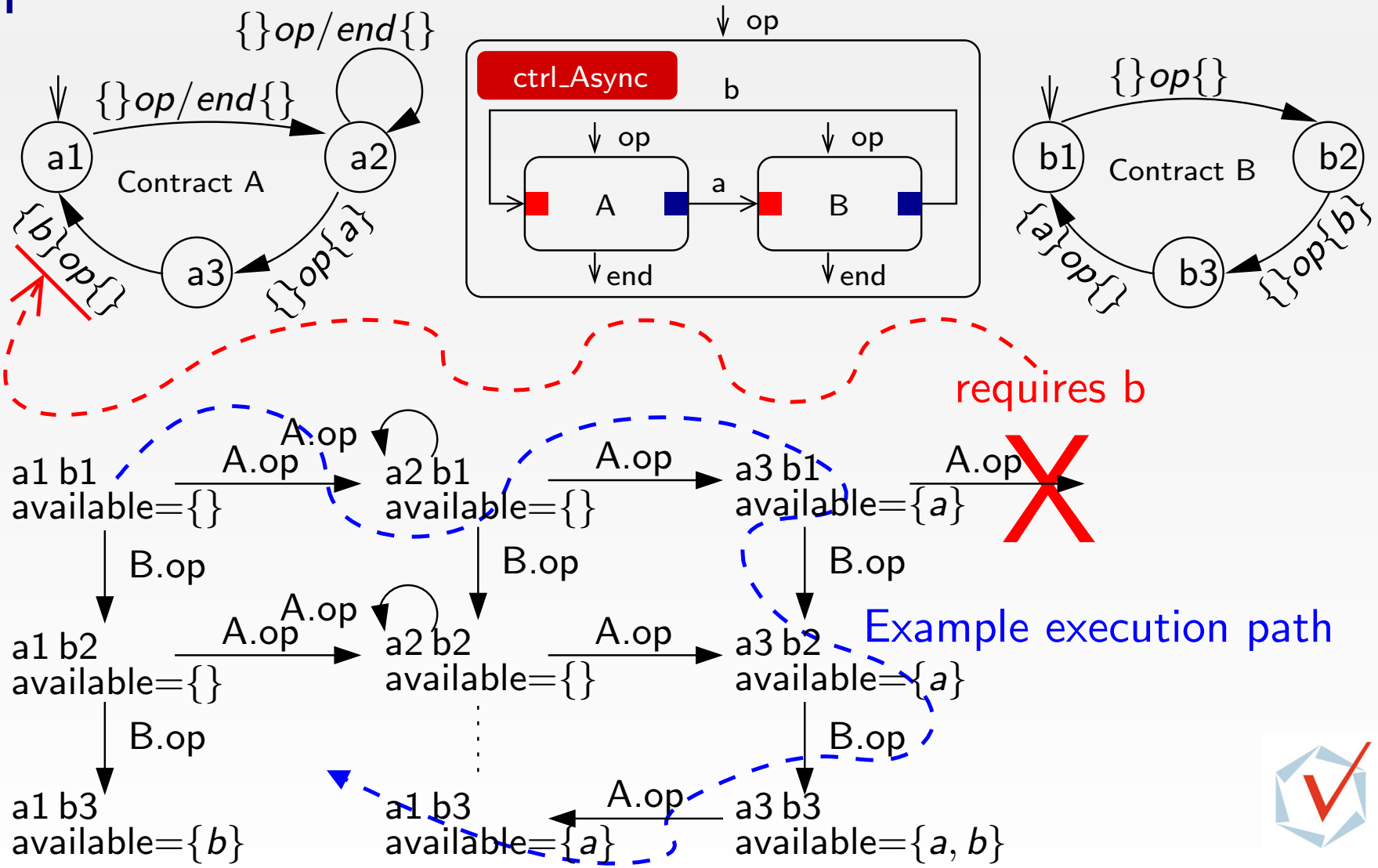






Example execution path

Asynchronous Simulation MoCC as a Contract Interpreter



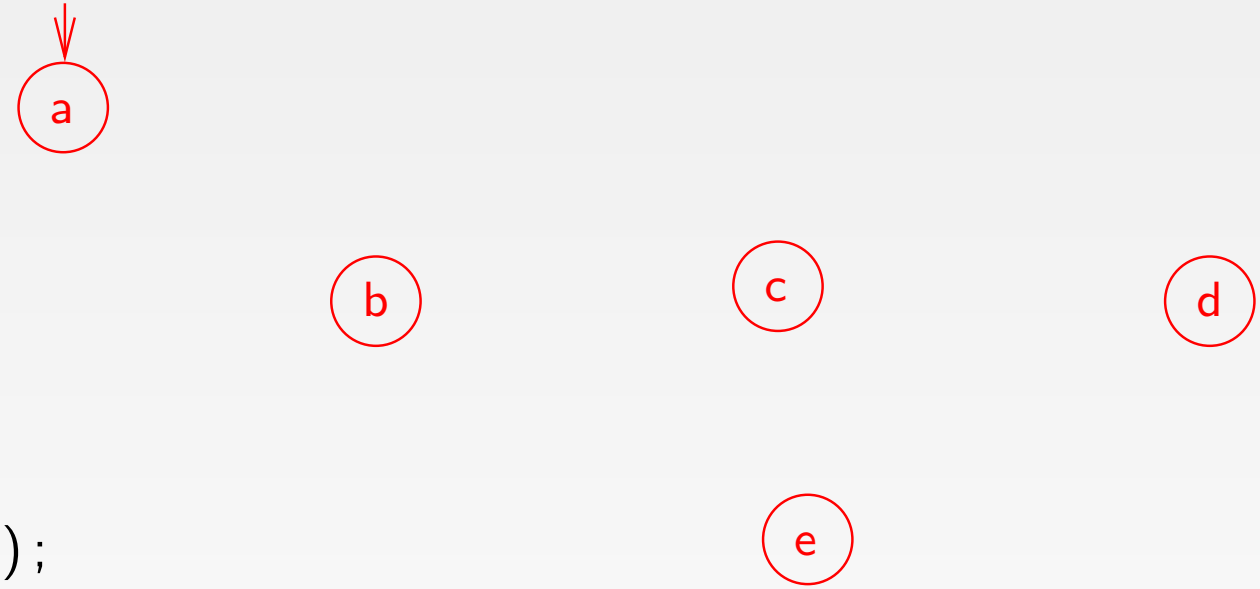
Contents

Extracting Control Contracts From SC-TLM Code

```
ⓐwhile(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑwait(e1);
        if(x < 42){
                ⓒp.f(x);
                ⓓp.g(x);
        }
        y=x+1;
        while(y < 5){
                y++;
                ⓔwait(e2);
        }
}
```

Extracting Control Contracts From SC-TLM Code

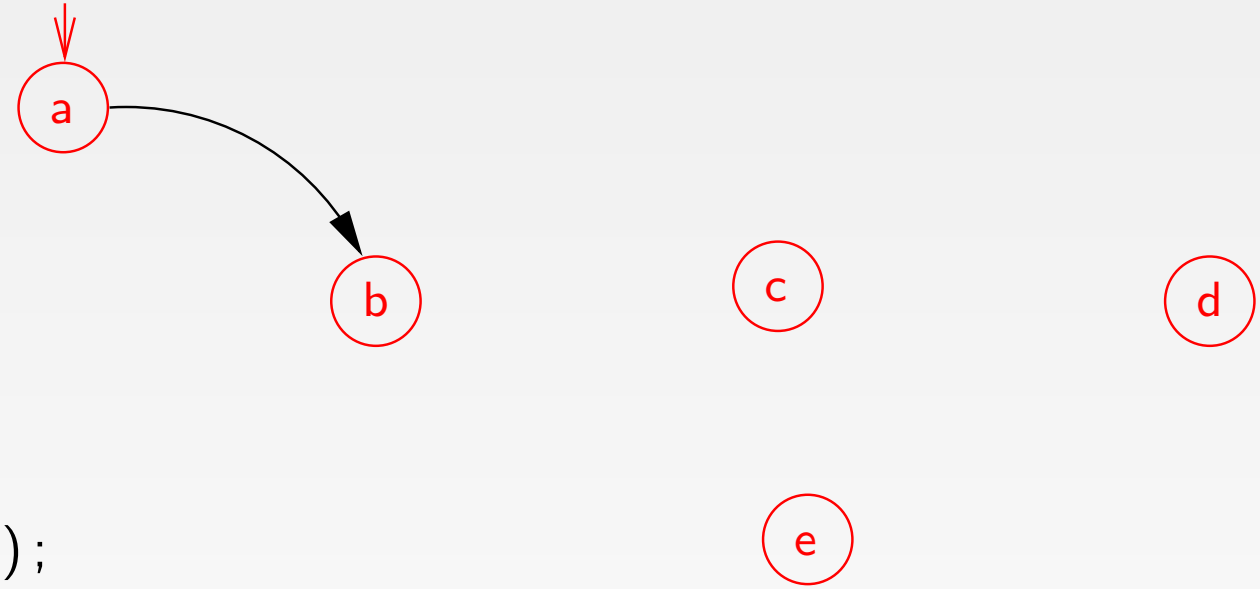
```
ⓐwhile(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑwait(e1);
        if(x < 42){
              ⓒp.f(x);
              ⓓp.g(x);
        }
        y=x+1;
        while(y < 5){
              y++;
              ⓔwait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.

Extracting Control Contracts From SC-TLM Code

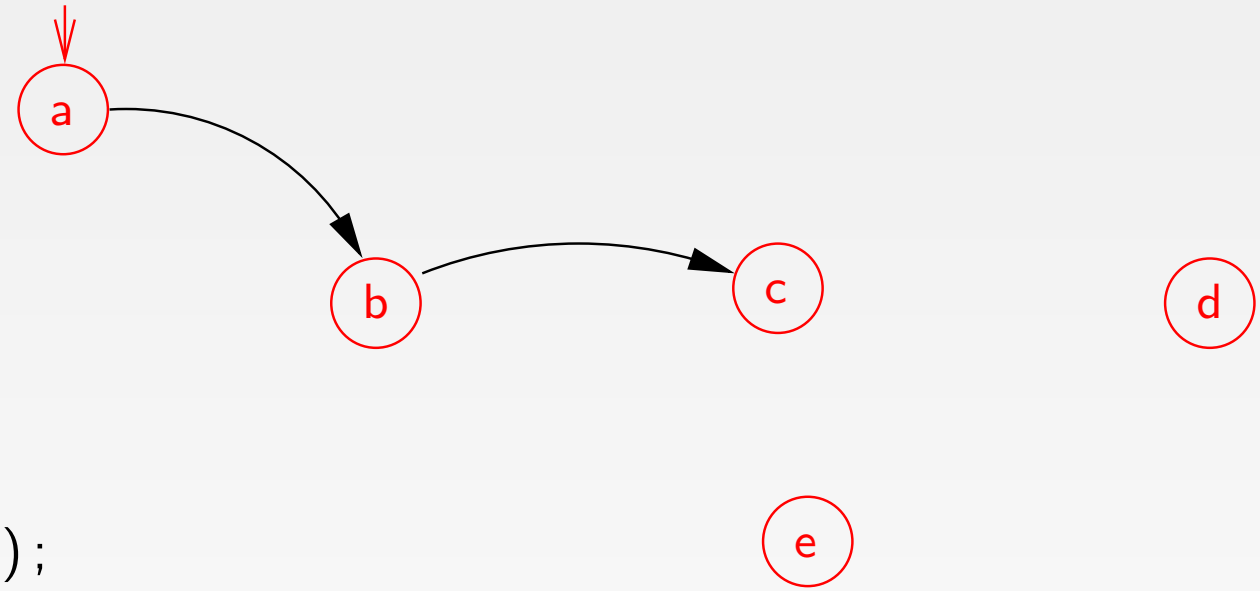
```
a while(true){
        x++;
        e3.notify();
        e4.notify();
        b wait(e1);
        if(x < 42){
            c p.f(x);
            d p.g(x);
        }
        y=x+1;
        while(y < 5){
            y++;
            e wait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.

