4th European SystemC Users Group Meeting

http://www-ti.informatik.uni-tuebingen.de/systemc

Copenhagen

October 5th, 2001, 1100-1600

SystemC 2.0 Tutorial

Thorsten Grötker
R & D Manager
Synopsys, Inc.

Motivation

- **SystemC 1.0**

  HW modeling (RTL and behavioral)

- **SystemC 2.0**

  extend scope to System-Level Modeling

- **System-Level Modeling**
  - functional models
  - transaction-level platform models
  - high-level architecture models

How do we achieve this?

- We use a flexible and powerful *Model of Computation* (MoC).

- A MoC is characterized by
  - the model of time employed,
  - the rules for process activation, and
  - the supported means of communication.

MoC: Model of Time

- **SystemC 1.0**

  Relative floating-point model of time (double)

- **SystemC 2.0**

  Absolute (64 bit) unsigned integer model of time

- **Why?**
  - **Avoid finite precision effects, e.g. underflow**
  - **Use absolute model of time: define time units (IP exchange)**

MoC: Rules for Process Activation

- 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);
```

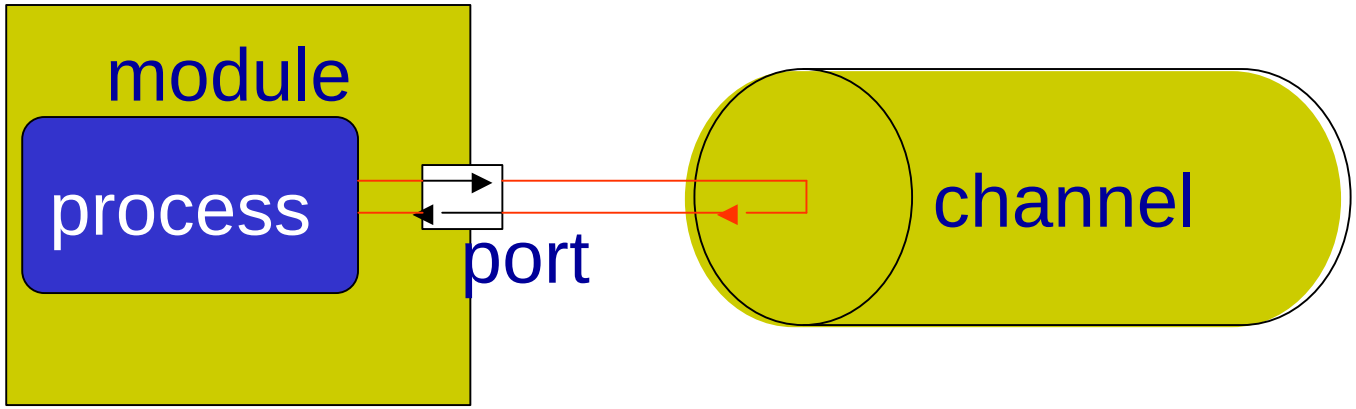
MoC: Communication

- SystemC 1.0
  - Fixed set of communication channels (sc_signal, ...) and ports (sc_in, sc_out, ...).
- SystemC 2.0
  - richer set of predefined channels (HW signals, FIFO, semaphore, mutex, ...)
  - user-defined
    - interfaces
    - channels
    - ports

Define your own bus, message queue, … etc.

SYSTEMC™

Interface Methods Calls

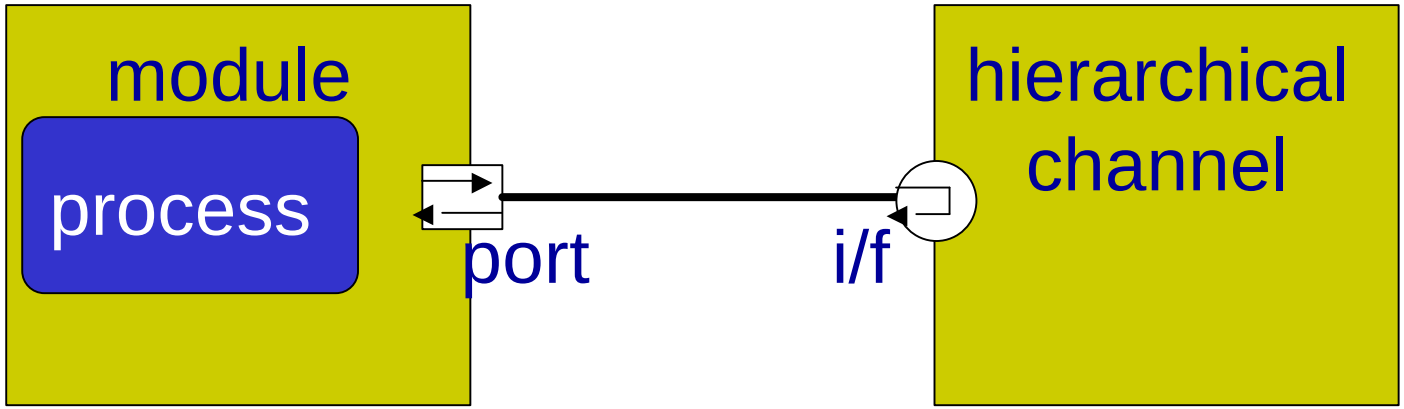



