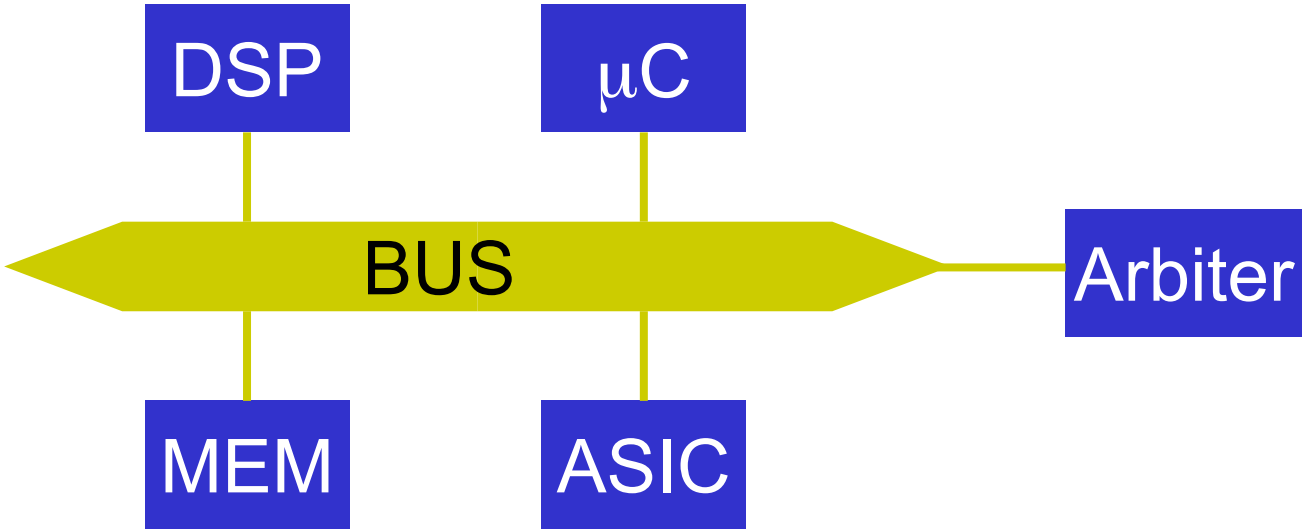
Transaction-level modeling of bus-based systems with SystemC 2.0

Ric Hilderink, Thorsten Grötker

Synopsys, Inc.

Efficient platform modeling

- Get to executable platform model ASAP
- Simulation speed >> 100k cycles/sec



Moving from pin-level to transaction-level models (TLM) is mandatory!

Outline

Idea:

- Based on an example show how SystemC 2.0 enables efficient platform modeling.
- Introduce some key language elements in the process.

Example: Simple bus model

- Cycle-accurate transaction-level model.
- "Simple" =
  - No pipelining
  - No split transactions
  - No master wait states
  - No request/acknowledge scheme
  - …

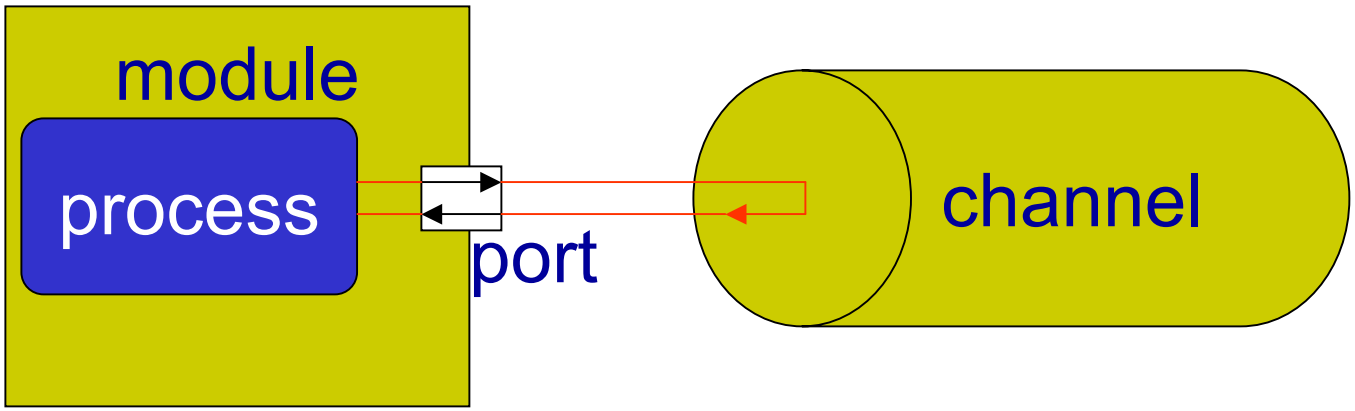
  NB: Of course, these features can be modeled at the transaction level

Interface Method Calls (IMC)

- Modules communicate via channels.
- Channels implement interfaces.
- An interface is a set of methods (functions).
- Ports
  - Modules have ports.
  - Ports are connected to channels.
  - Modules access channels through ports.

```
...
some_port->some_method(some_data);
...
```

Interface Methods Calls (cont'd)

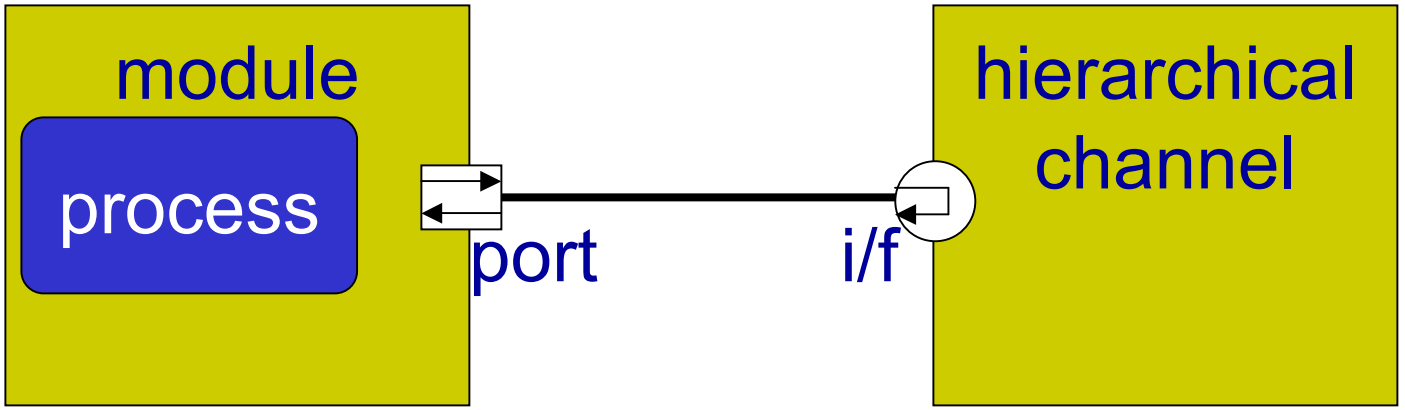



```
module::process() {
      ...
      port->some_method(42);
      ...
}
```