Extracting Control Contracts From SC-TLM Code

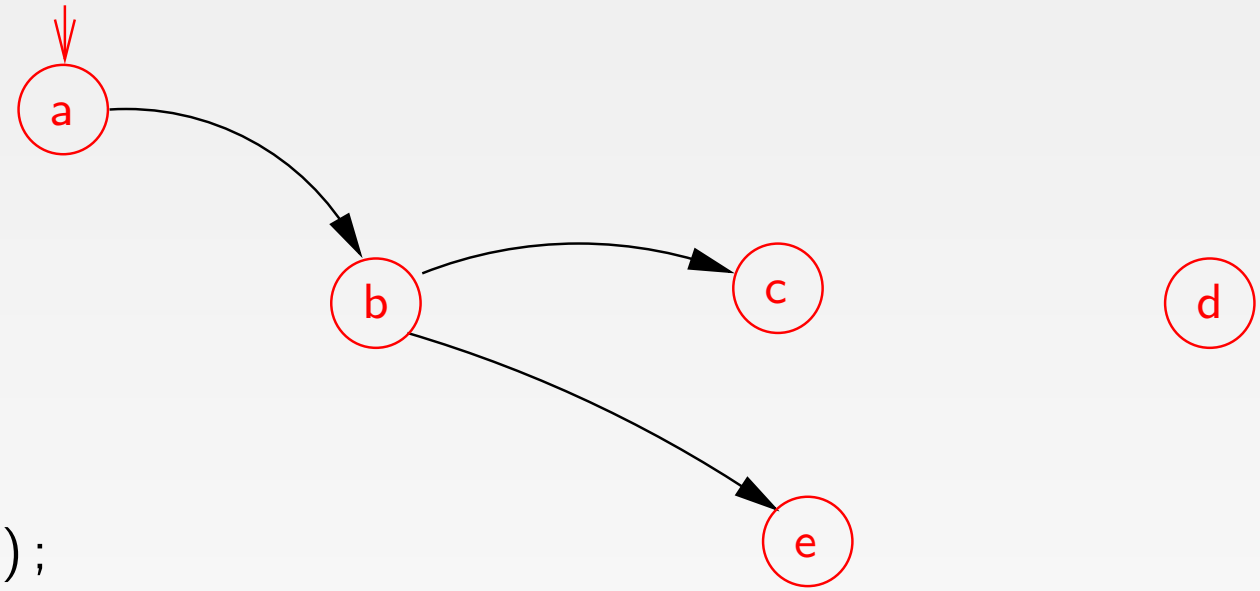
```
ⓐ while(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑ wait(e1);
        if(x < 42){
              ⓒ p.f(x);
              ⓓ p.g(x);
        }
        y=x+1;
        while(y < 5){
              y++;
              ⓔ wait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.

Extracting Control Contracts From SC-TLM Code

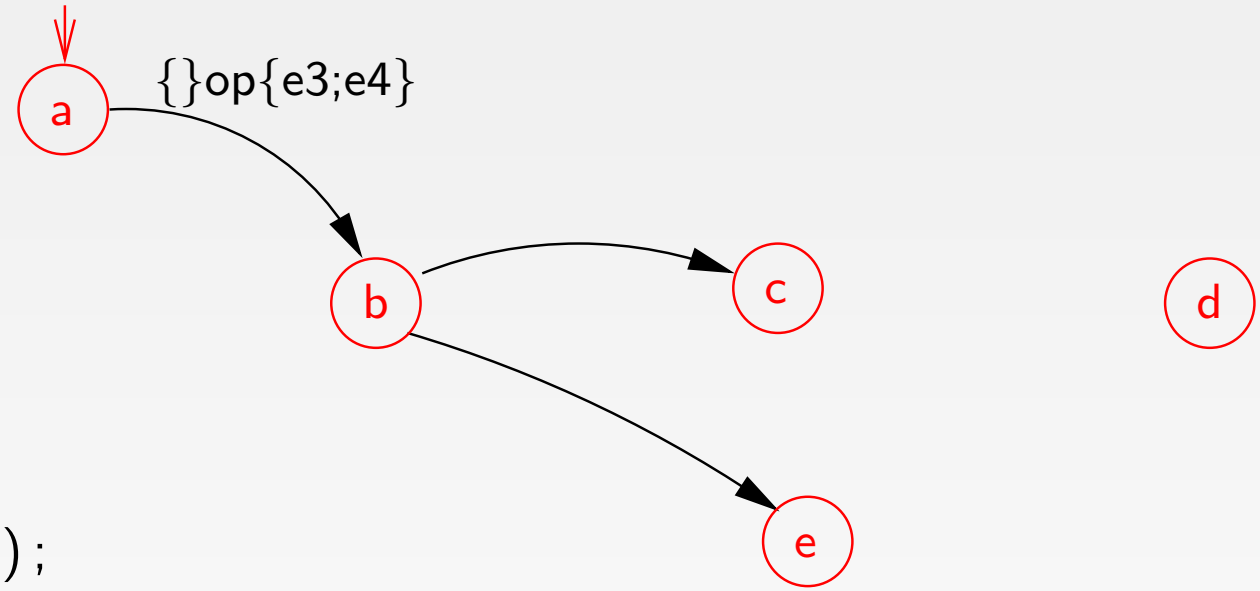
```
ⓐ while(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑ wait(e1);
        if(x < 42){
              ⓒ p.f(x);
              ⓓ p.g(x);
        }
        y=x+1;
        while(y < 5){
              y++;
              ⓔ wait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.
- Data values are abstracted

Extracting Control Contracts From SC-TLM Code

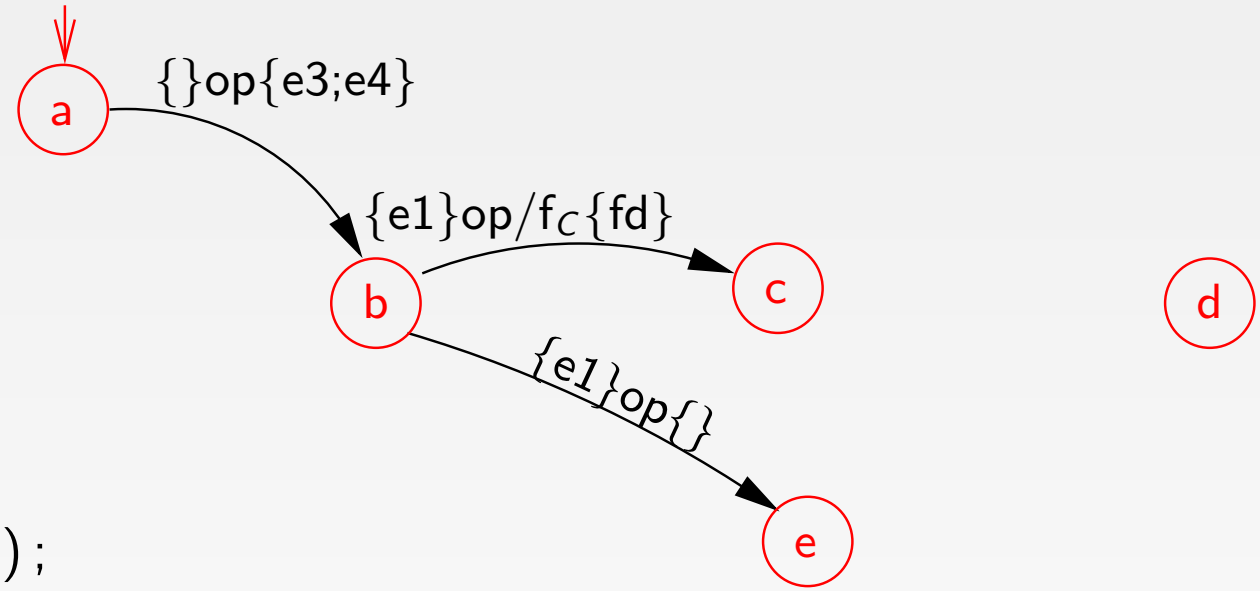
```
ⓐ while(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑ wait(e1);
        if(x < 42){
              ⓒ p.f(x);
              ⓓ p.g(x);
        }
        y=x+1;
        while(y < 5){
              y++;
              ⓔ wait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.
- Data values are abstracted

Extracting Control Contracts From SC-TLM Code

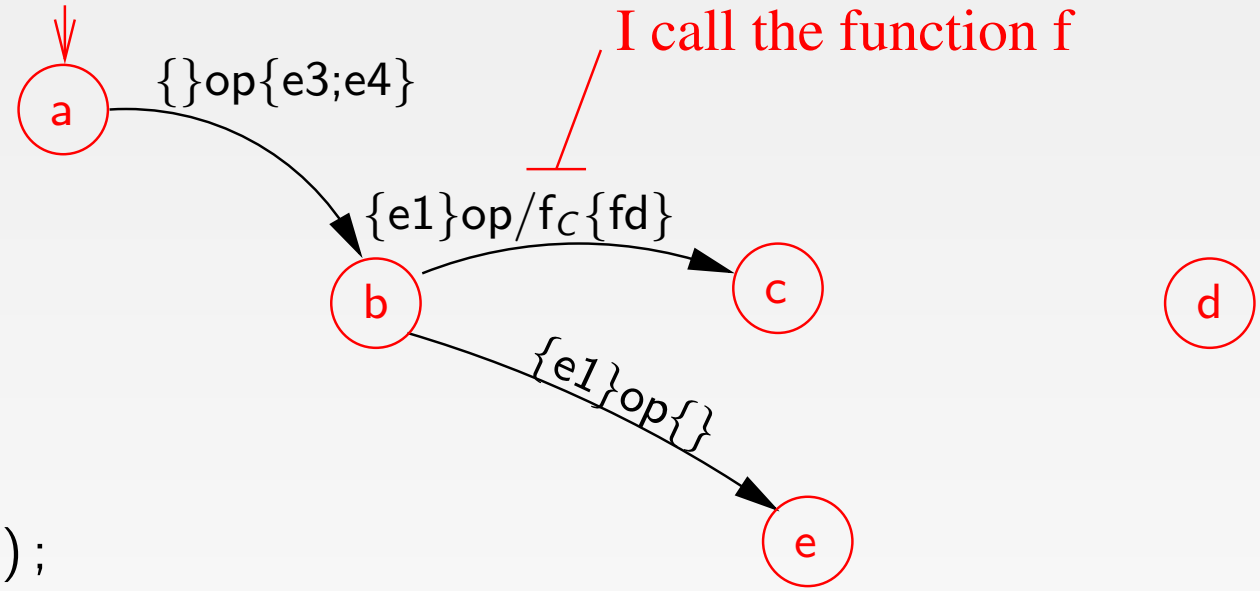
```
ⓐ while(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑ wait(e1);
        if(x < 42){
              ⓒ p.f(x);
              ⓓ p.g(x);
        }
        y=x+1;
        while(y < 5){
              y++;
              ⓔ wait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.
- Data values are abstracted