```
module::process() {
      ...
      port->some_method(42);
      ...
}
```


SYSTEMC™

Hierarchical channels

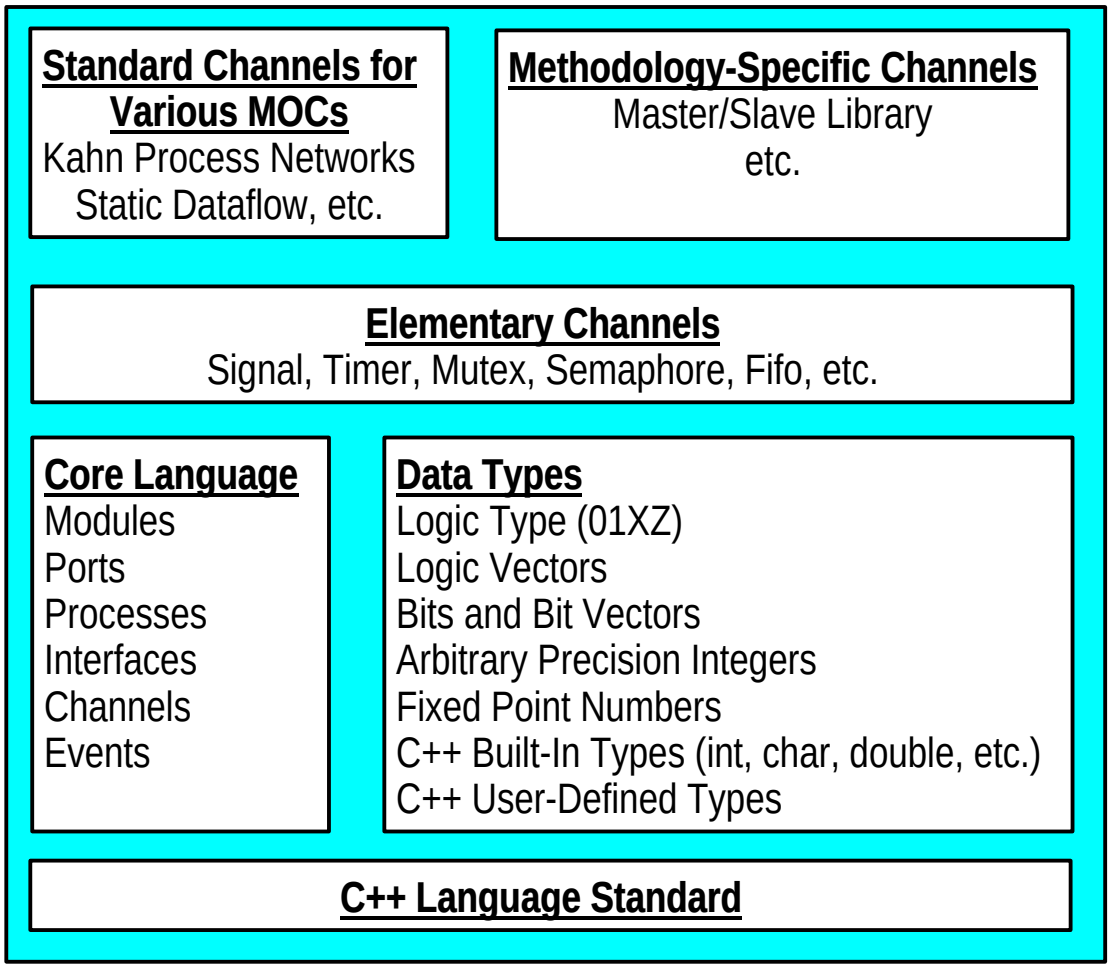
- Channels can be hierarchical, i.e. they can contain modules, processes, and channels.
- A module that implements an interface is a hierarchical channel.



SystemC 2.0 Language Architecture