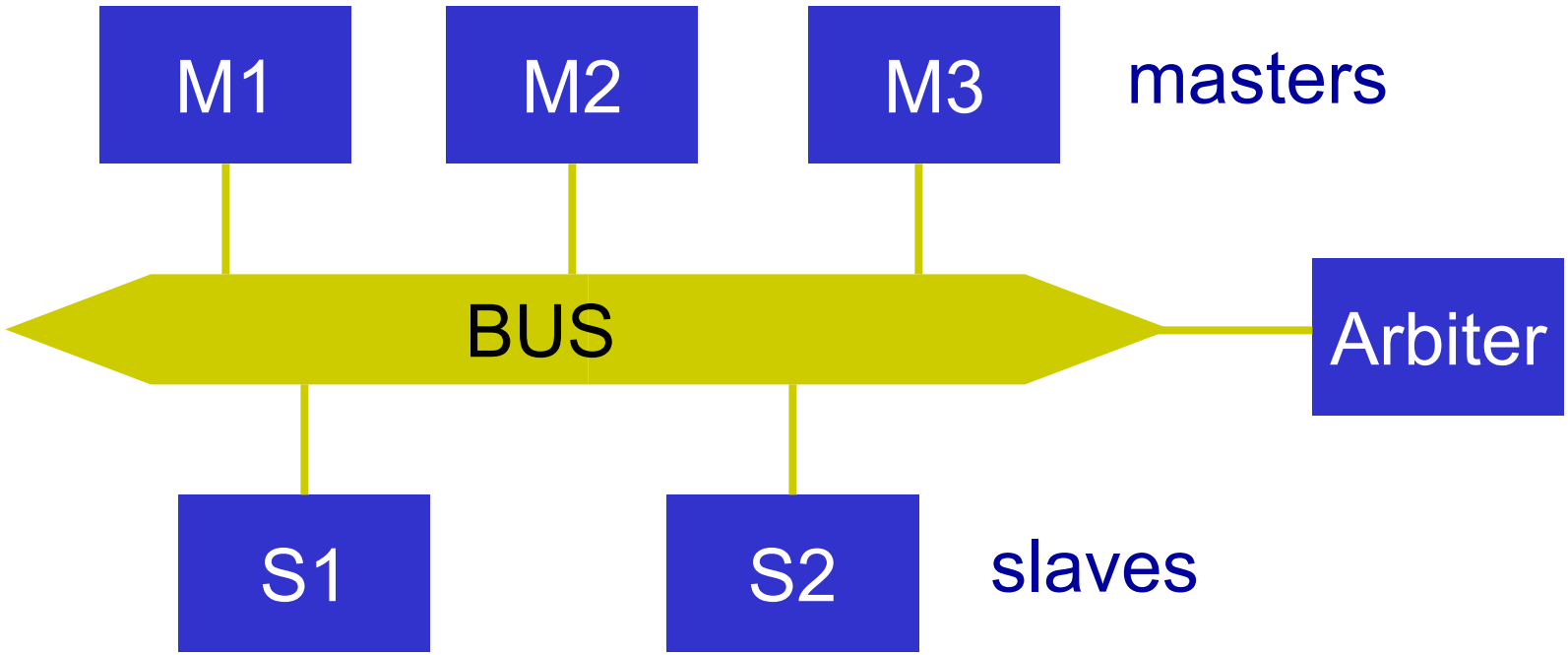

SYSTEMC™

Hierarchical channels

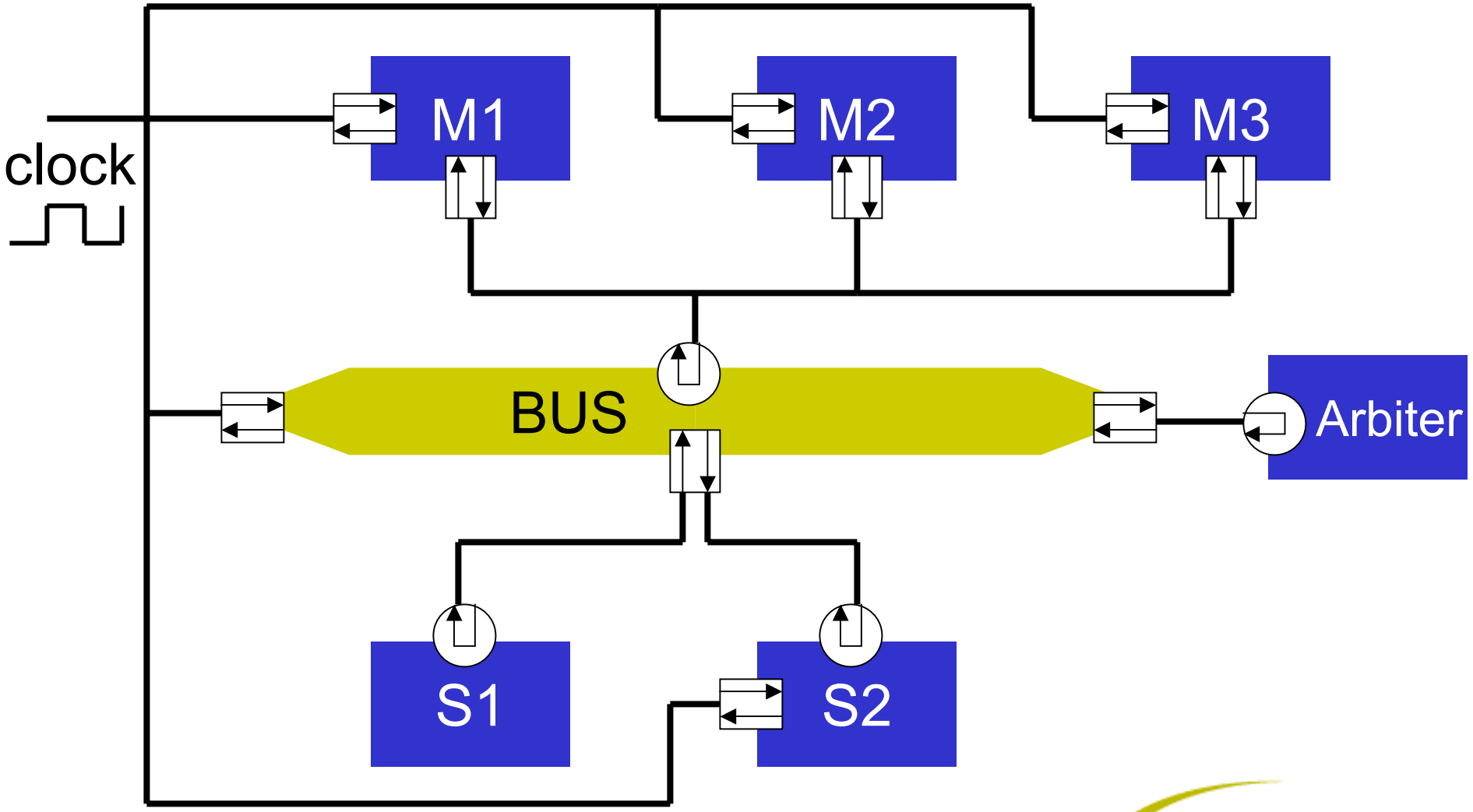
- Channels can be hierarchical, i.e. they can contain modules, processes, and channels.
- A module that implements an interface is a hierarchical channel.



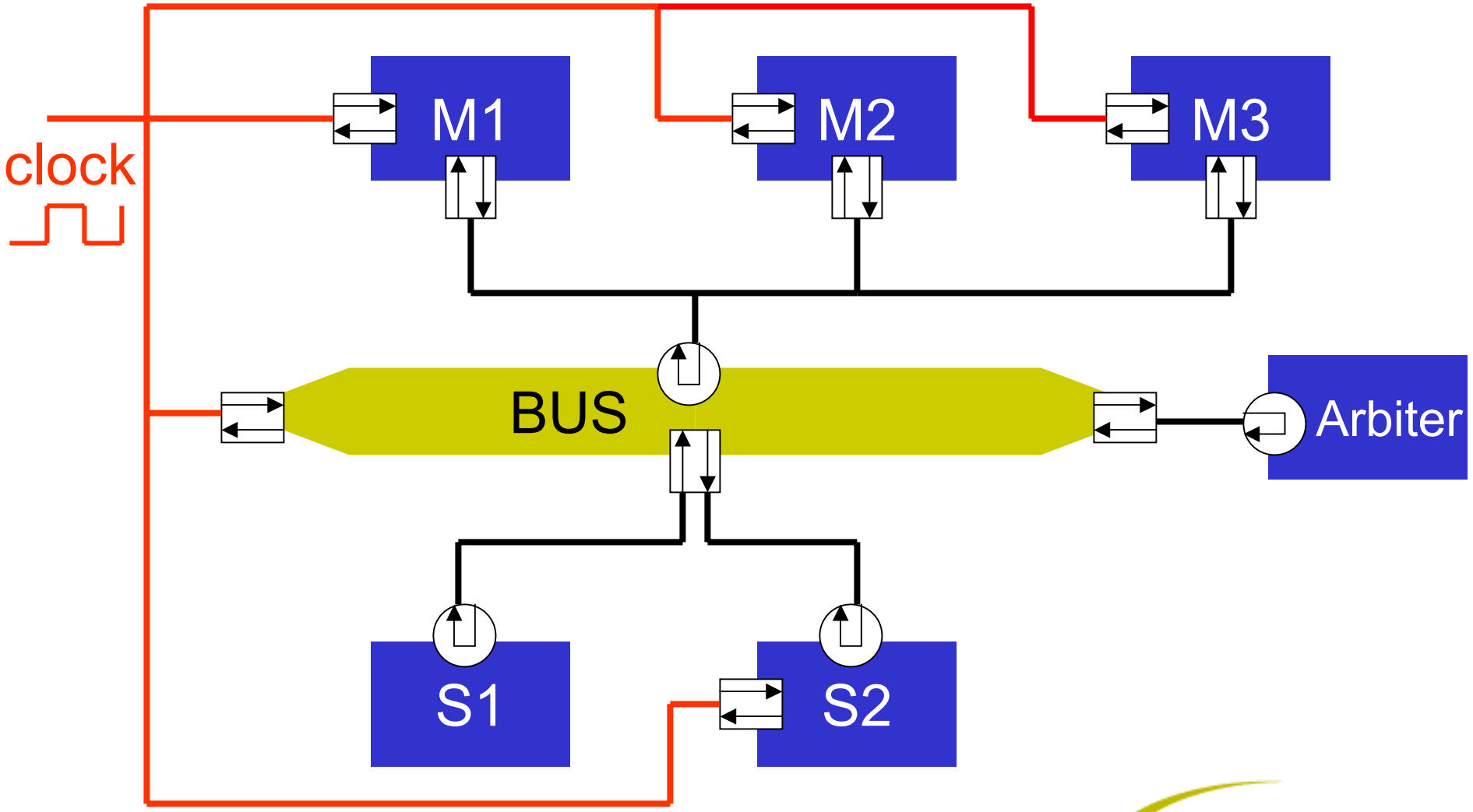
Example system (napkin view)