Extracting Control Contracts From SC-TLM Code

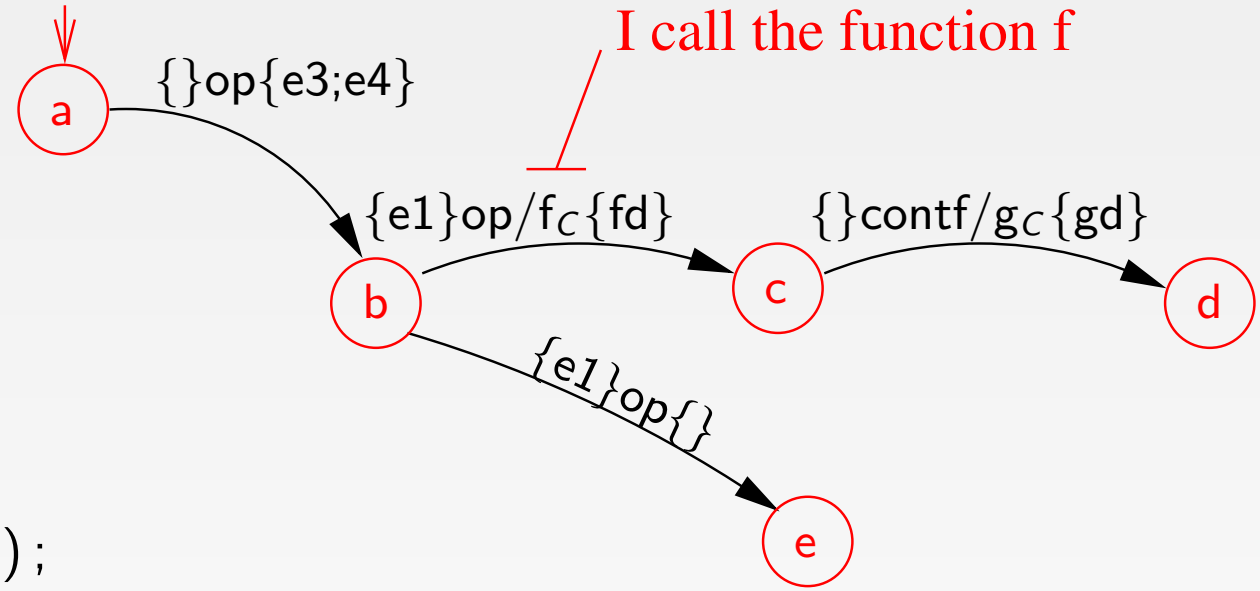
```
ⓐ while(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑ wait(e1);
        if(x < 42){
              ⓒ p.f(x);
              ⓓ p.g(x);
        }
        y=x+1;
        while(y < 5){
              y++;
              ⓔ wait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.
- Data values are abstracted

Extracting Control Contracts From SC-TLM Code

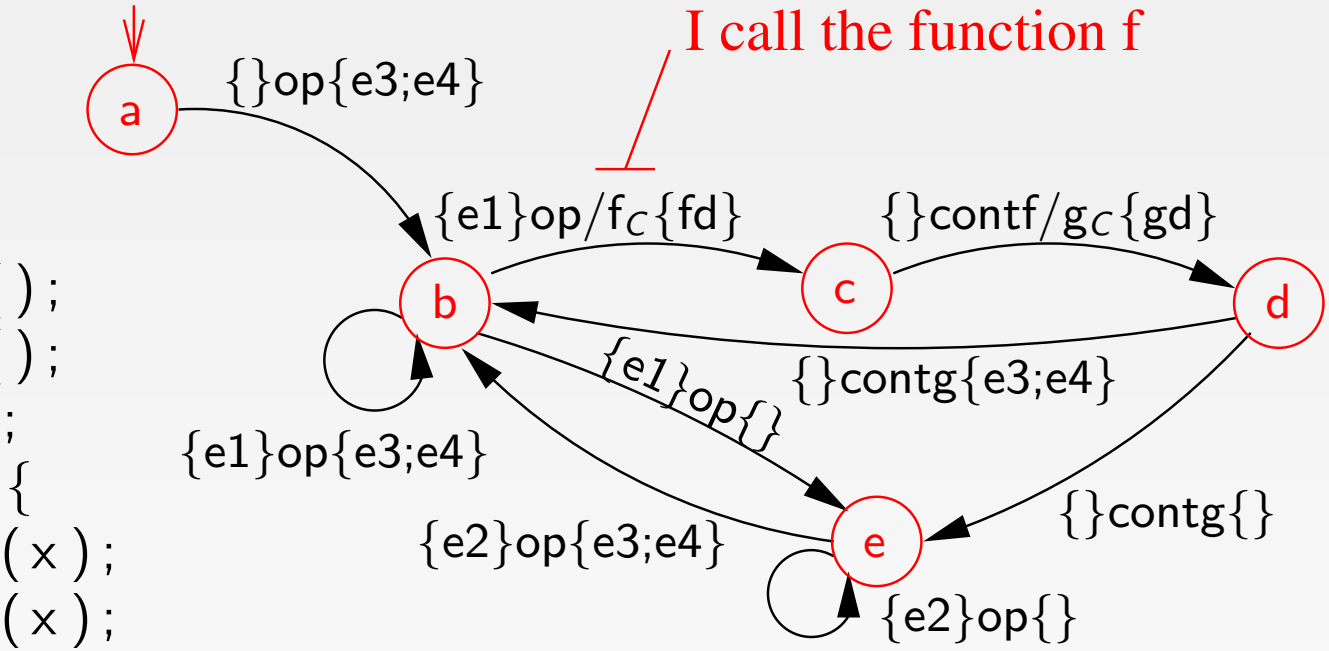
```
ⓐwhile(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑwait(e1);
        if(x < 42){
              ⓒp.f(x);
              ⓓp.g(x);
        }
        y=x+1;
        while(y < 5){
              y++;
              ⓔwait(e2);
        }
}
```



- The component loses Control Flow in wait and function calls.
- Data values are abstracted

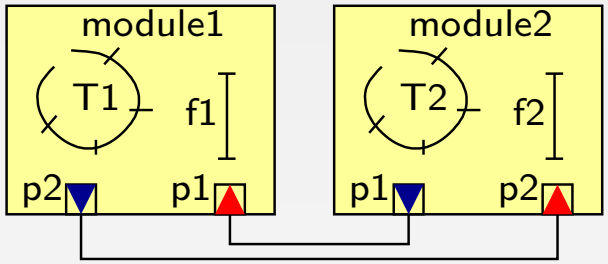
Extracting Control Contracts From SC-TLM Code

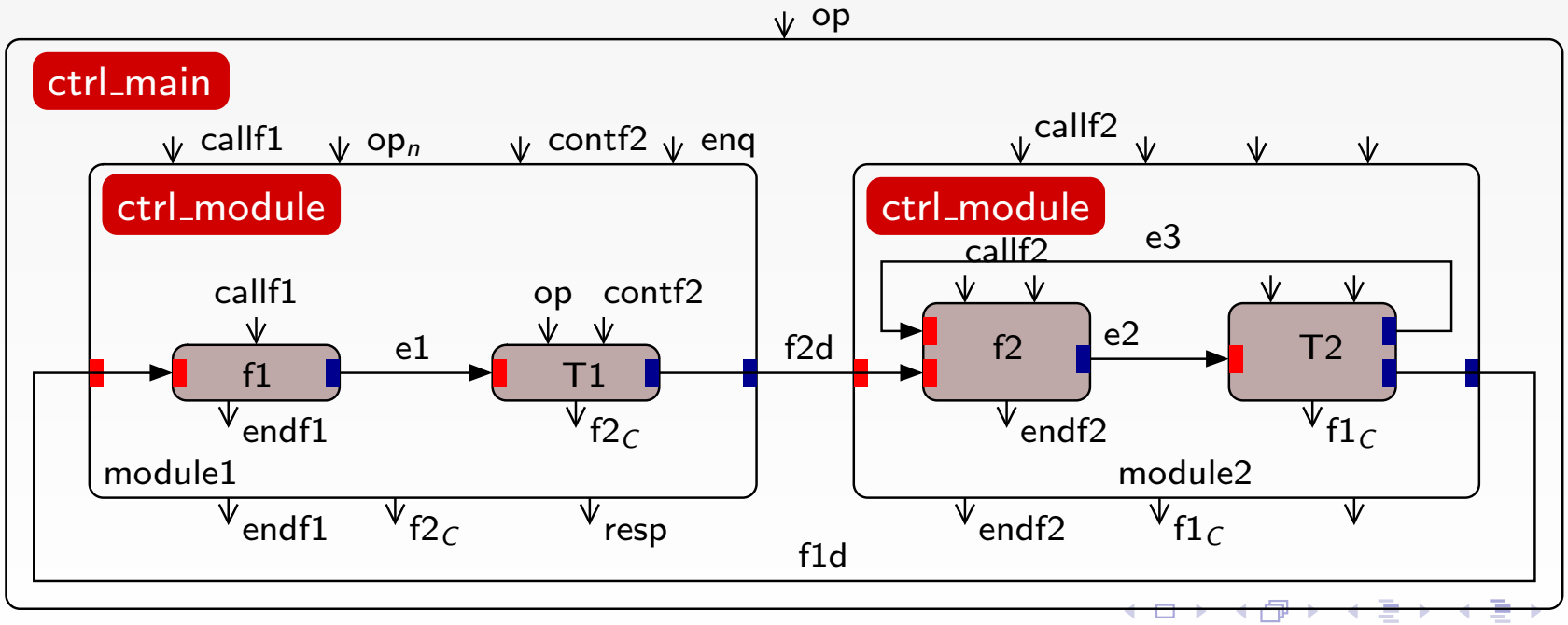
```
ⓐ while(true){
      x++;
      e3.notify();
      e4.notify();
      ⓑ wait(e1);
      if(x < 42){
          ⓒ p.f(x);
          ⓓ p.g(x);
      }
      y=x+1;
      while(y < 5){
          y++;
          ⓔ wait(e2);
      }
}
```