Upper layers are built cleanly on lower layers.

Lower layers can be used without upper layers.

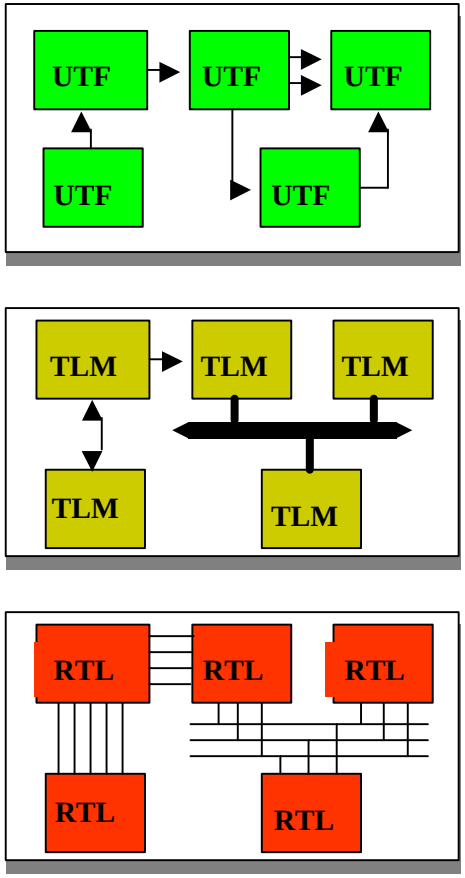


Model of Computation

- Very powerful and flexible
- Supports well known MoCs such as
  - discrete-event models
    - RTL / behavioral HW models
    - network modeling
    - transaction-level SoC platform modeling