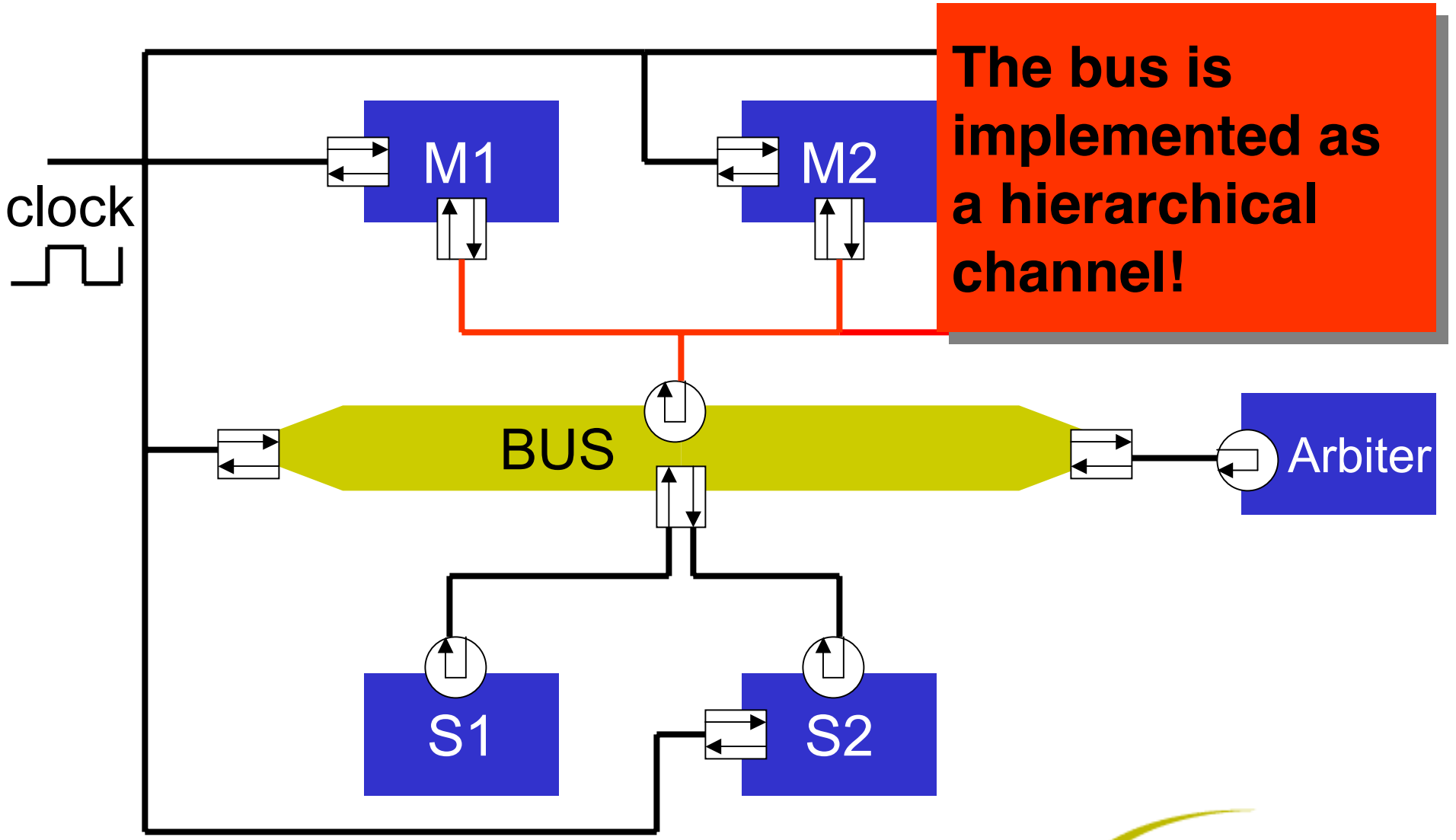
SystemC 2.0 transaction-level model



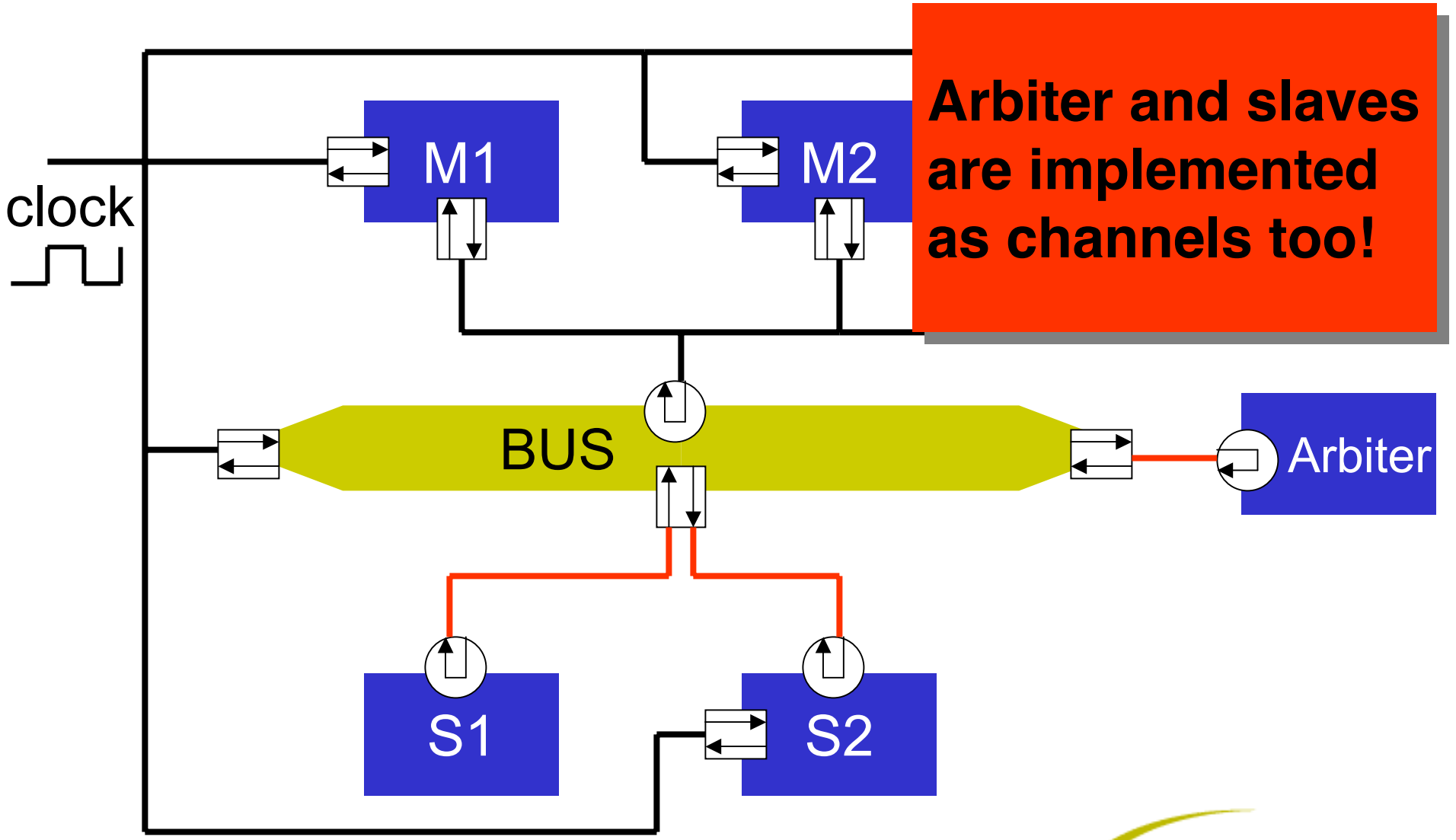
SystemC 2.0 transaction-level model



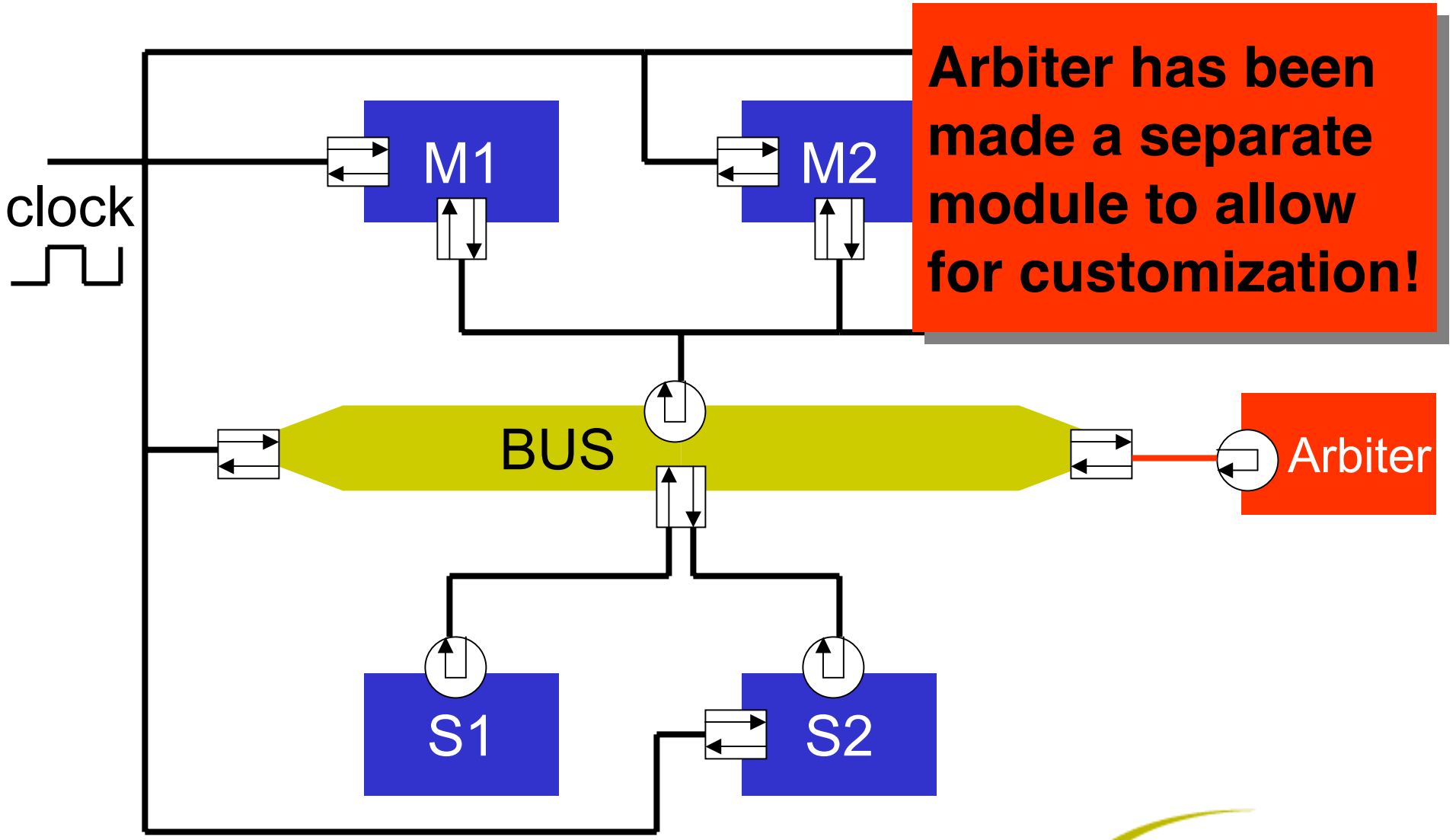
SystemC 2.0 transaction-level model