- The component loses Control Flow in wait and function calls.
- Data values are abstracted

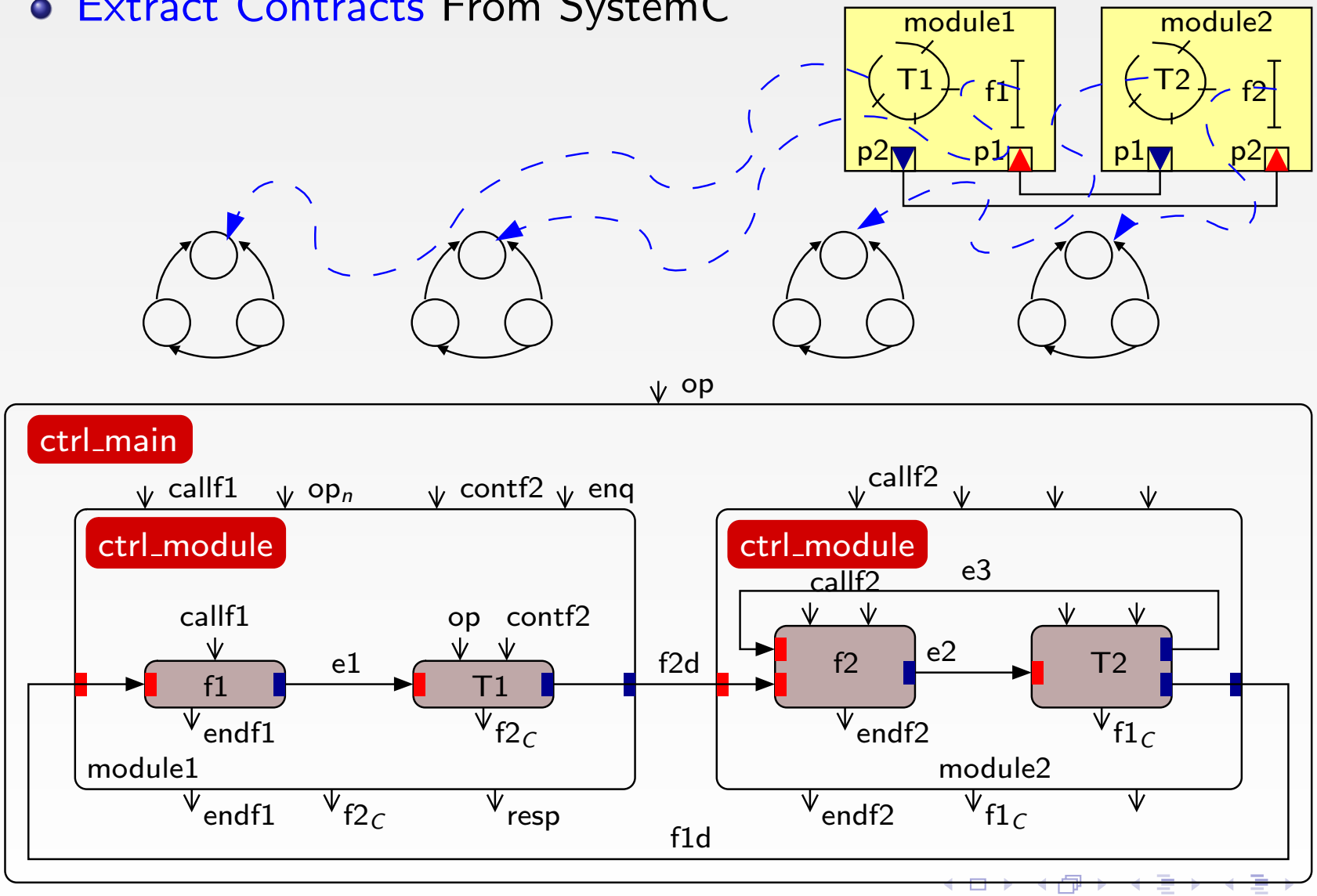
Lightweight Execution for SystemC-TLM





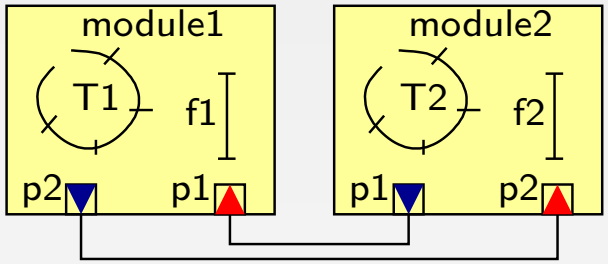
Lightweight Execution for SystemC-TLM

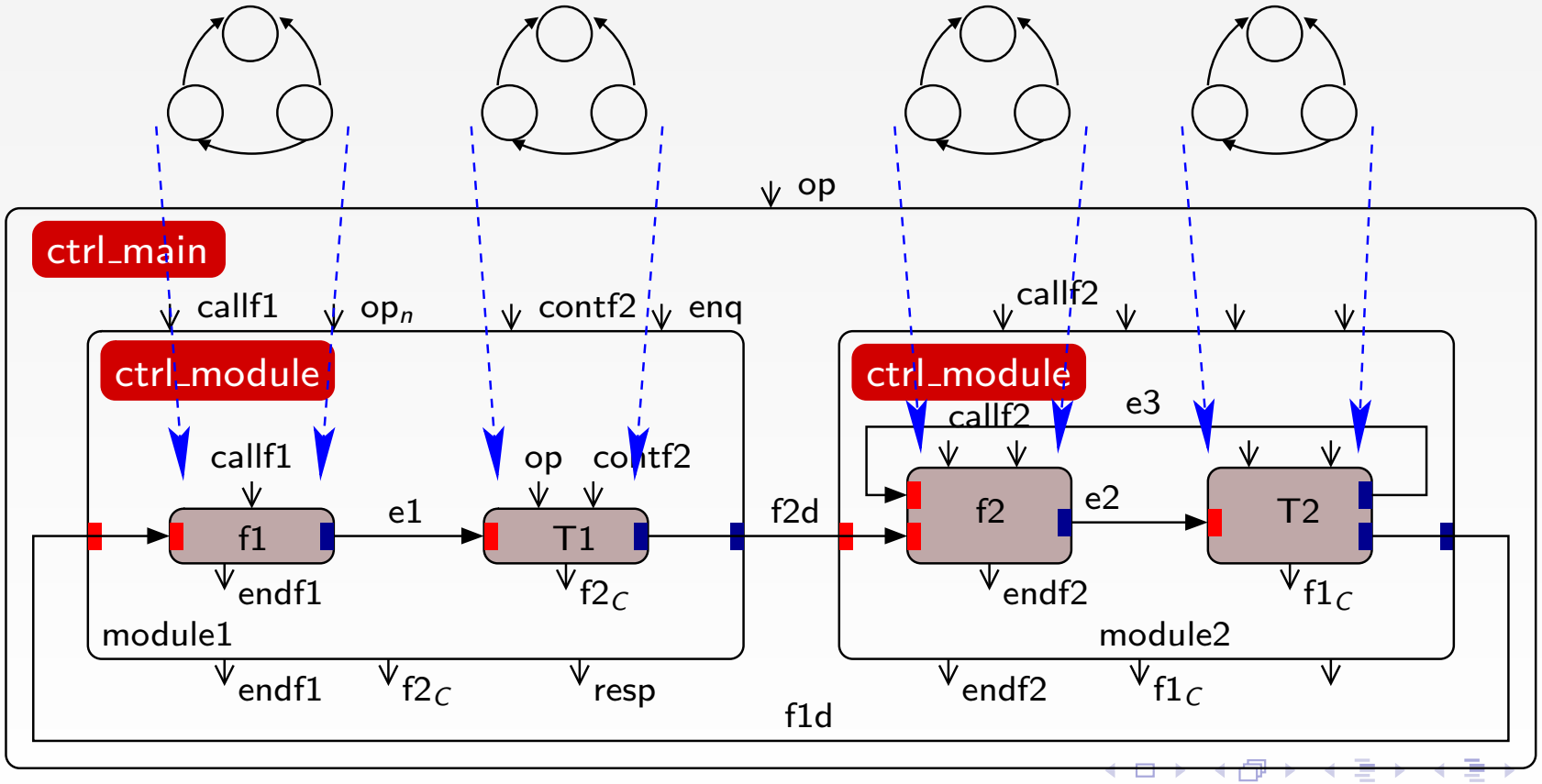
- Extract Contracts From SystemC



Lightweight Execution for SystemC-TLM

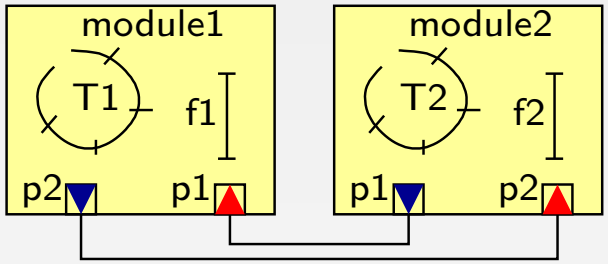
- Extract Contracts From SystemC
- Execute Contracts instead of Code

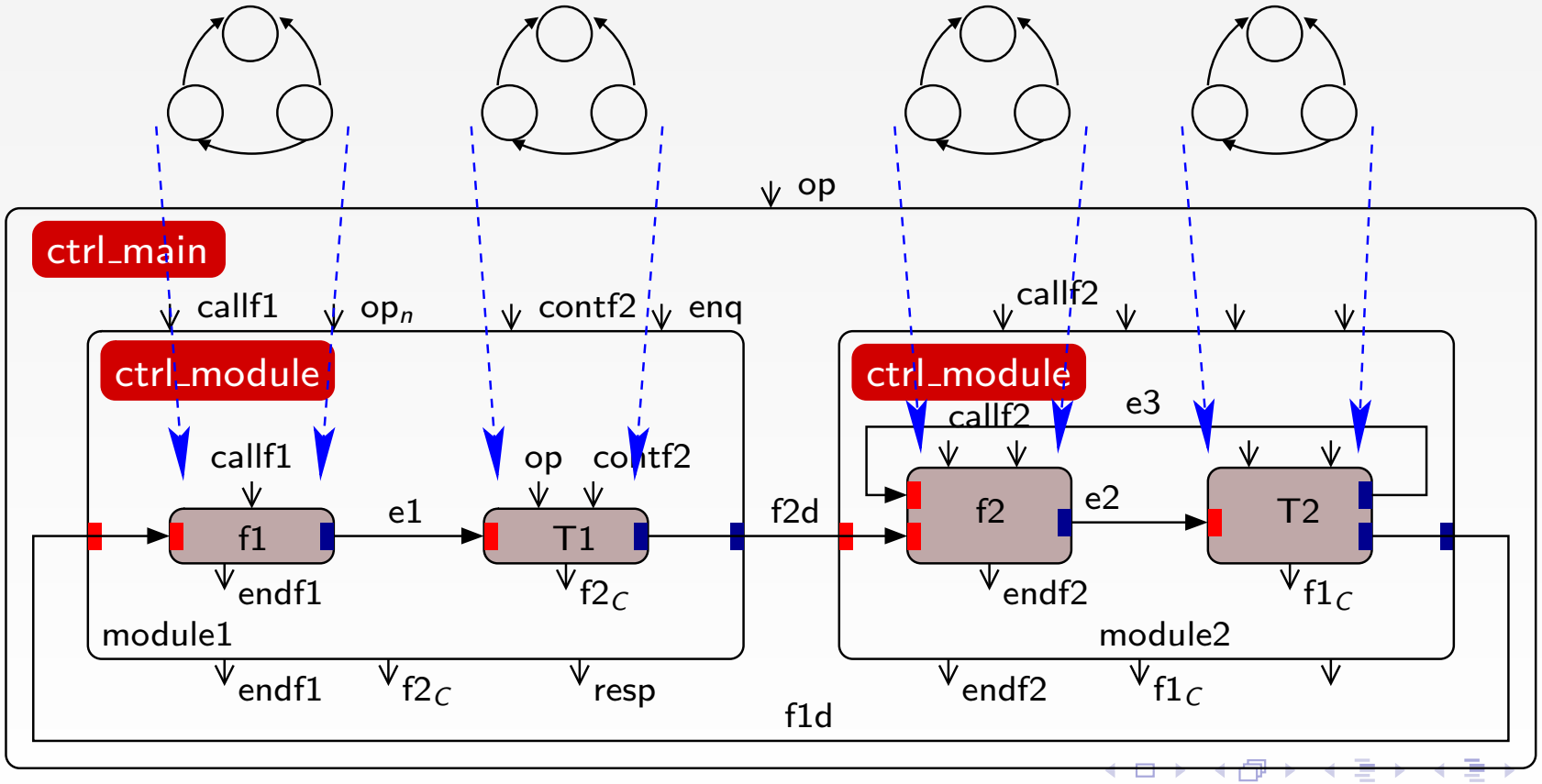




Lightweight Execution for SystemC-TLM

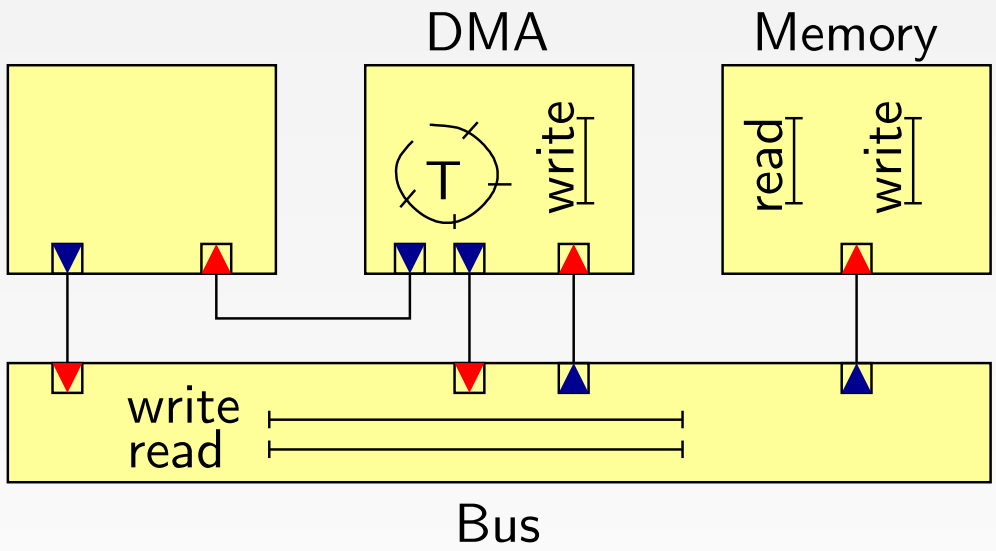
- Extract Contracts From SystemC
- Execute Contracts instead of Code
- Produces a super-set of SystemC executions



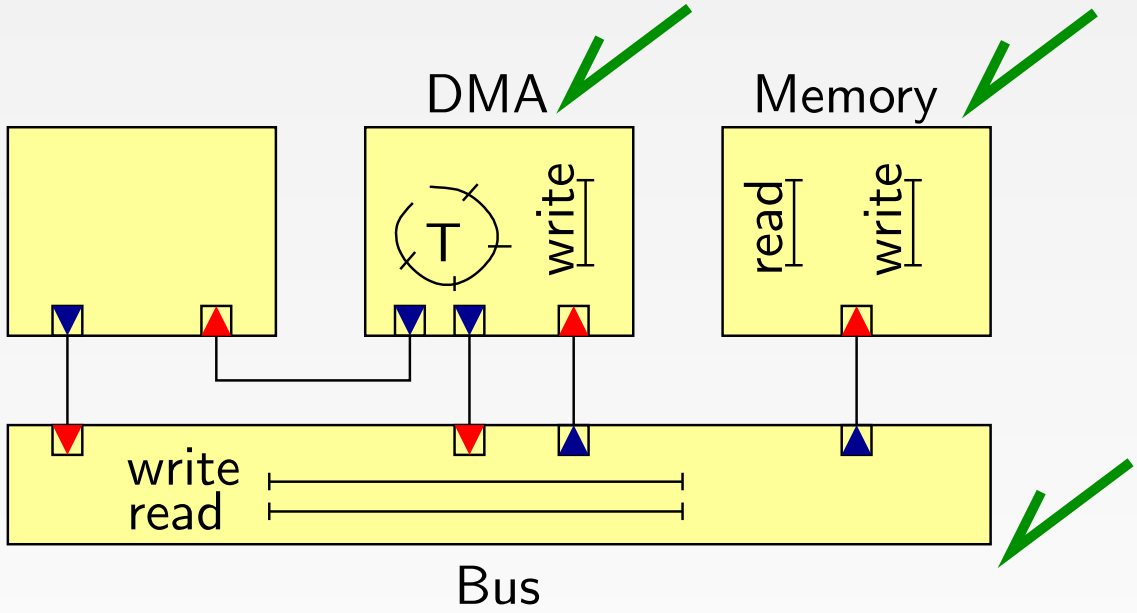


Contents

TL-Modeling with SystemC: an Example

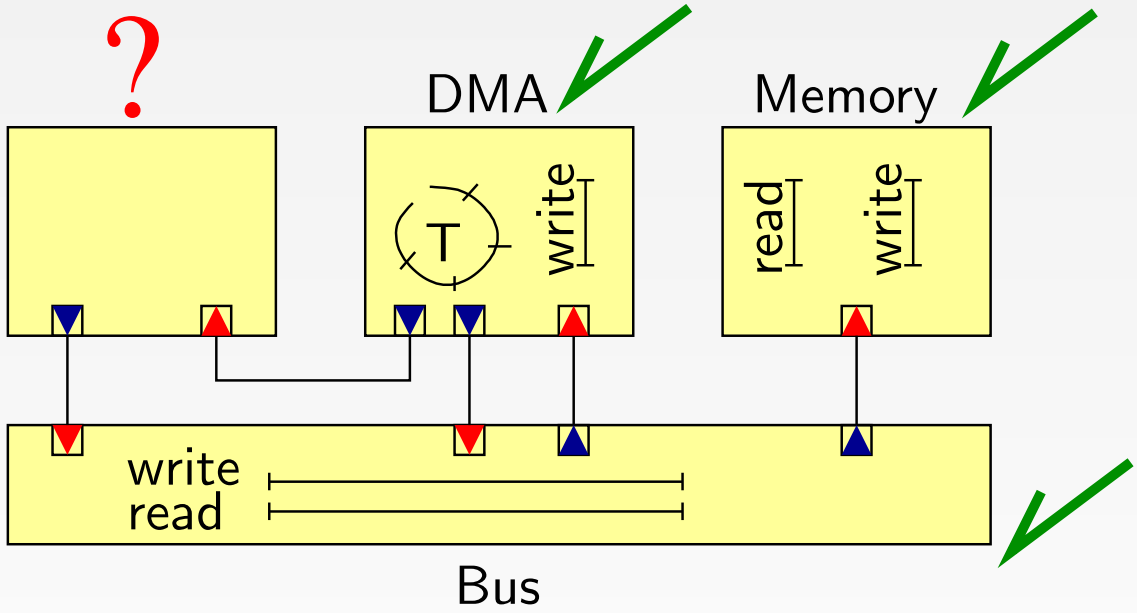


TL-Modeling with SystemC: an Example



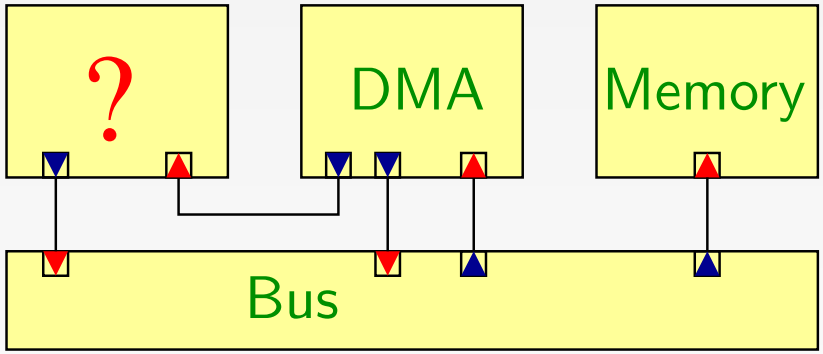
- Reuse of existing components (e.g., DMA, Memory, Bus)

TL-Modeling with SystemC: an Example

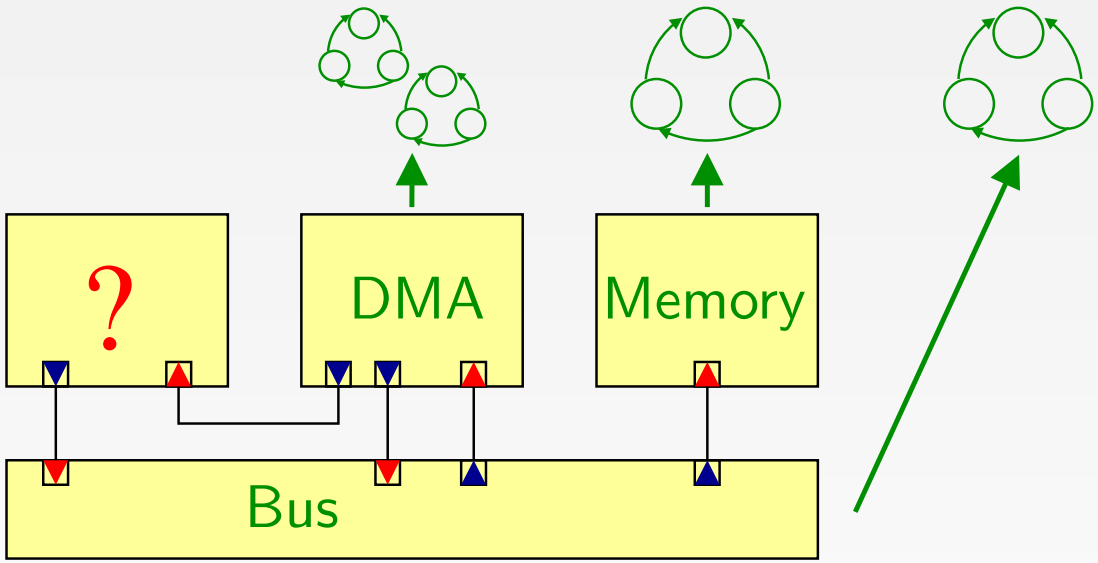


- Reuse of existing components (e.g., DMA, Memory, Bus)
- Part of the functionality is not implemented