  - Kahn process networks
    - static multi-rate data flow
    - dynamic data flow
  - Communicating Sequential Processes

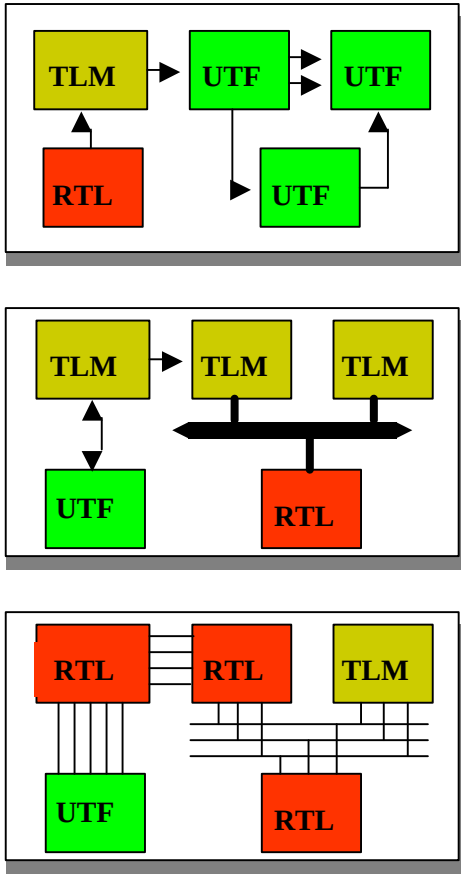
SystemC 2.0: A single language spanning multiple levels of abstraction

- (untimed) functional level
  - → executable specification
- transaction level
  - → platform design, HW/SW co-verification
- pin level