SystemC 2.0 transaction-level model

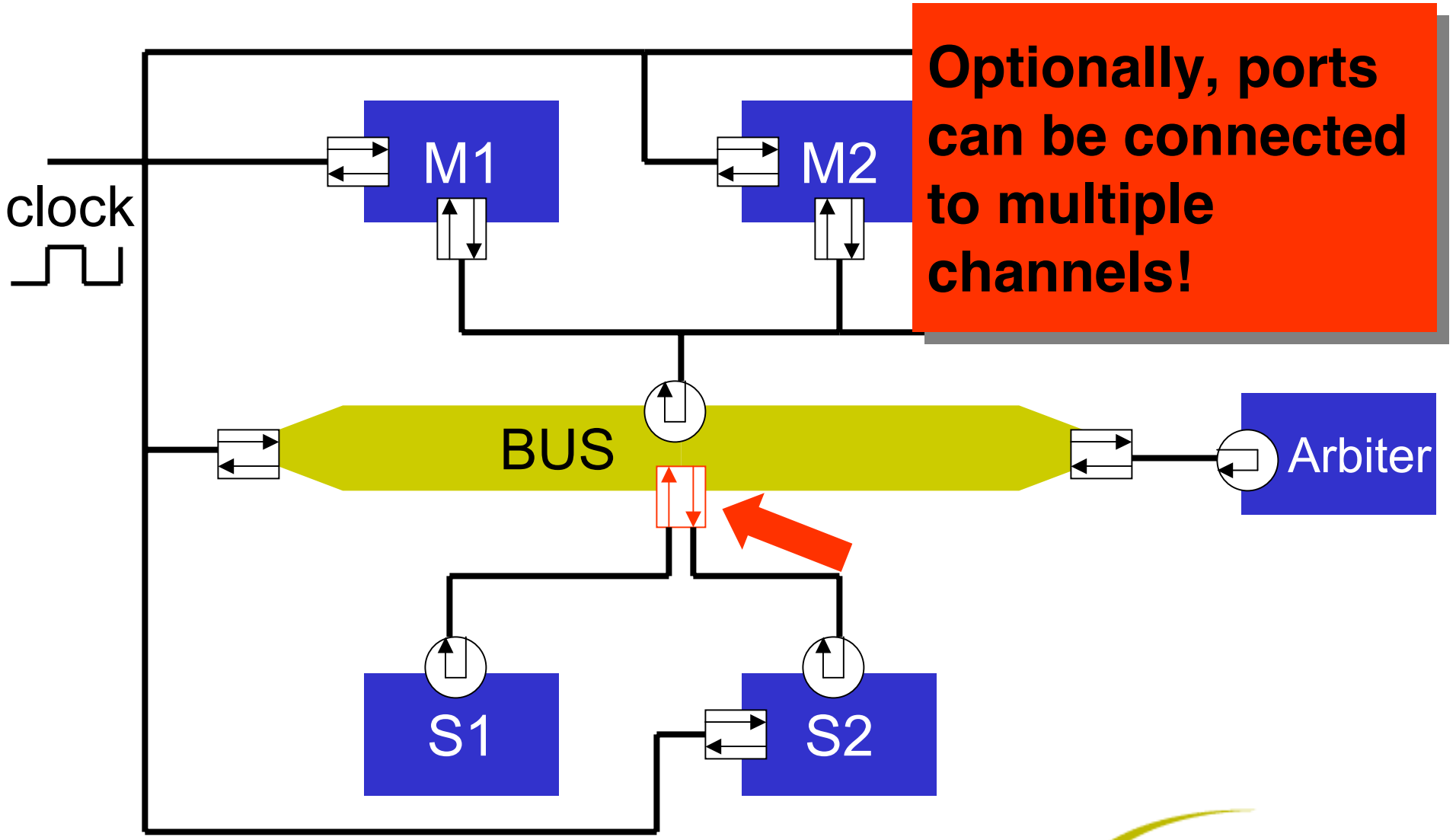


SystemC 2.0 transaction-level model

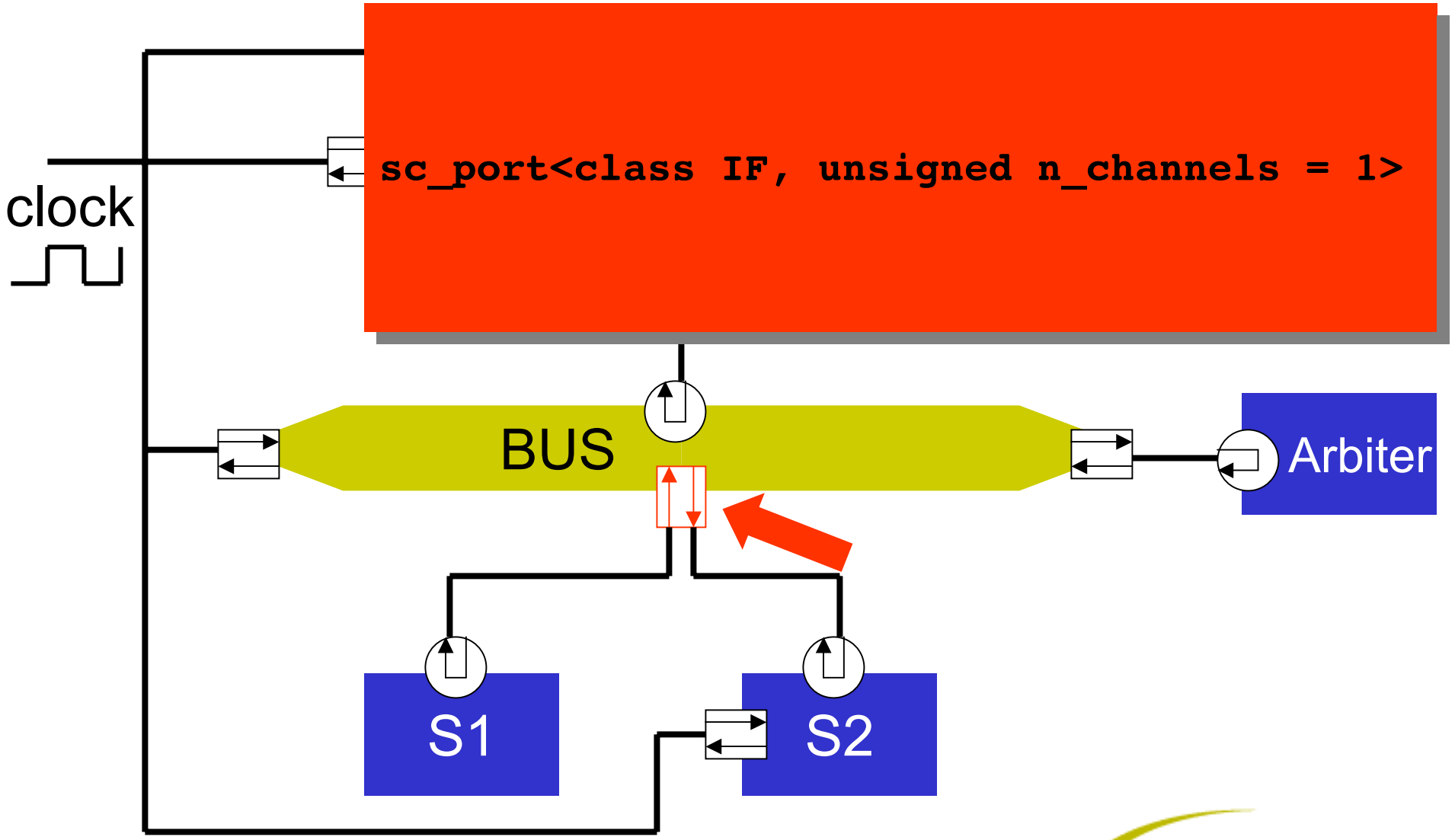


Arbiter has been made a separate module to allow for customization!