Contract-Based Execution of the System

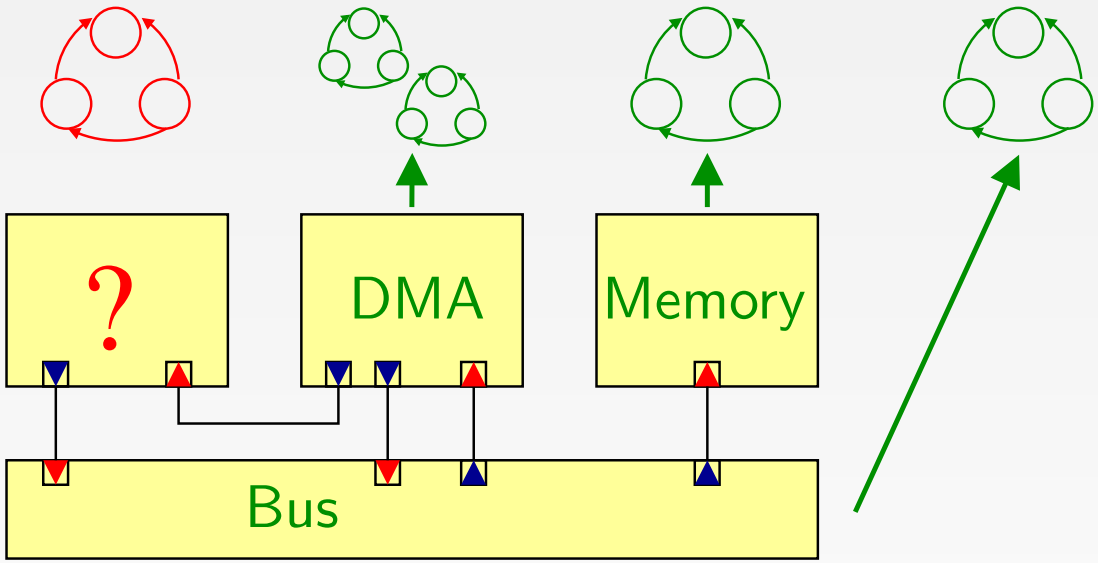


Contract-Based Execution of the System



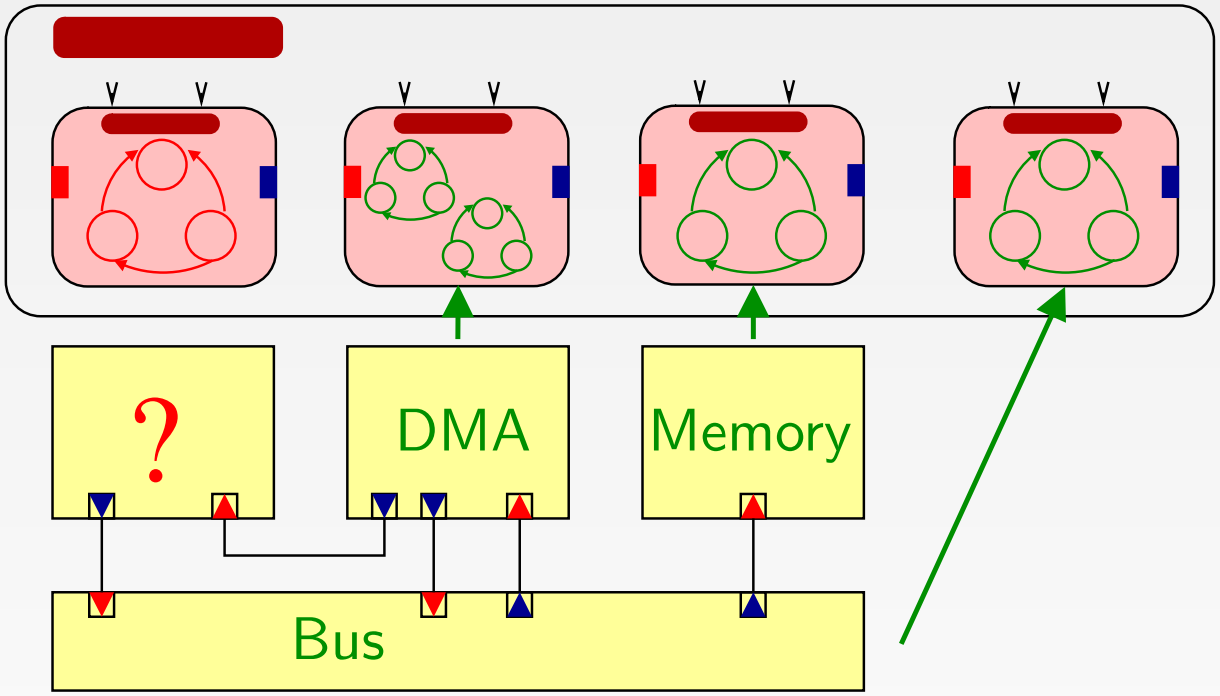
- Extract control contracts from existing SC-TLM components

Contract-Based Execution of the System



- Extract control contracts from existing SC-TLM components
- Write new contracts for the missing ones

Contract-Based Execution of the System



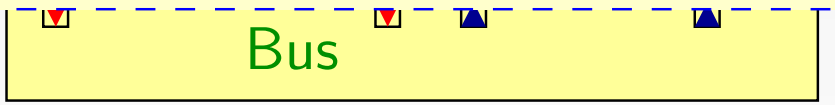
- Extract control contracts from existing SC-TLM components
- Write new contracts for the missing ones
- Execute the contracts with 42 controllers

Contract-Based Execution of the System

- If there is a bug, the new contract is not compatible with the extracted ones

- A bug may lead the simulation to a deadlock

- Safety properties may be monitored by means of observers



- Extract control contracts from existing SC-TLM components
- Write new contracts for the missing ones
- Execute the contracts with 42 controllers
- $\Rightarrow$ Find Synchronization Bugs

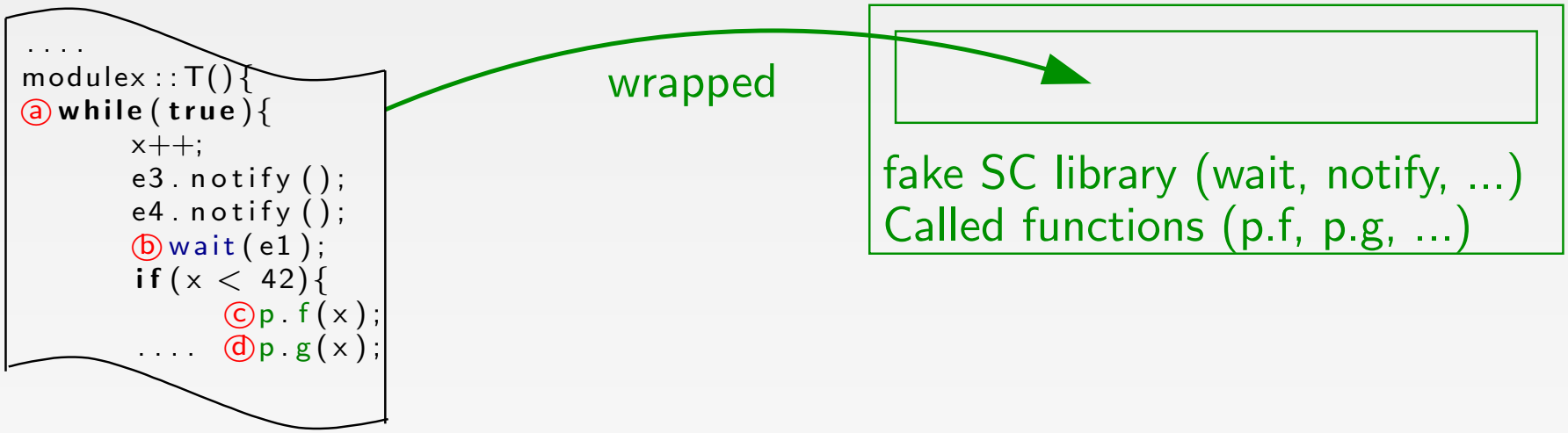
Wrapping SystemC Code

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑwait(e1);
        if(x < 42){
              ⓒp.f(x);
        ....  ⓓp.g(x);
```