  - → RTL/behavioral HW design and verification



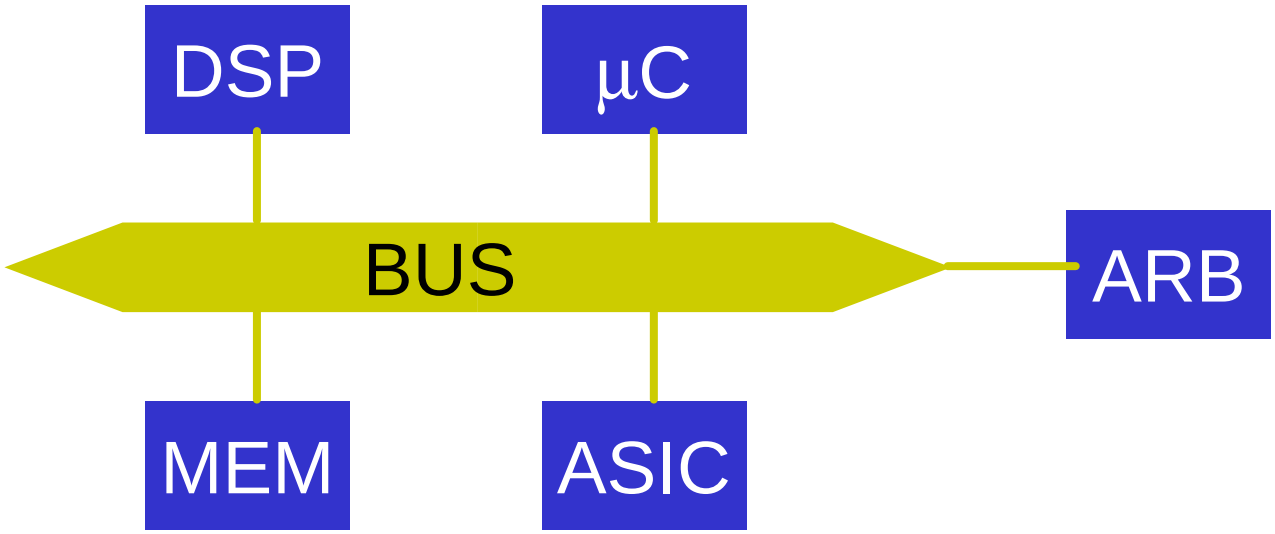
Gradual refinement, early verification

- No need to refine executable specification in one giant step into RTL model
- Bus-cycle accurate transaction level models for fast platform simulation (>> 100 k cycles/sec) early in the development process