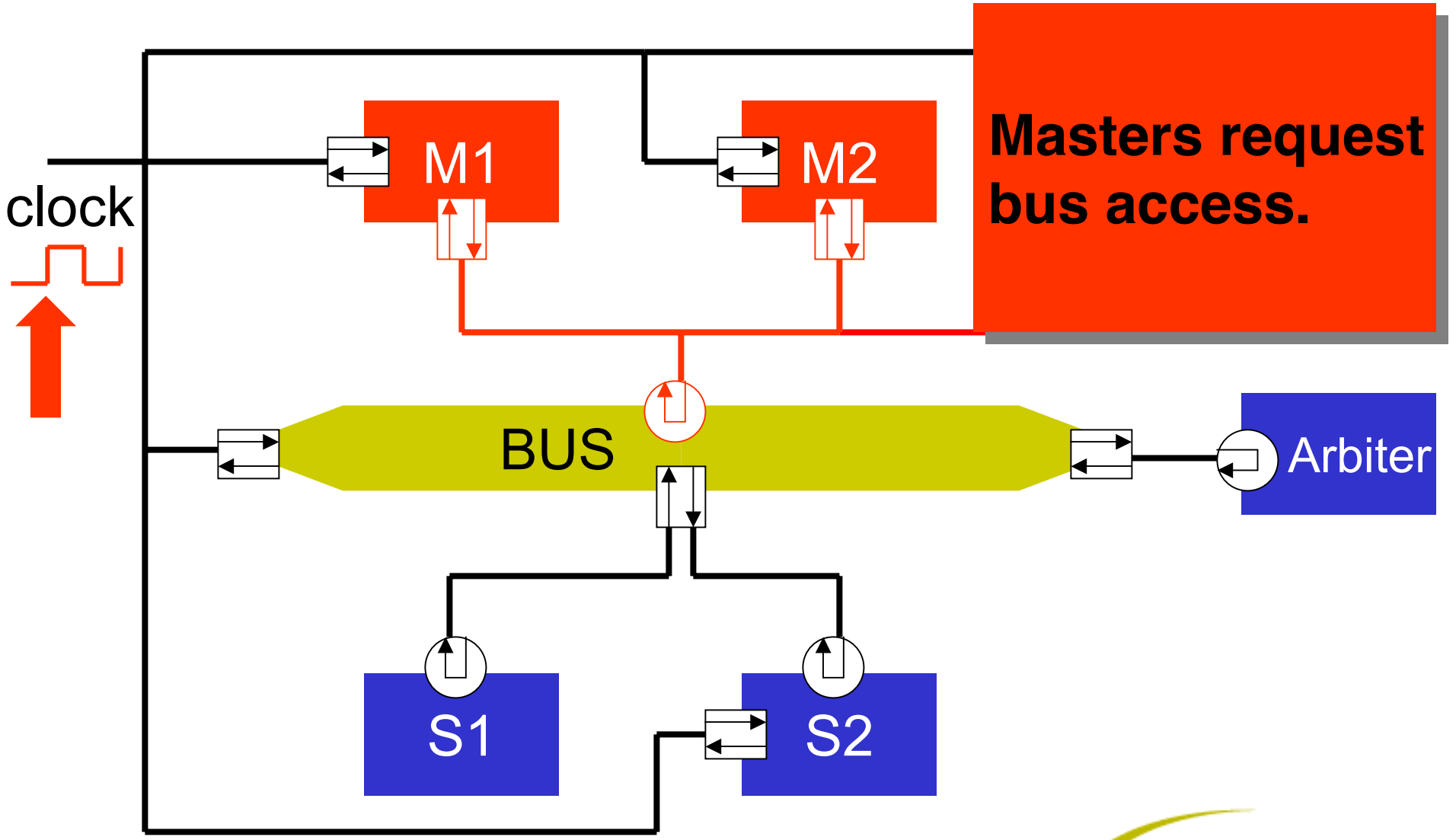
SystemC 2.0 transaction-level model



SystemC 2.0 transaction-level model

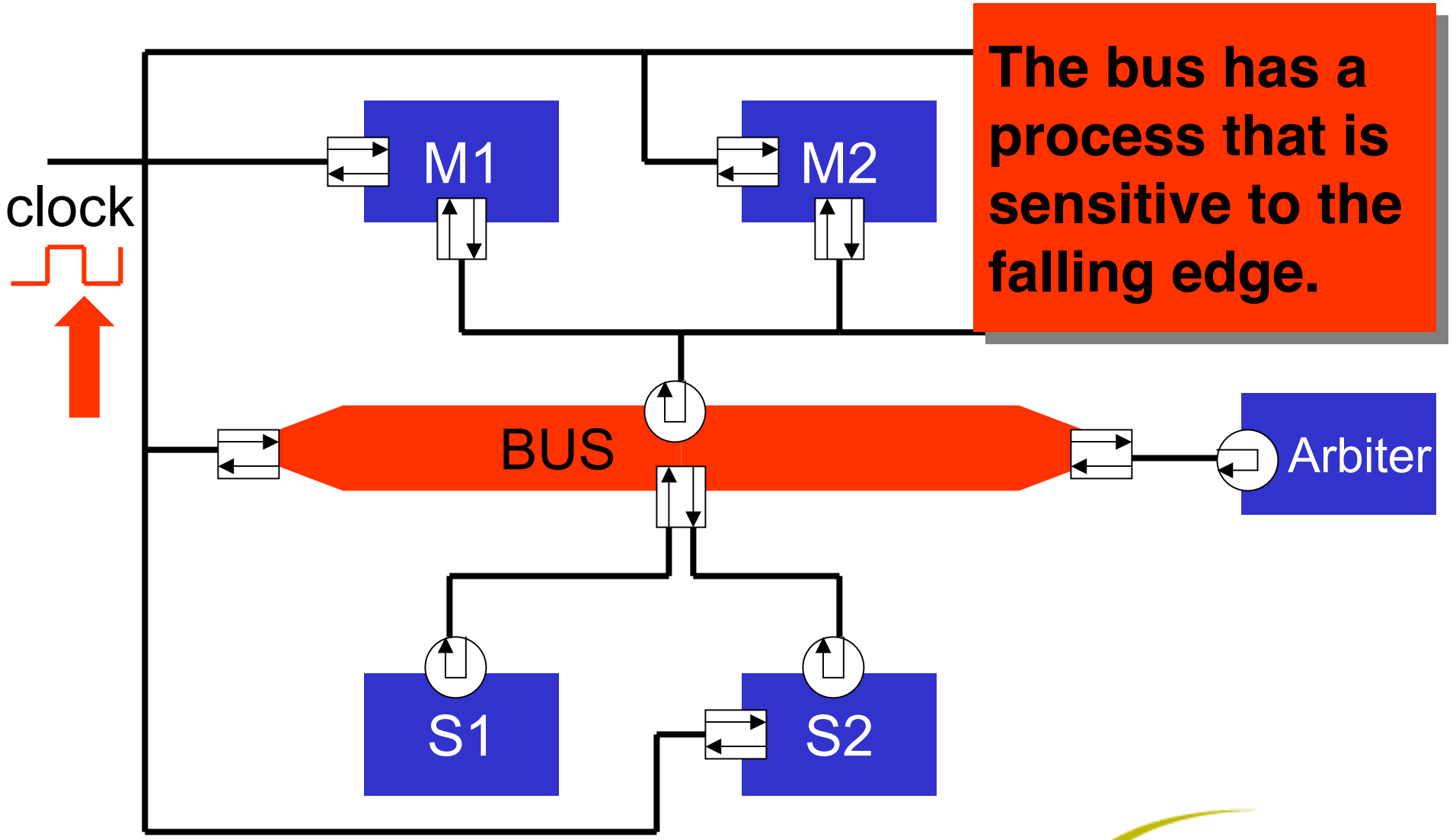


Rising clock edge



SYSTEMC™

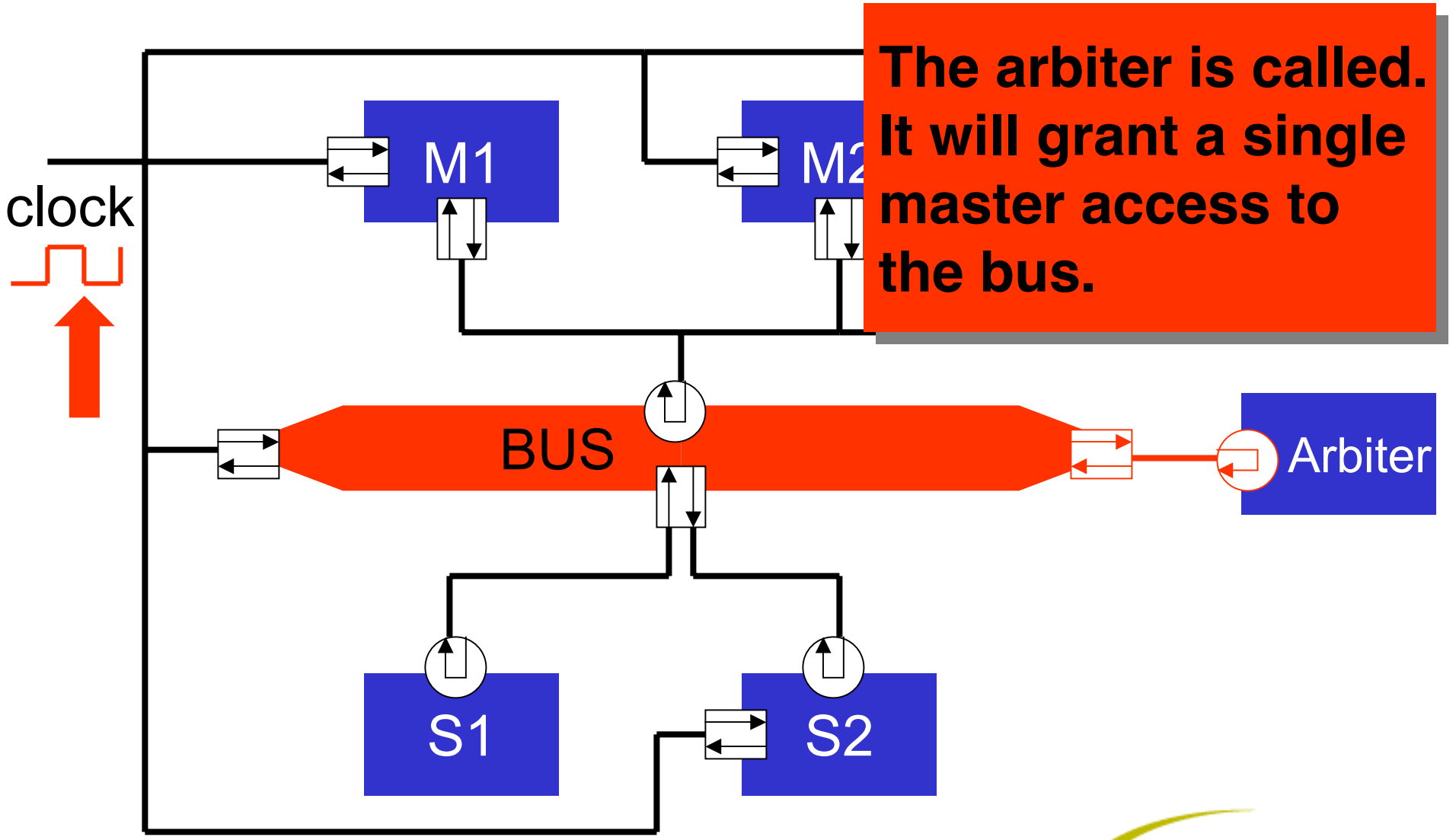
Falling clock edge



The bus has a process that is sensitive to the falling edge.

SYSTEMC™

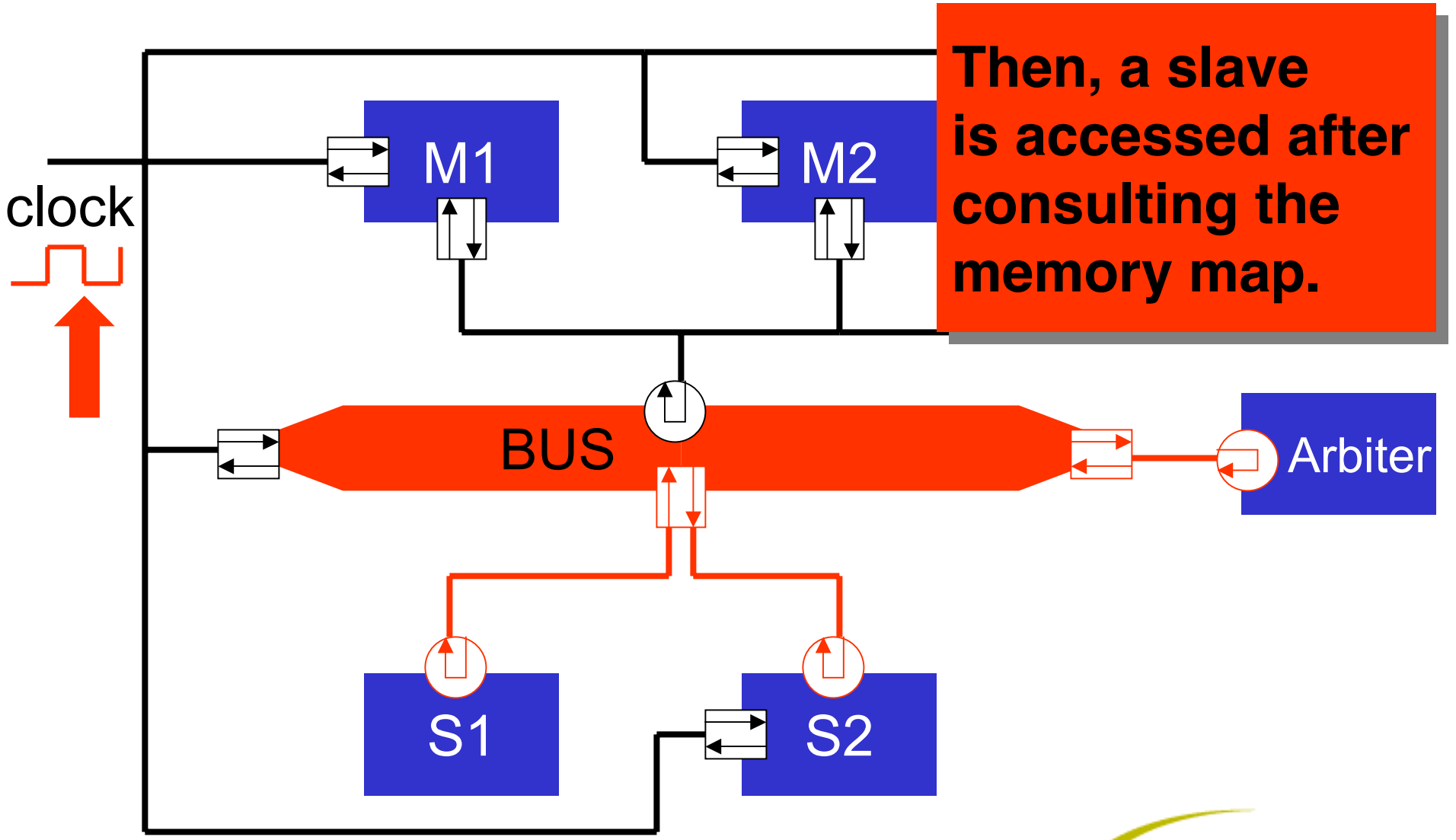
Falling clock edge



The arbiter is called. It will grant a single master access to the bus.