Wrapping SystemC Code

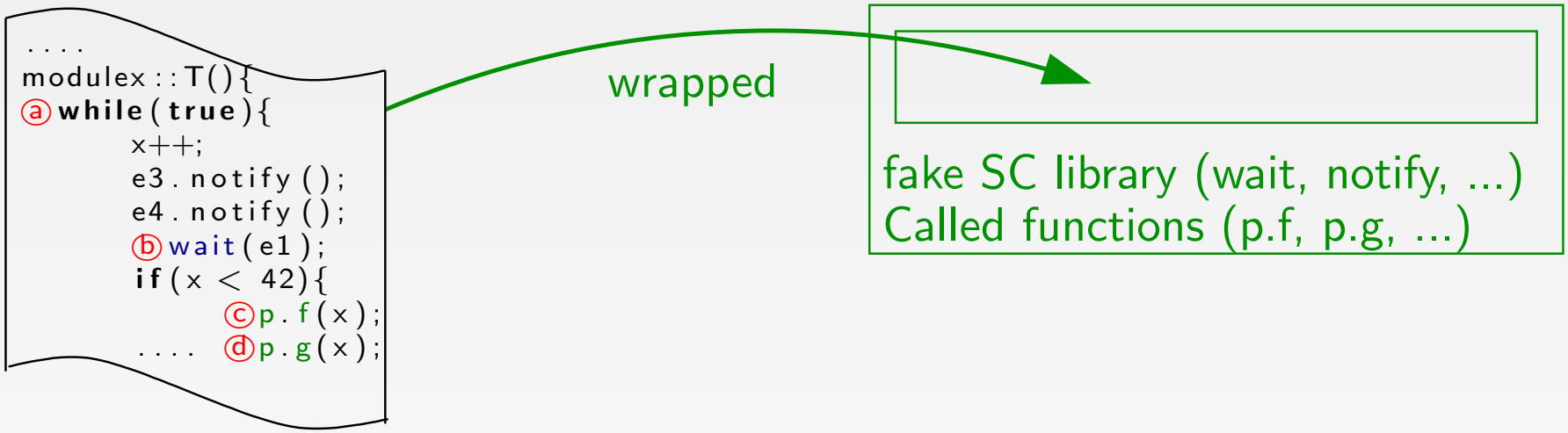
Component implementation:



The SystemC/TLM implementation is wrapped in a fake SystemC library.

Wrapping SystemC Code

Component implementation:



The SystemC/TLM implementation is wrapped in a fake SystemC library.

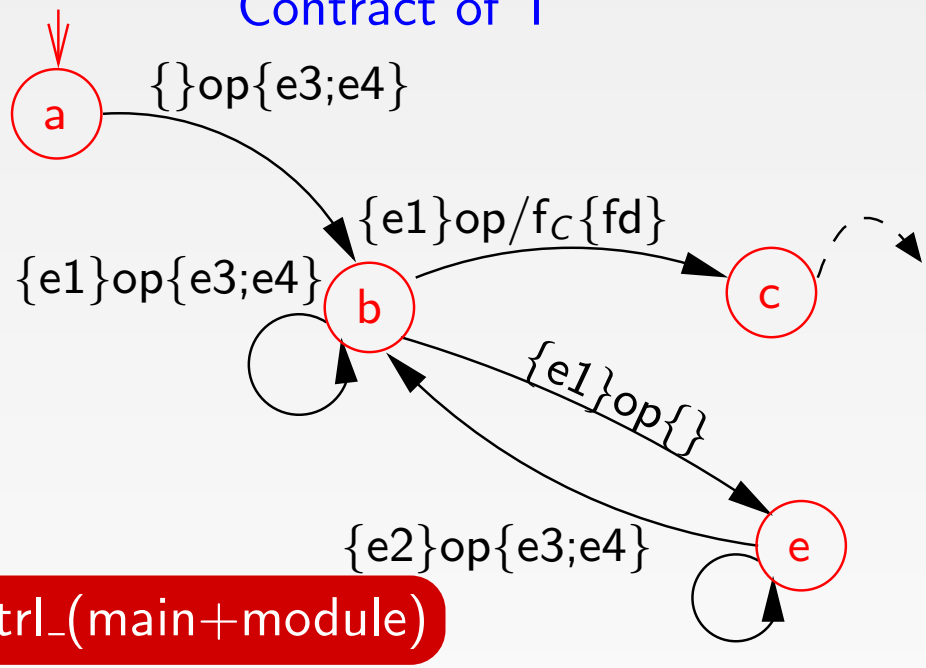
The goal is to capture all the control flow going out of the component (wait, notify, p.f, ...).

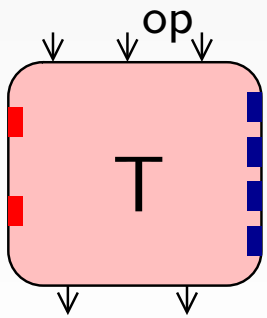
Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑwait(e1);
        if(x < 42){
              ⓒp.f(x);
              ⓓp.g(x);
        .....
```

Contract of T



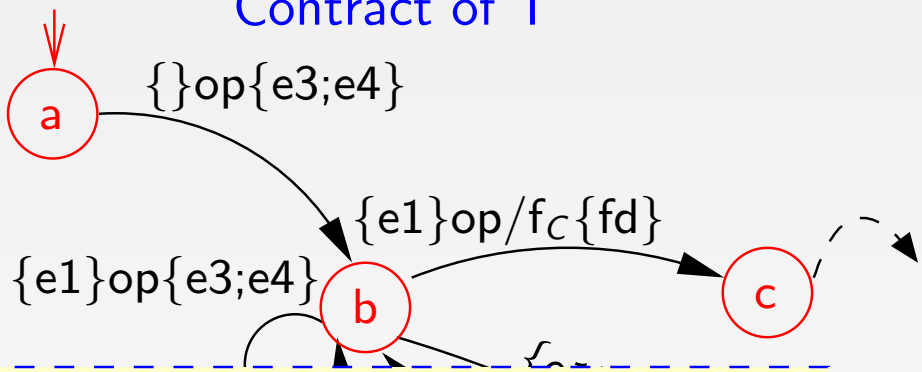


Executing the Implementation against the Contract

Component implementation:

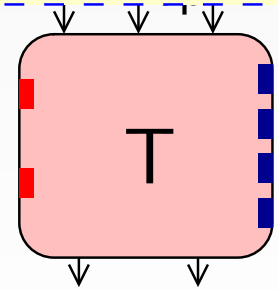
```
....
modulex::T(){
ⓐ while(true){
      x++;
      e3.notify();
      e4.notify();
      ⓑ wait(e1);
      if(x < 42){
```

Contract of T



The implementation executes in a thread. It is suspended and resumed by the 42 execution engine

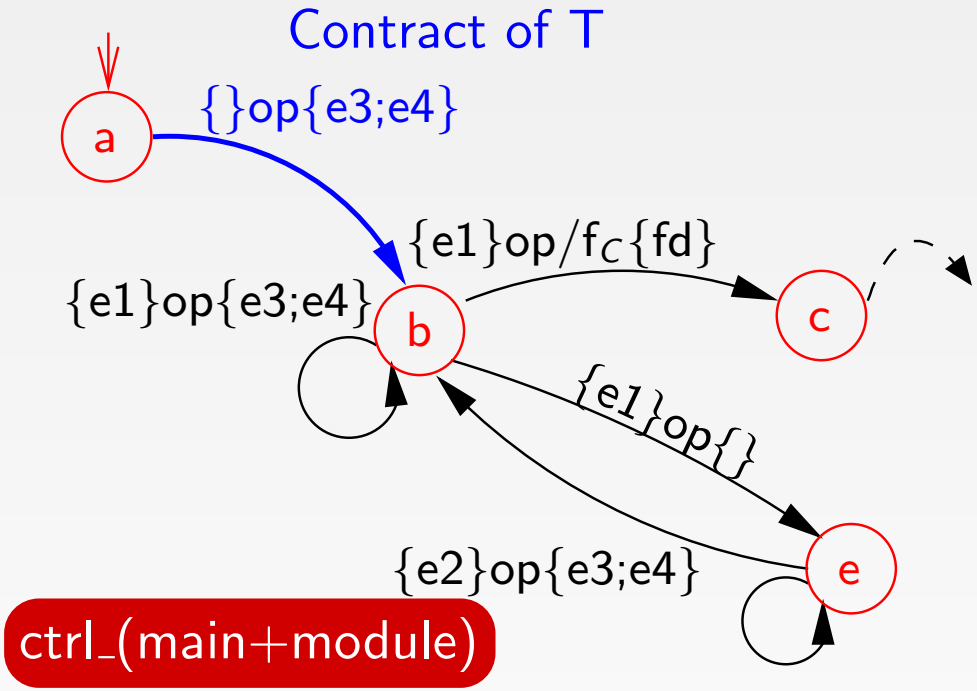
The contract of T is used as defensive code

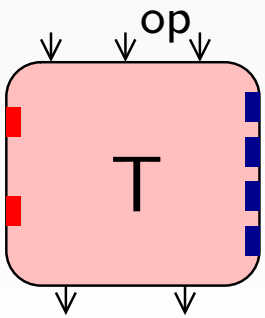


Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑwait(e1);
        if(x < 42){
              ⓒp.f(x);
              ⓓp.g(x);
        .....
```