- No need to glue together different simulators for co-simulation



Efficient platform modeling

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



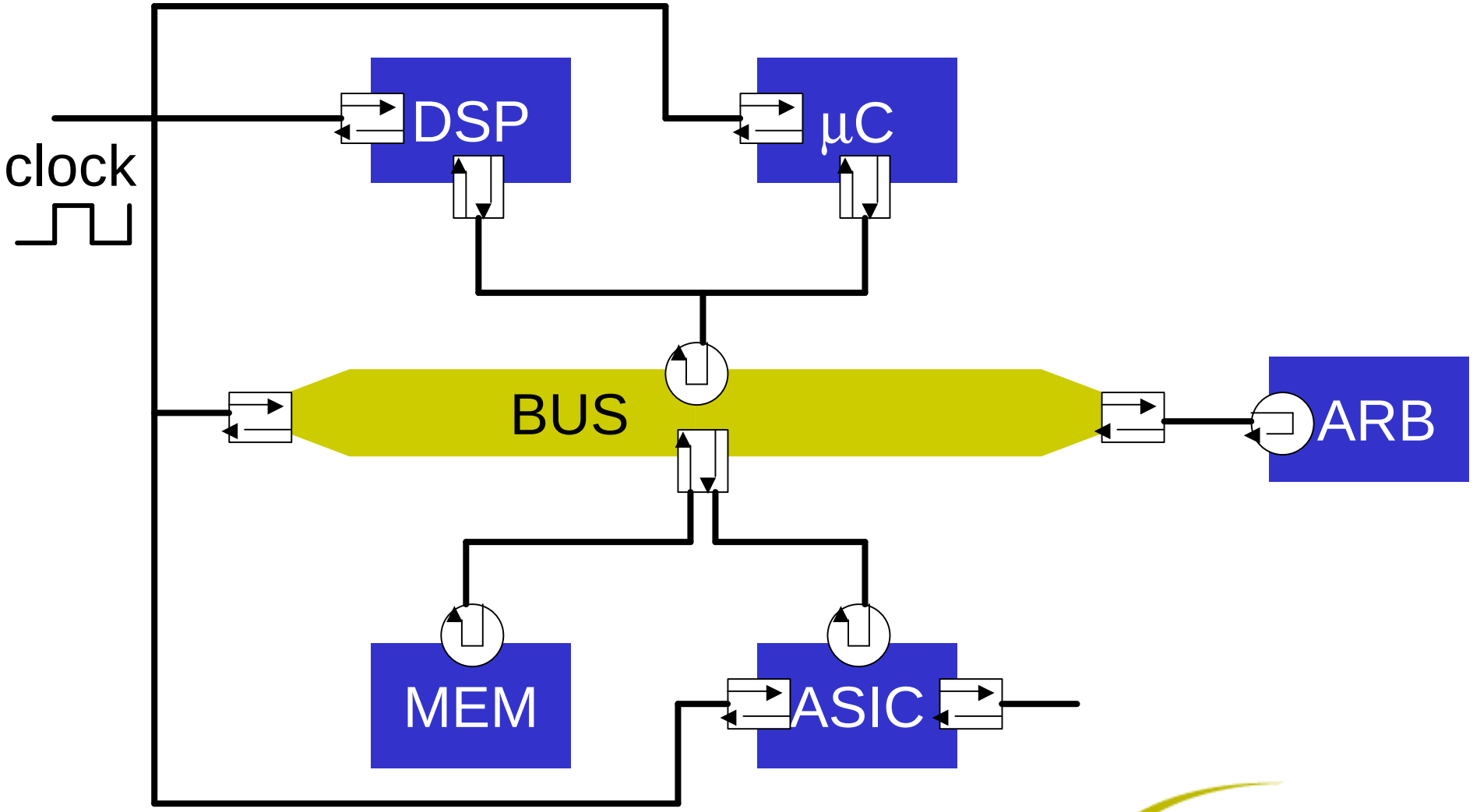
Moving from pin-level to transaction-level models (TLM) is mandatory!

Example: Simple bus model

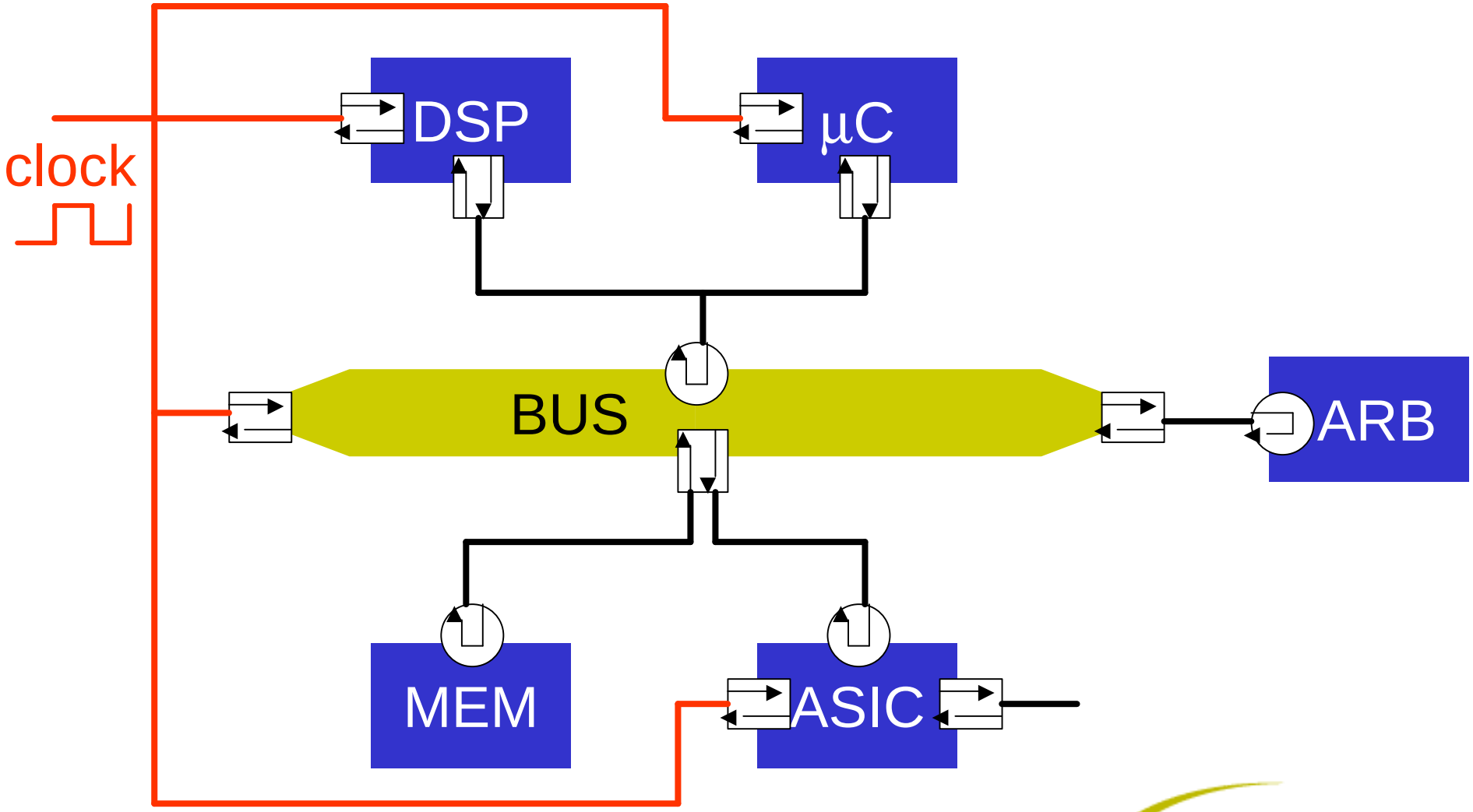
- Cycle-accurate transaction-level model.