SYSTEMC™

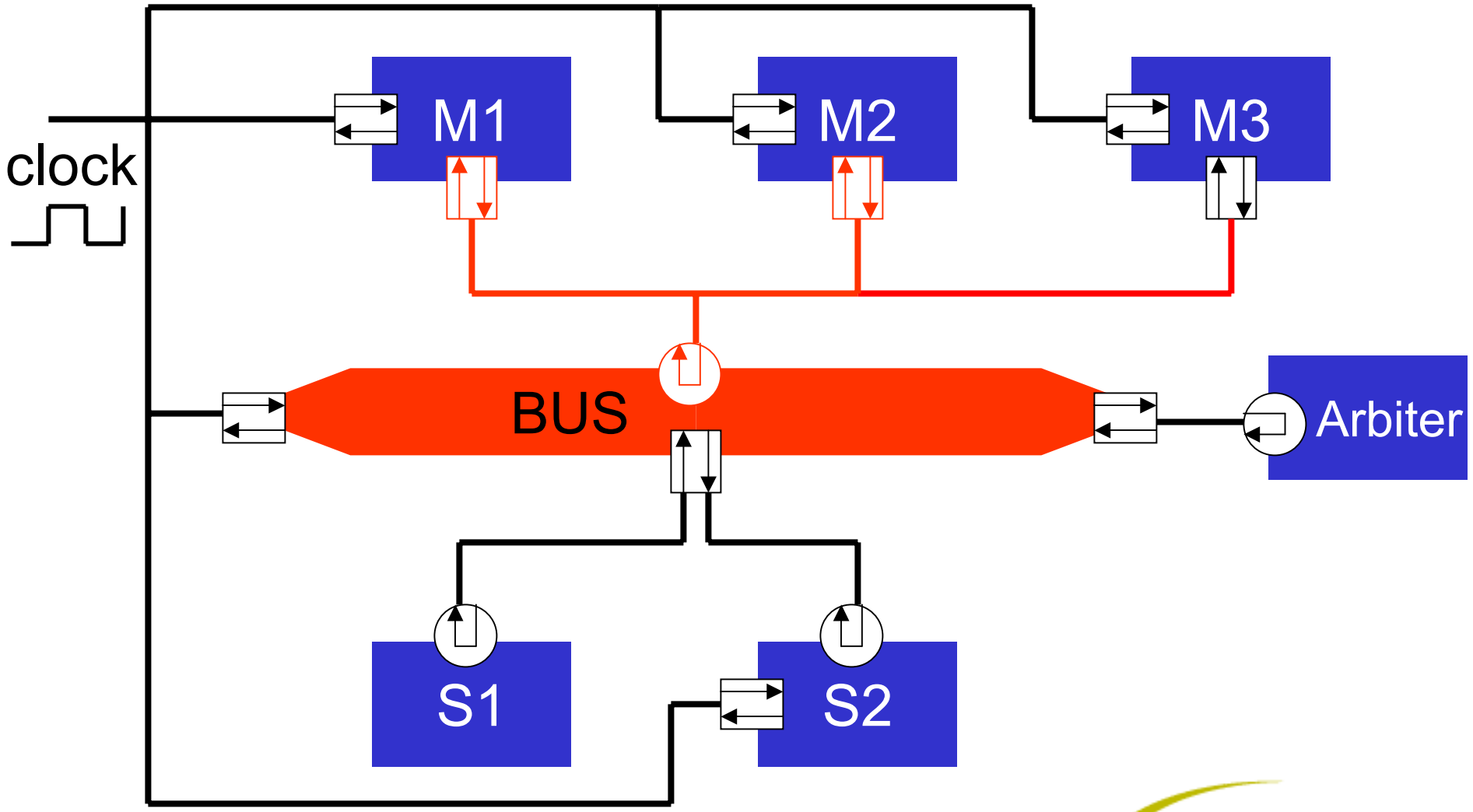
Falling clock edge



Then, a slave is accessed after consulting the memory map.

SYSTEMC™

Bus interfaces



Master interfaces of the bus

- Blocking:
  - Complete bursts
  - Used by high-level models
- Non-blocking:
  - Cycle-based
  - Used by processor models
- Direct:
  - Immediate slave access
  - Put SW debugger to work

Blocking master interface

- ```
  status burst_read(unique_priority, data*,
                    start_address, length=1,
                    lock=false);
  ```
- ```
  status burst_write(unique_priority, data*,
                     start_address, length=1,
                     lock=false);
  ```
- "Blocking" because call returns only after complete transmission is finished.
- Master is identified by its unique priority.

Dynamic Sensitivity

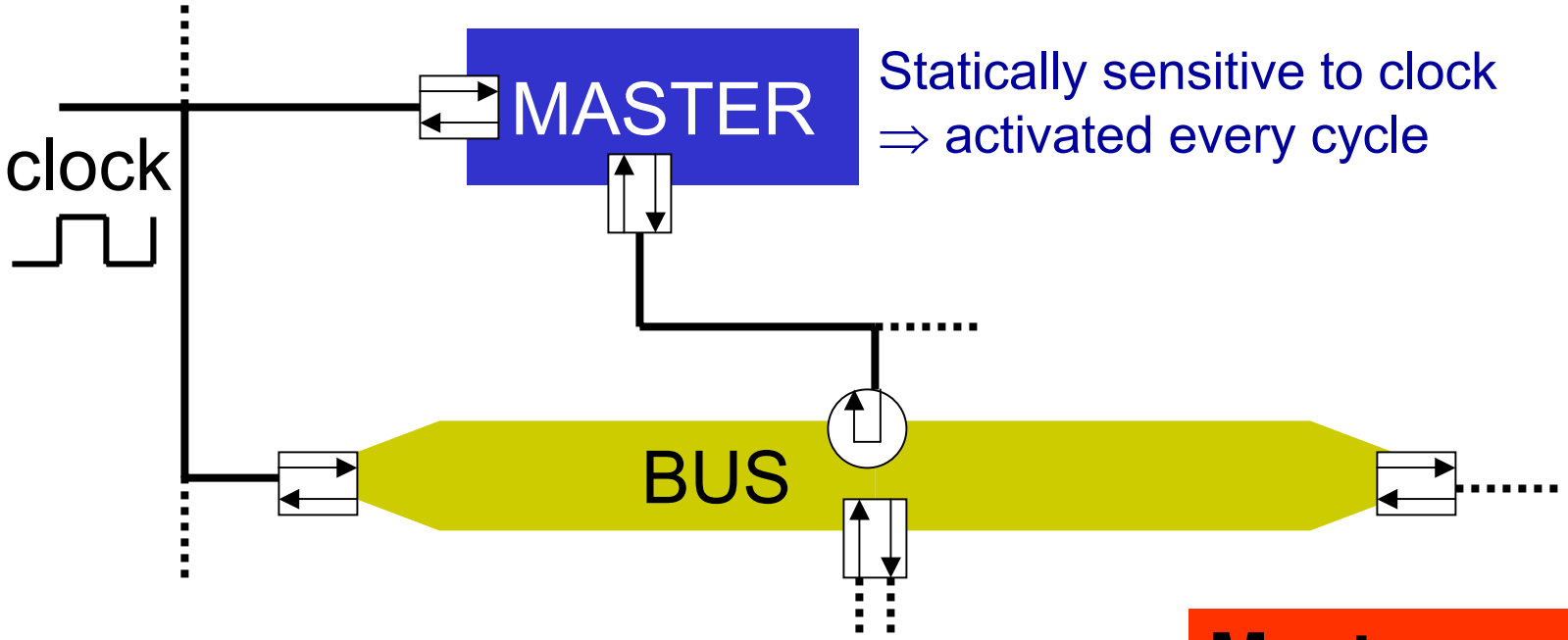
- SystemC 1.0
  - Static sensitivity
    - Processes are made sensitive to a fixed set of signals during elaboration
- SystemC 2.0
  - Static sensitivity
  - Dynamic sensitivity
    - The sensitivity (activiation condition) of a process can be altered during simulation (after elaboration)
    - Main features: events and extended wait() method

Waiting

```
wait();                    // as in SystemC 1.0
wait(event);               // wait for event
wait(e1 | e2 | e3); // wait for first event
wait(e1 & e2 & e3); // wait for all events
wait(200, SC_NS);          // wait for 200ns

// wait with timeout
wait(200, SC_NS, e1 | e2);
wait(200, SC_NS, e1 & e2);
```

Dynamic sensitivity



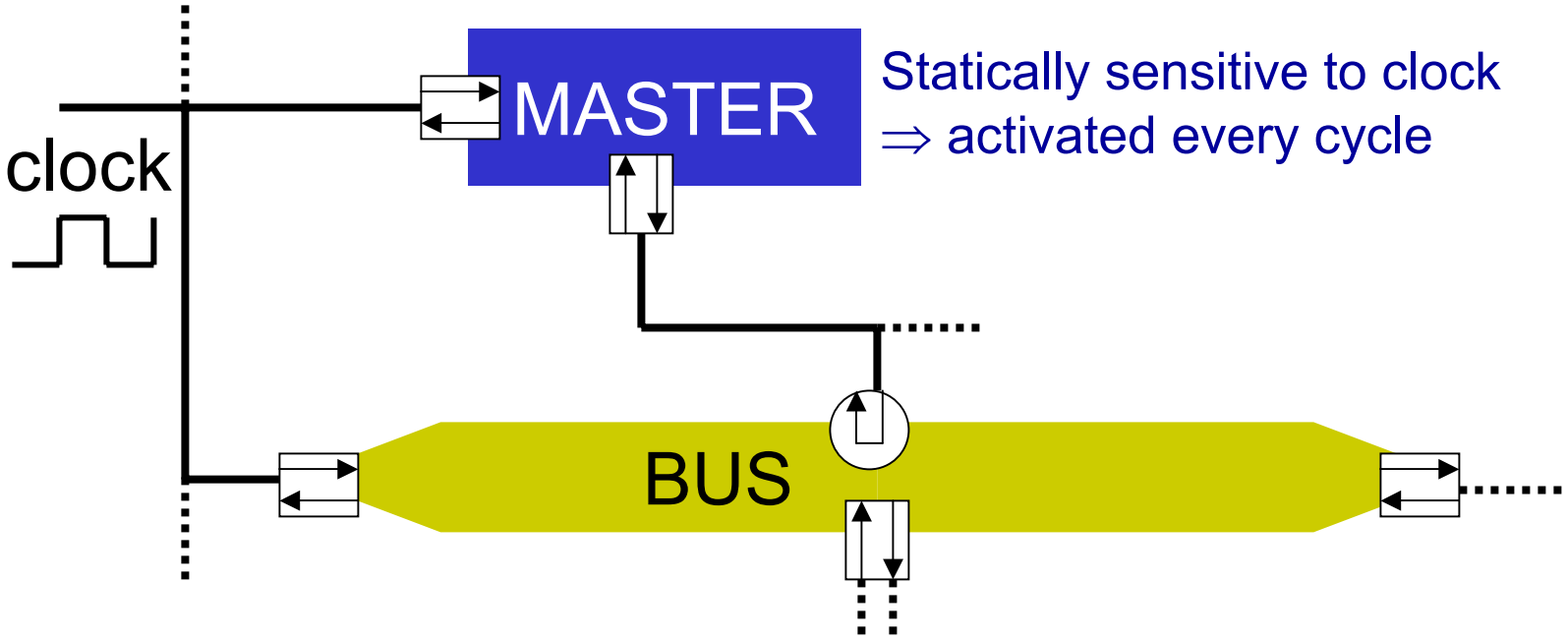
Statically sensitive to clock
⇒ activated every cycle

```
status bus::burst_write(...) {
      ...
      wait(transmission_done);
      ...
}
```

Master won't be activated until transmission is completed!