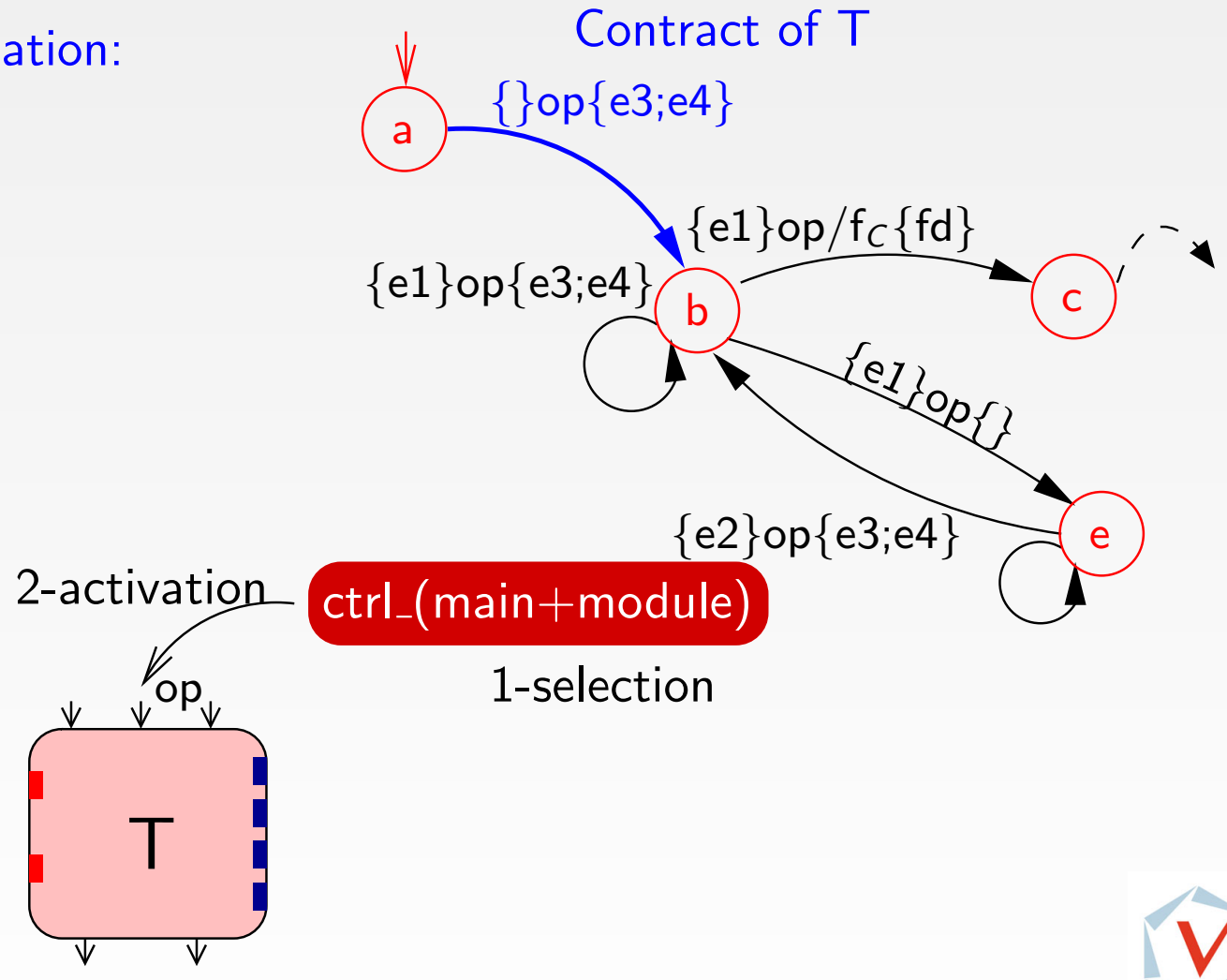




Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
        x++;
        e3.notify();
        e4.notify();
        ⓑwait(e1);
        if(x < 42){
              ⓒp.f(x);
              ⓓp.g(x);
        .....
```

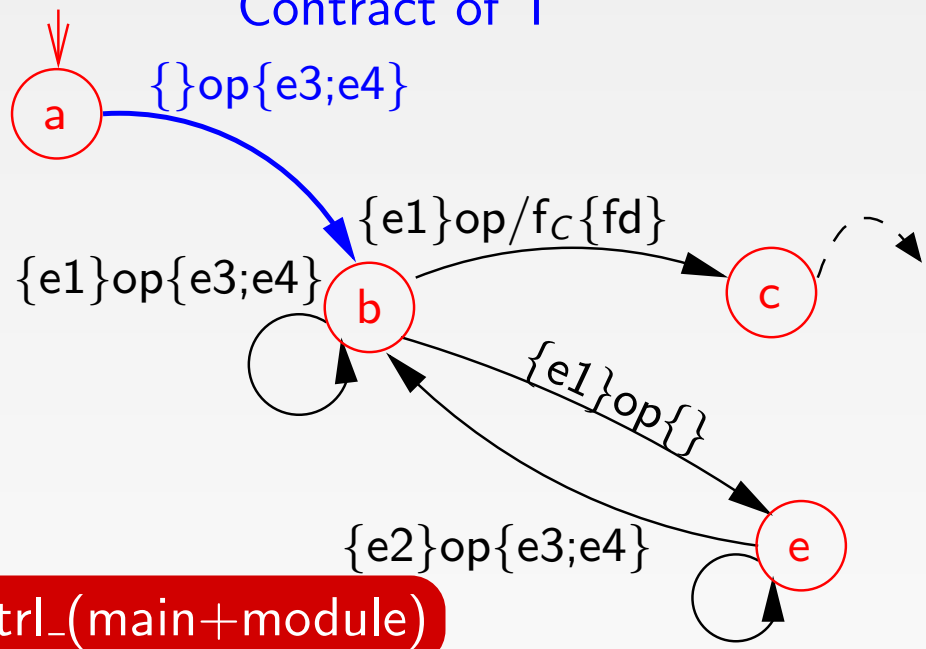


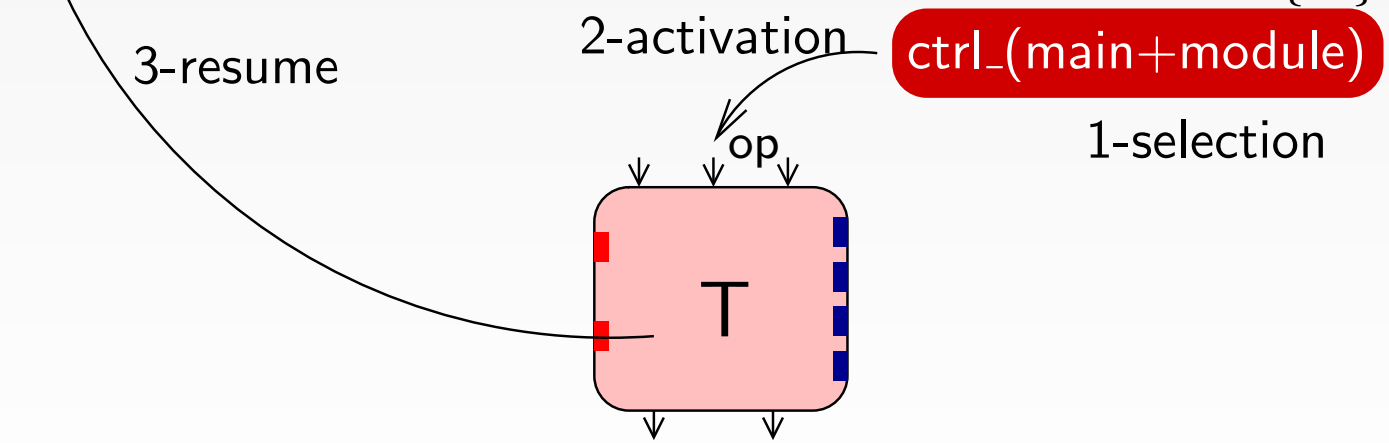
Executing the Implementation against the Contract

Component implementation:

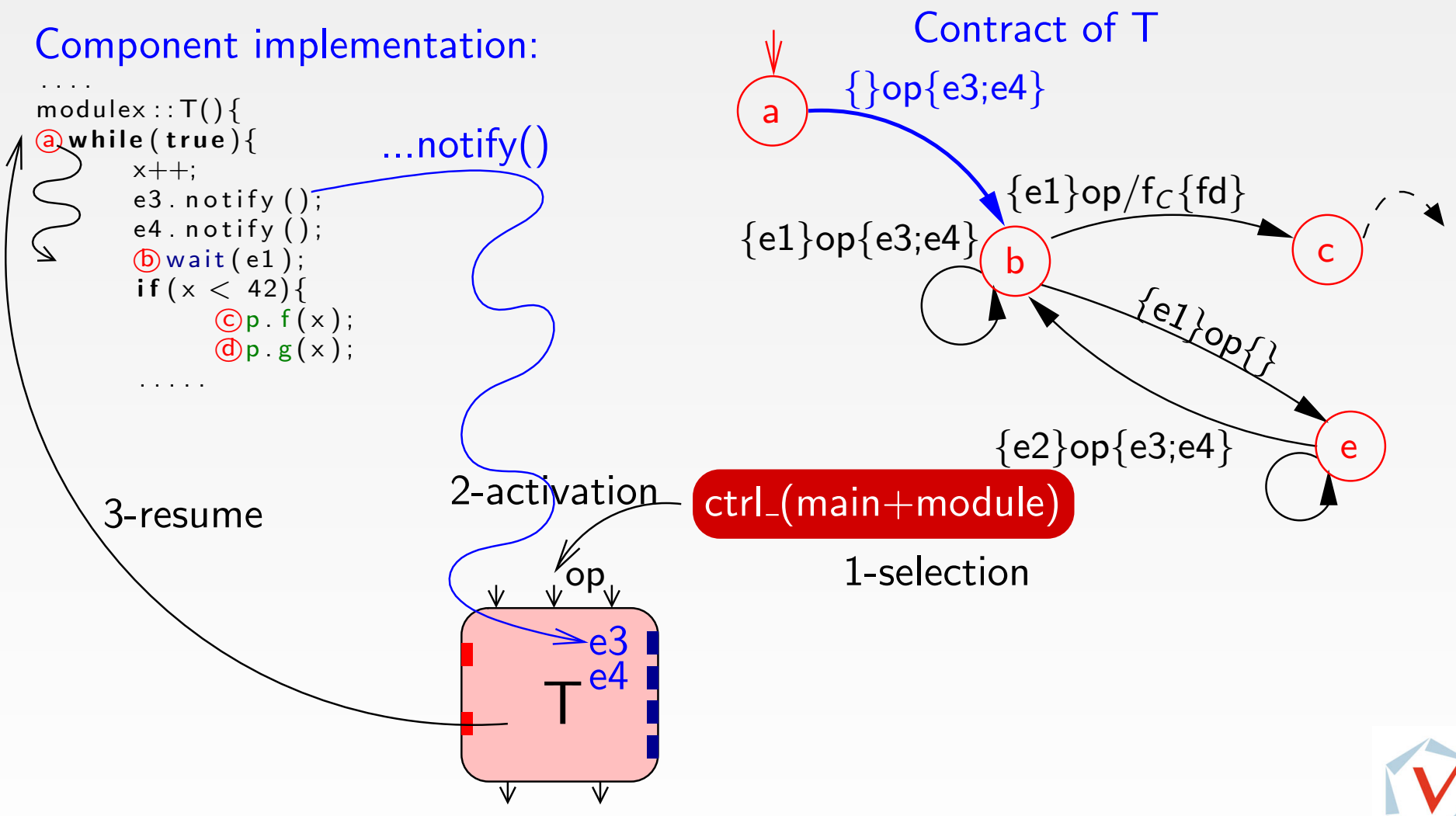
```
....
modulex::T(){
ⓐwhile(true){
      x++;
      e3.notify();
      e4.notify();
      ⓑwait(e1);
      if(x < 42){
            ⓒp.f(x);
            ⓓp.g(x);
      .....
```

Contract of T

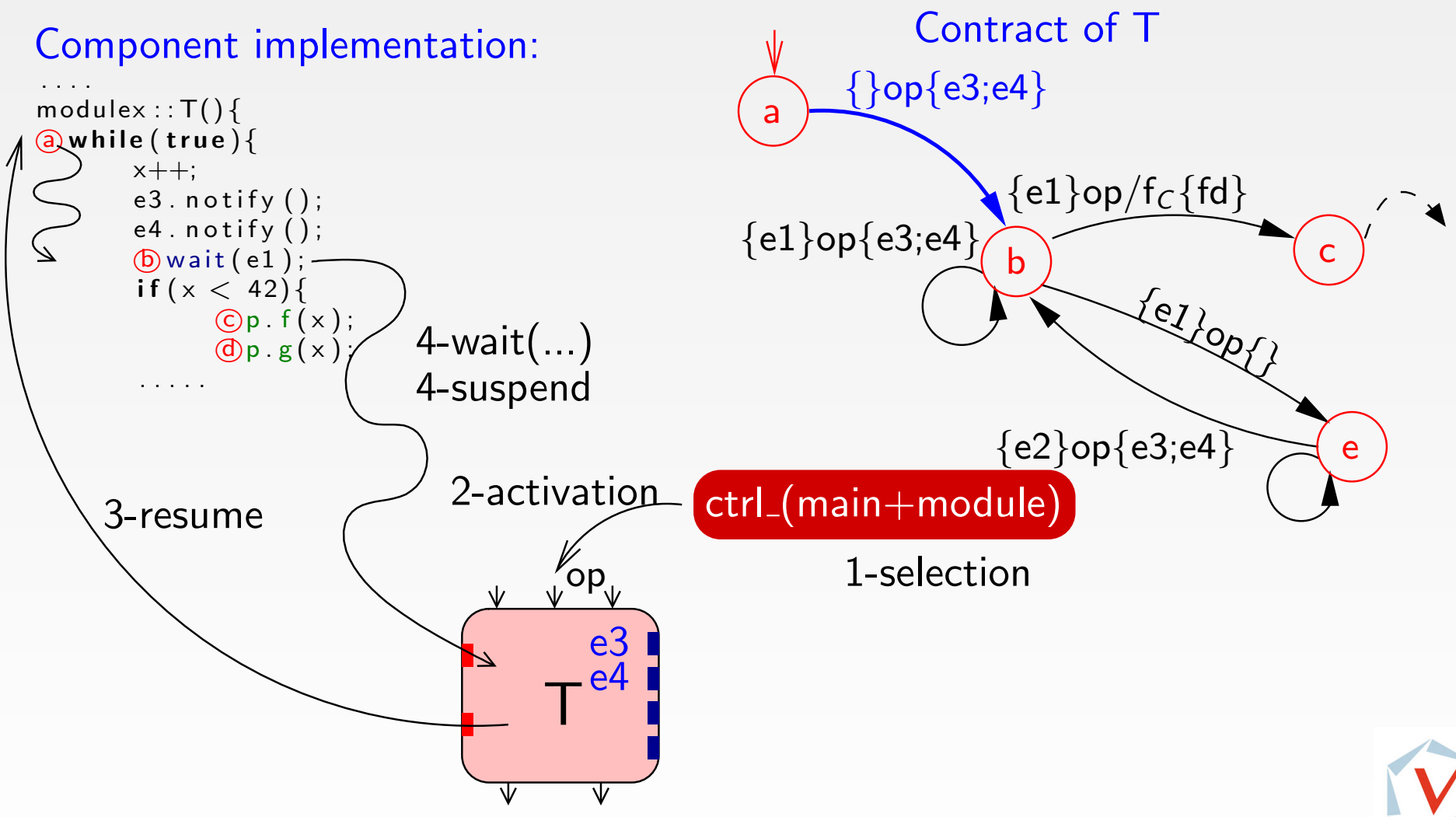




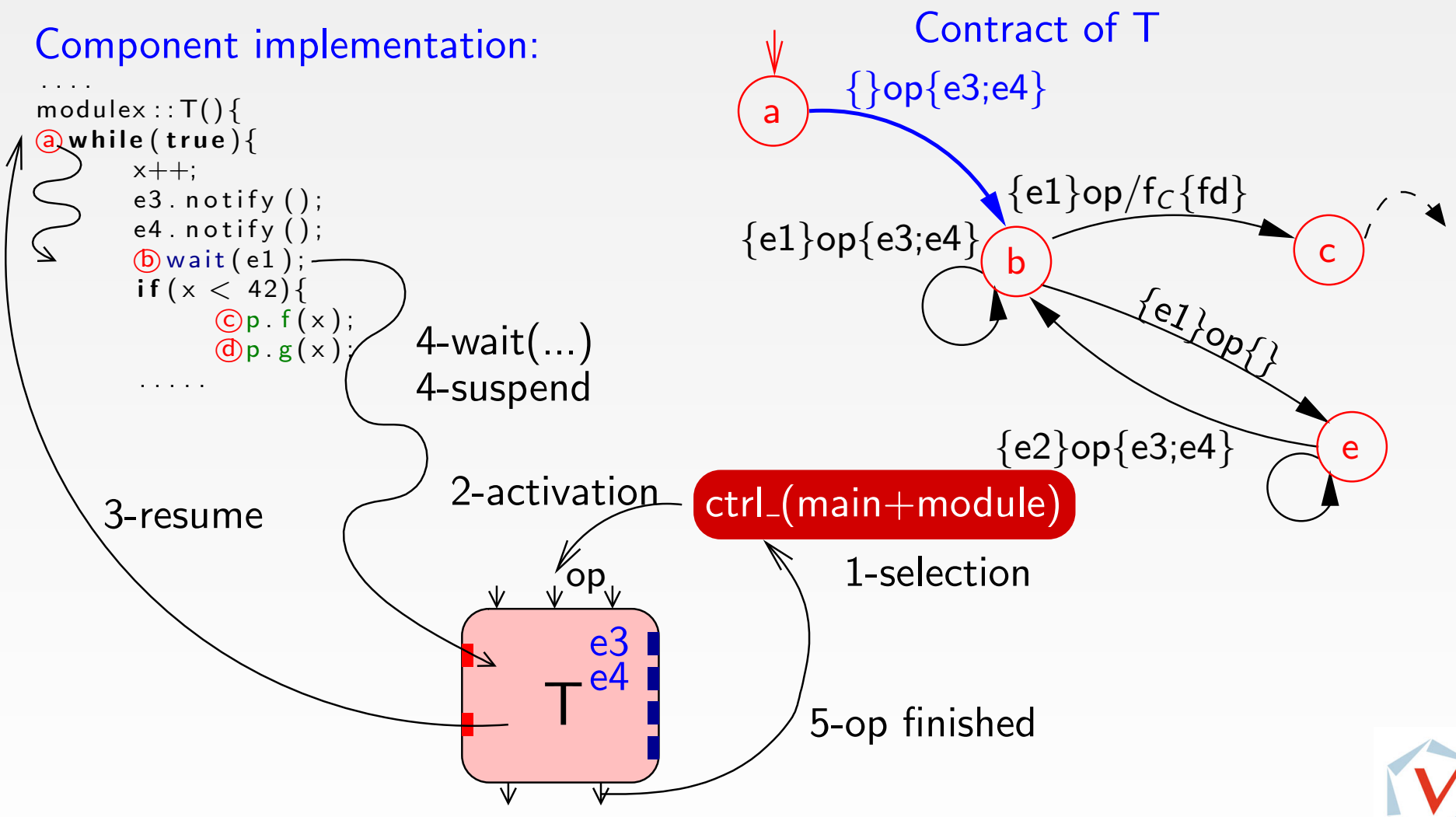
Executing the Implementation against the Contract



Executing the Implementation against the Contract