- "Simple" =
  - No pipelining
  - No split transactions
  - No master wait states
  - No request/acknowledge scheme
  - …

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

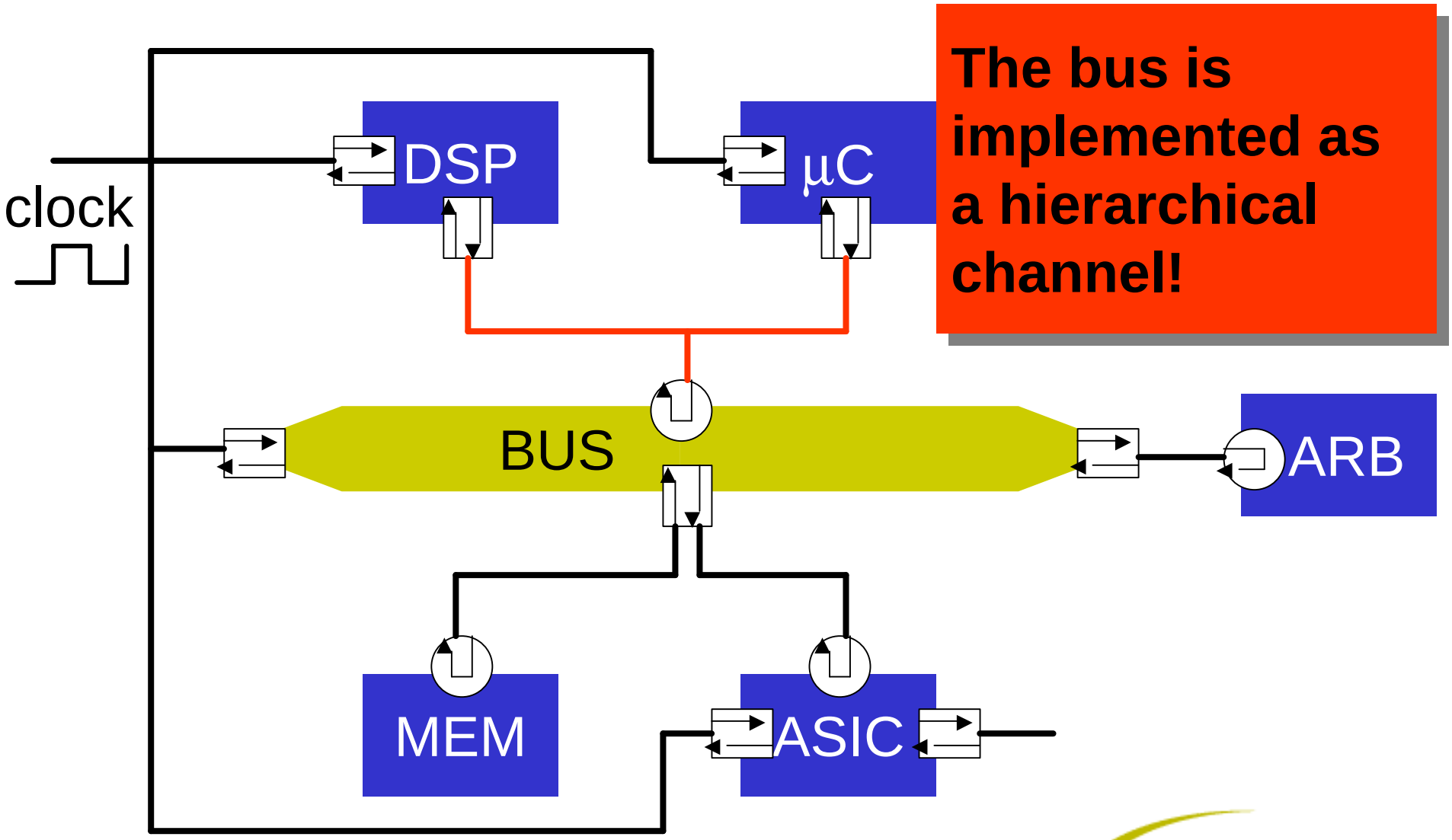
SystemC 2.0 transaction-level model



SystemC 2.0 transaction-level model



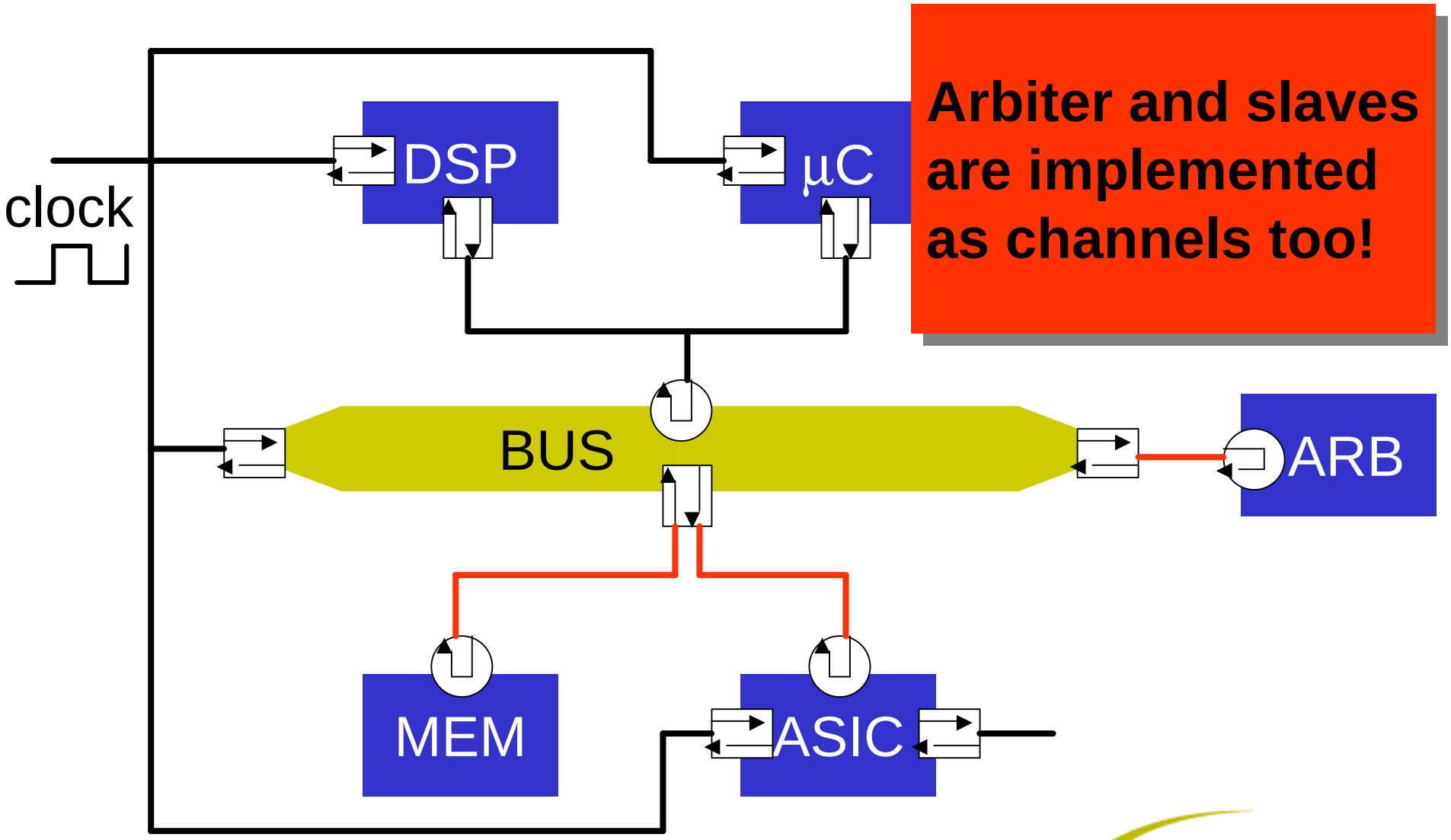