SYSTEMC™

Dynamic sensitivity



Advantages:

- Easy-to-use interface (blocking interface)
- Simulation speed

SYSTEMC™

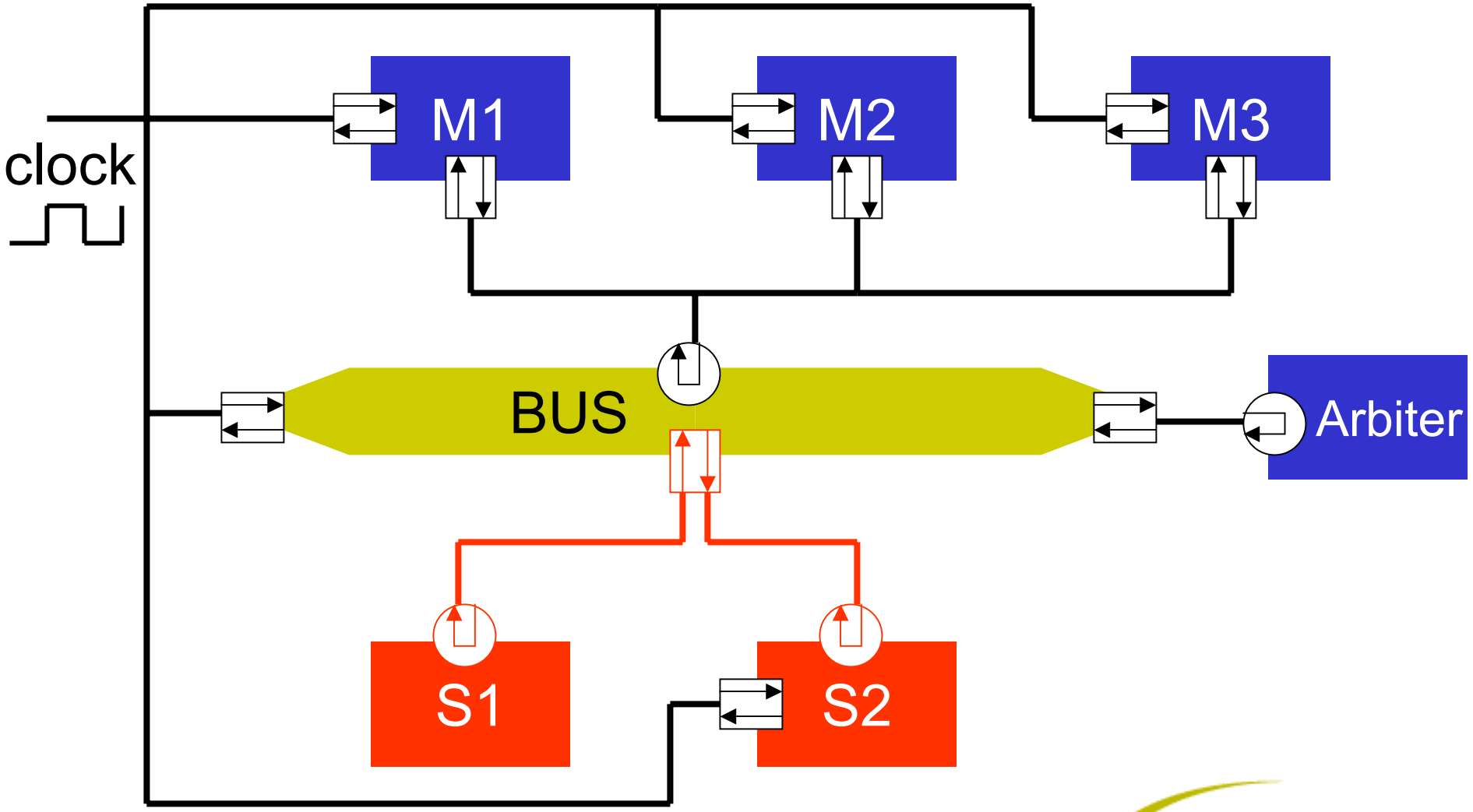
Non-blocking master interface

- `status get_status(unique_priority);`
- `status read(unique_priority, data*, address, lock=false);`
- `status write(unique_priority, data*, address, lock=false);`
- "Non-blocking" because calls return immediately.
- Less convenient than blocking API but caller remains in control (needed e.g. for most processor models).

Direct master interface

- `status direct_read(data*, address);`
- `status direct_write(data*, address);`
- Provides direct access to slaves (using the bus' address map).
  - Immediate access ⇒ simulated time does not advance
  - No arbitration
- Use for SW debuggers or decoupling of HW and SW.
- Use with care!

Slave interface



Slave interfaces

- `unsigned start_address();`
- `unsigned end_address();`

address mapping

- `status read(data*, address);`
- `status write(data*, address);`

regular I/O

- `status direct_read(data*, address);`
- `status direct_write(data*, address);`

debug interface

What's so cool about transaction-level bus models?

They are …

- relatively easy to develop and extend
- easy to use
- fast
  - use of IMC $\Rightarrow$ function calls instead of HW signals and control FSMs
  - use of dynamic sensitivity $\Rightarrow$ reduce unnecessary process activations

Key language elements used in the example

- Interface method calls (IMC)
- Hierarchical channels
- Connecting ports to multiple channels
- Dynamic sensitivity / waiting

Conclusions

SystemC 2.0 enables efficient platform modeling.

- **Ease of modeling**
  $\Rightarrow$ get to executable platform model ASAP
- **Simulation speed**

Still not convinced?

Try it out! **(see following slides)**

How to install

> cd <systemc_installation_directory>/examples/systemc

> gtar zxvf simple_bus_v2.tgz

This will create a directory 'simple_bus'. Go to this directory and build the executable, e.g.

For gcc-2.95.2 on Solaris:

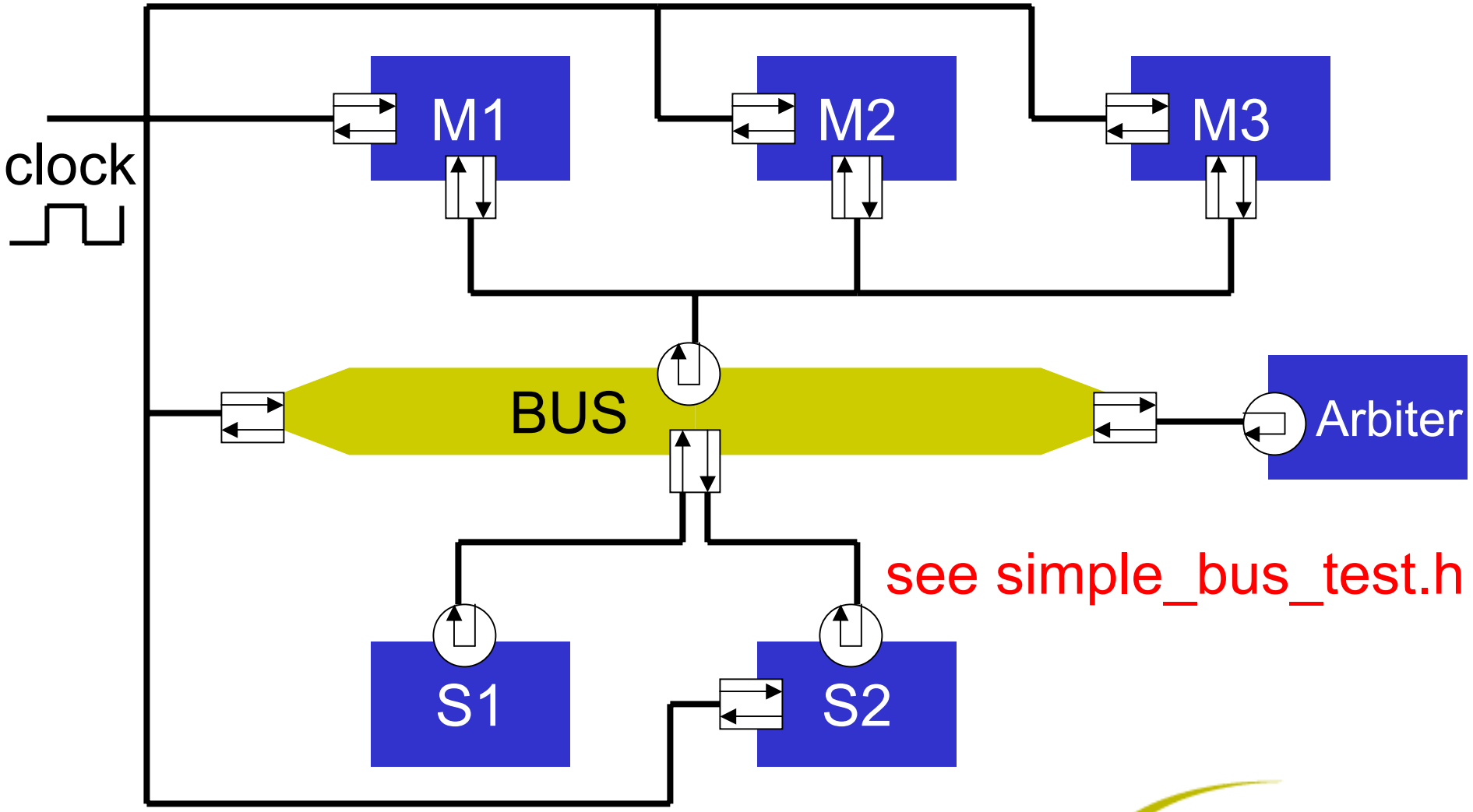
> gmake -f Makefile.gcc

See README.txt for detailed information!

Now you can run the executable, e.g.