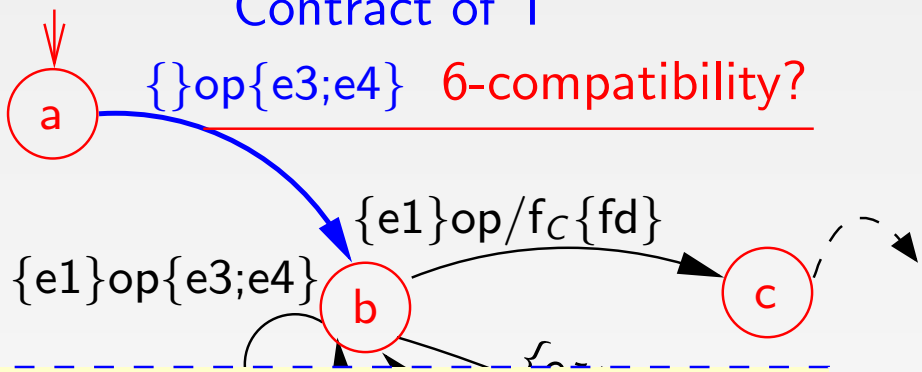
Executing the Implementation against the Contract



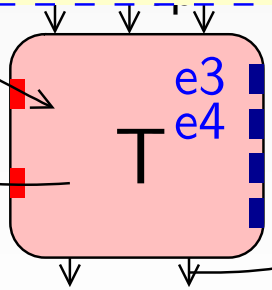
Executing the Implementation against the Contract

Component implementation:

```
....
modulex::T(){
ⓐwhile(true){
    x++;
    e3.notify();
    e4.notify();
    ⓑwait(e1);
    if(x < 42){
```



This execution step should notify *e3 and e4* (calls to *e.notify()*). Otherwise, the implementation does not respect the contract.



5-op finished

Contents

Summary and Perspectives

Summary:

- Formal Contracts For SystemC-TLM Components
- Executable, lightweight simulation $\Rightarrow$ early detection of bugs
- Joint execution of contracts and SystemC $\Rightarrow$ Check compatibility of implementation

Perspectives:

- Extend the translation to time
- Connection to verification tools
- Use 42 contracts to describe abstract SC-TLM components