> simple_bus.x

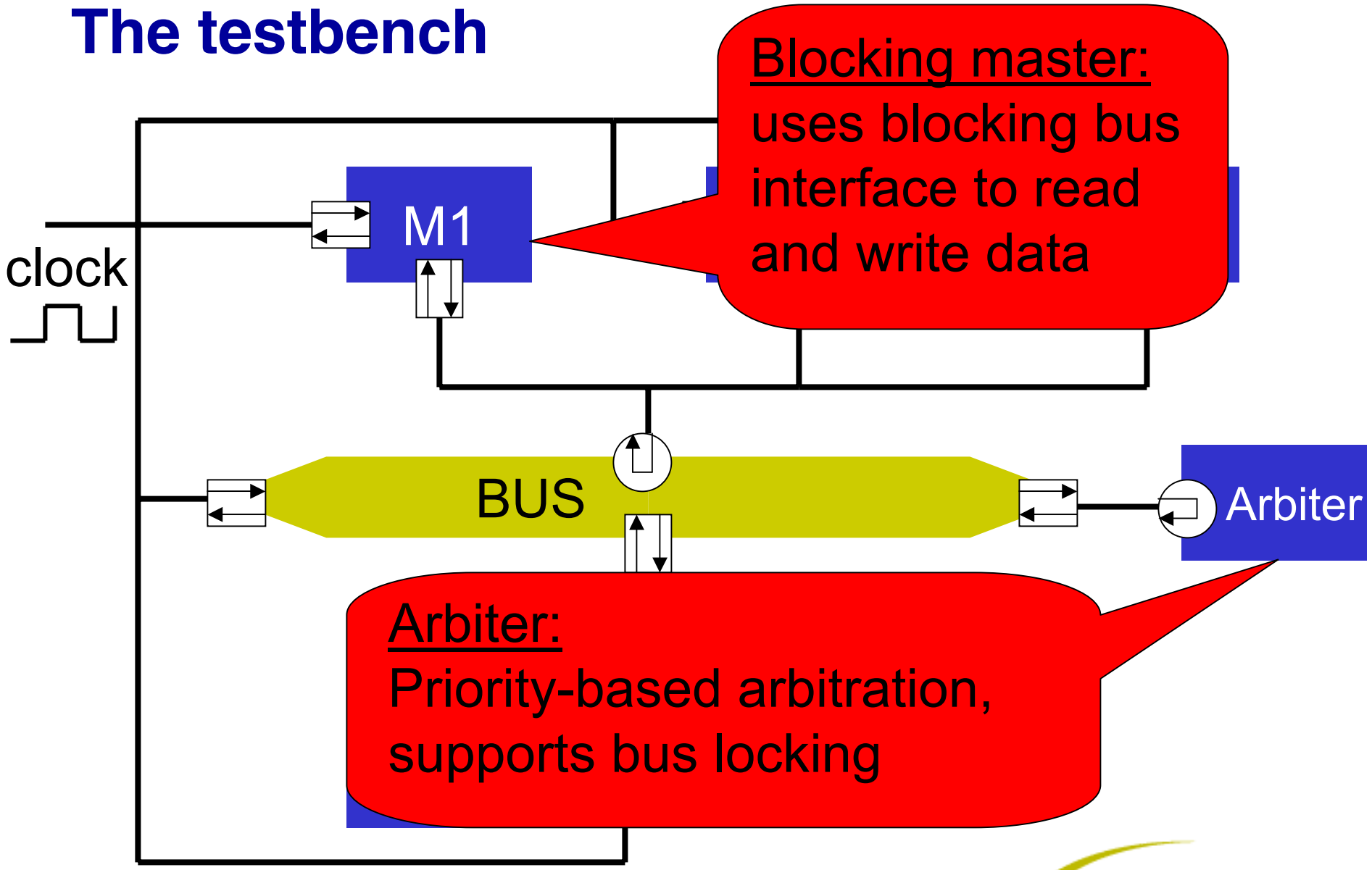
The testbench



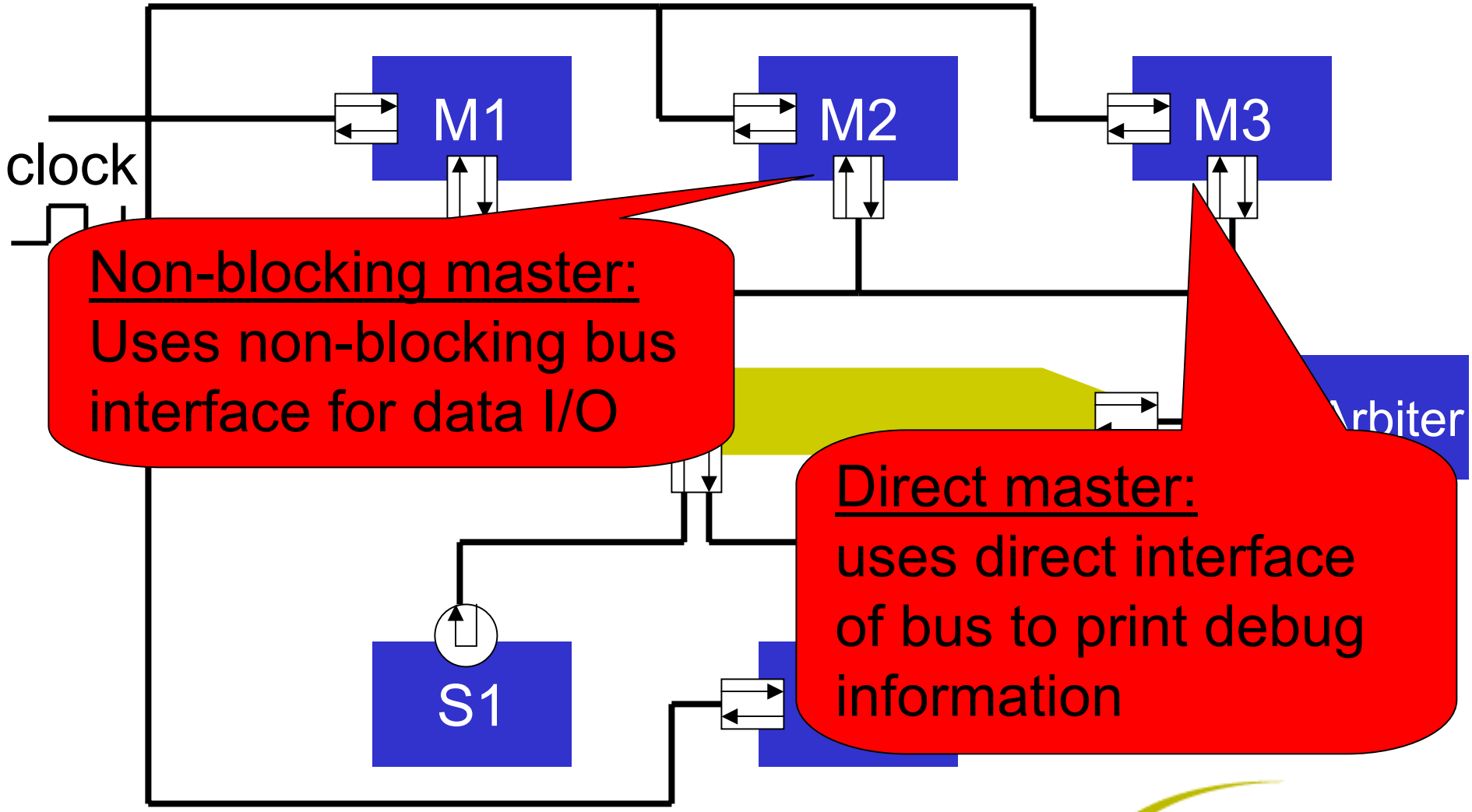
see simple_bus_test.h

SYSTEMC™

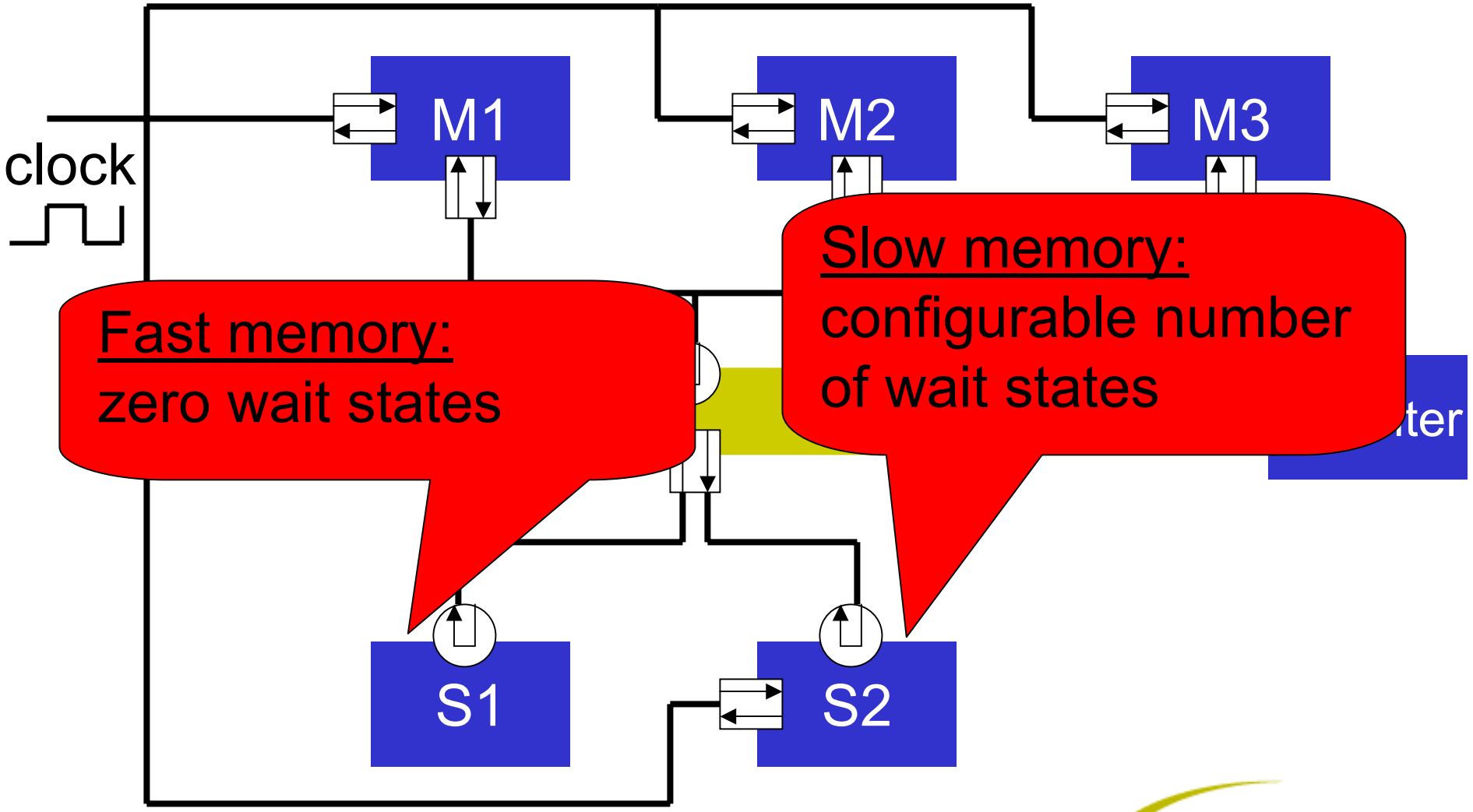
The testbench



The testbench



The testbench



The testbench (cont'd)

- Most modules are configurable
  - Masters
    - Priority (not direct master)
    - Delay / timeout
    - Bus locking on/off (not direct master)
  - Slaves
    - Address ranges
    - Number of wait-states (only slow memory)
  - Bus, arbiter, direct master
    - Verbosity
- Change parameter settings in simple_bus_test.h
- See README.txt for details

That's it!

Thank you and have fun trying it out!