SystemC 2.0 transaction-level model



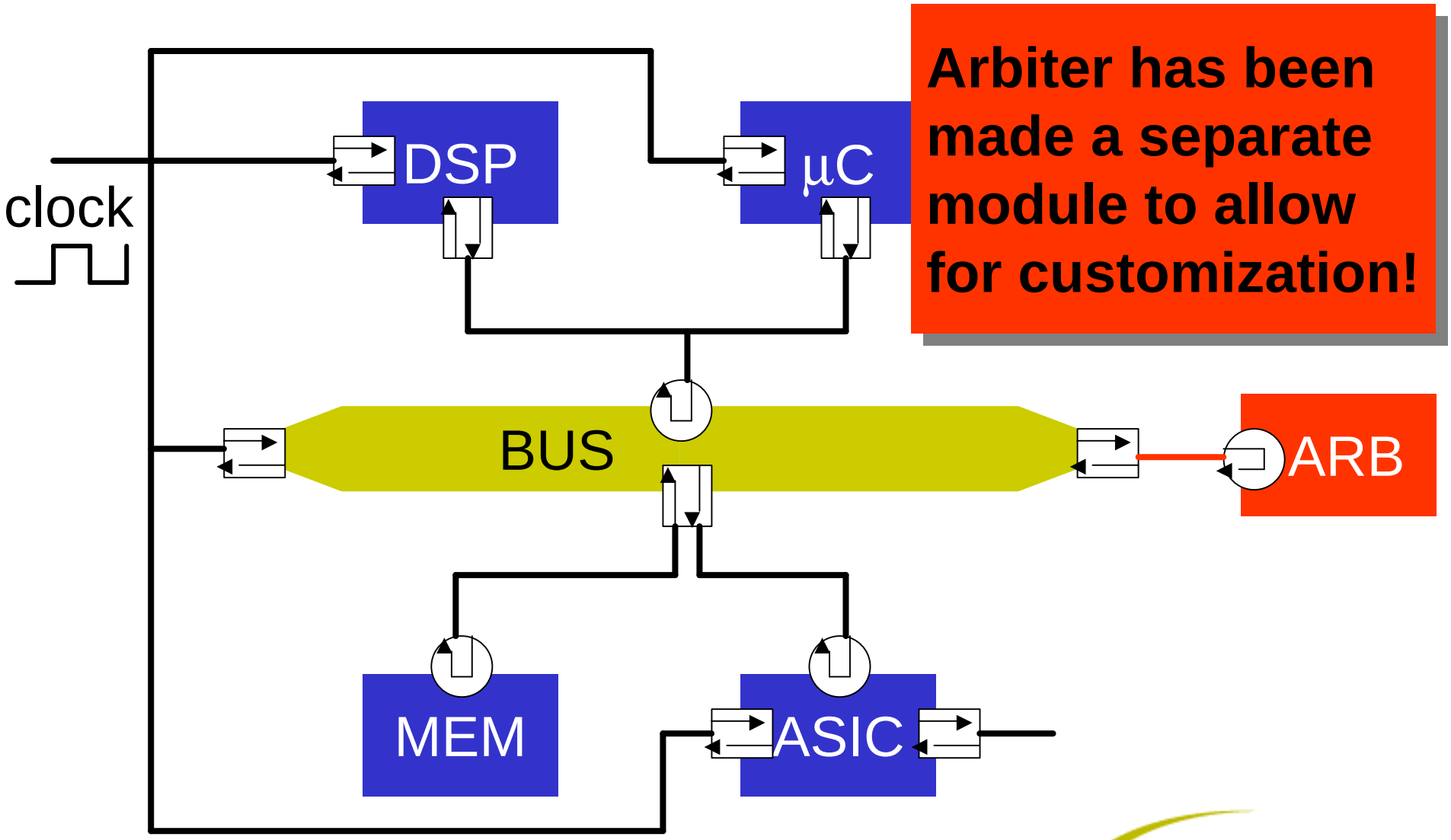
The bus is implemented as a hierarchical channel!

SYSTEMC™

SystemC 2.0 transaction-level model

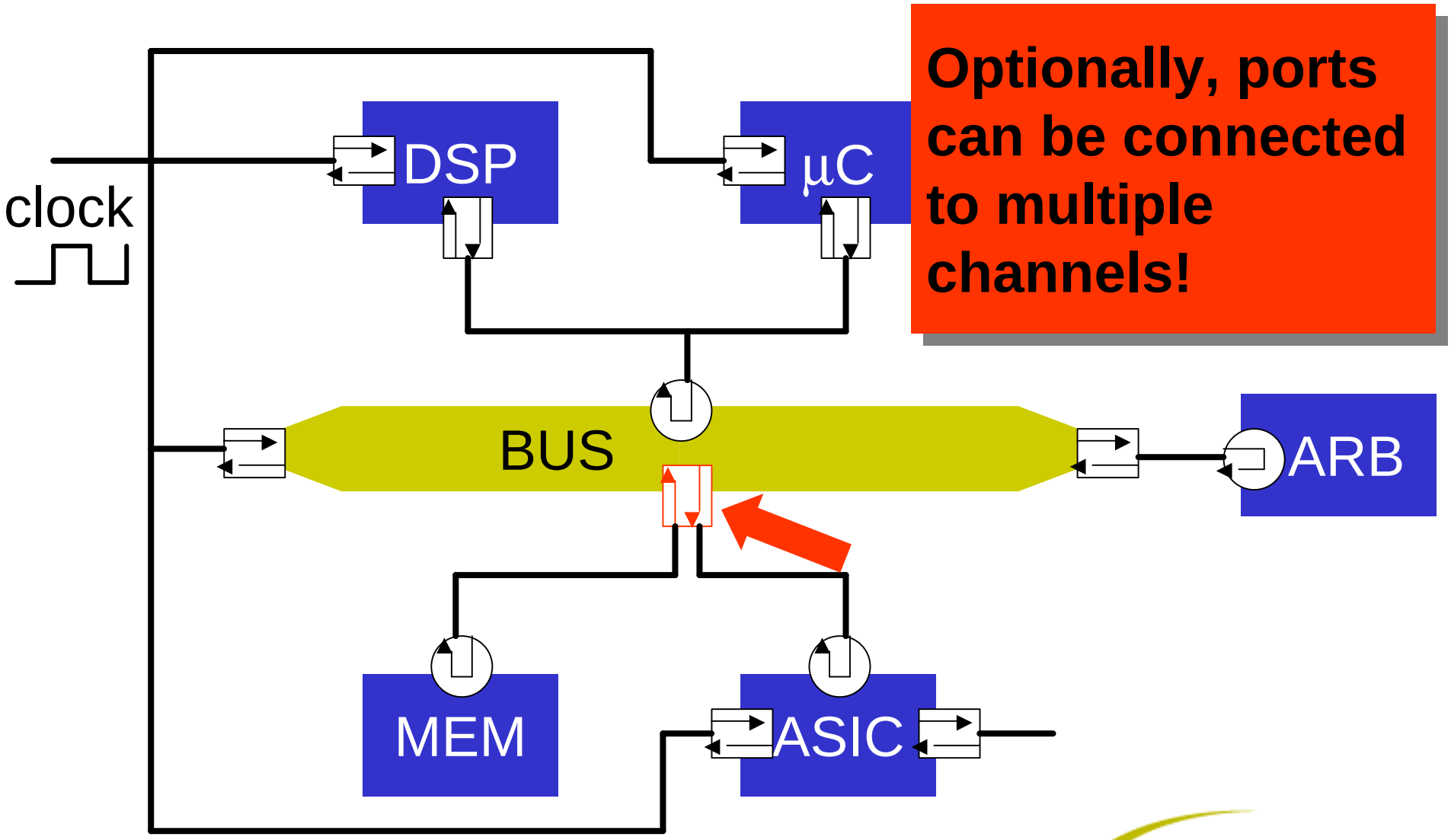


SystemC 2.0 transaction-level model

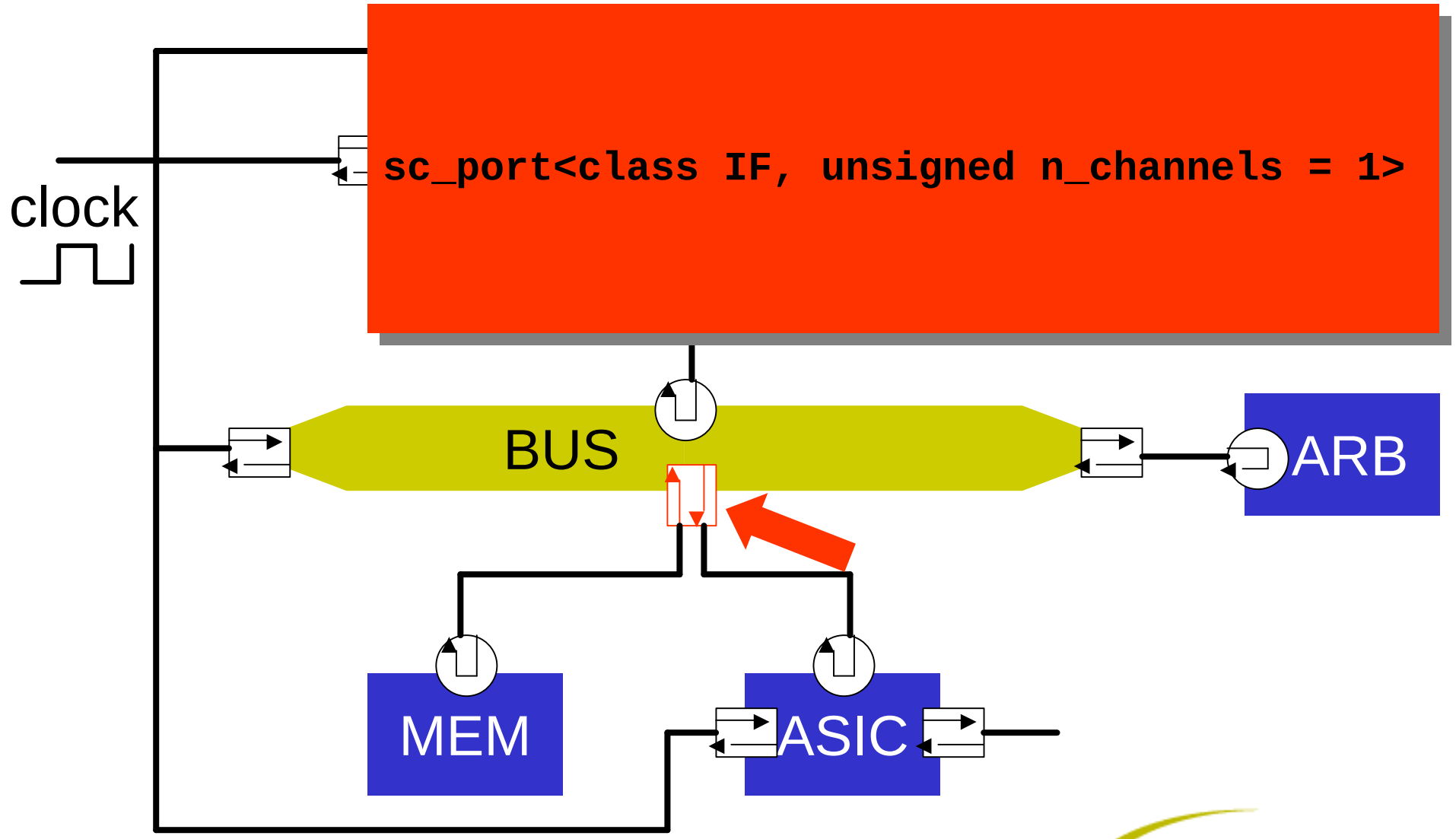


Arbiter has been made a separate module to allow for customization!

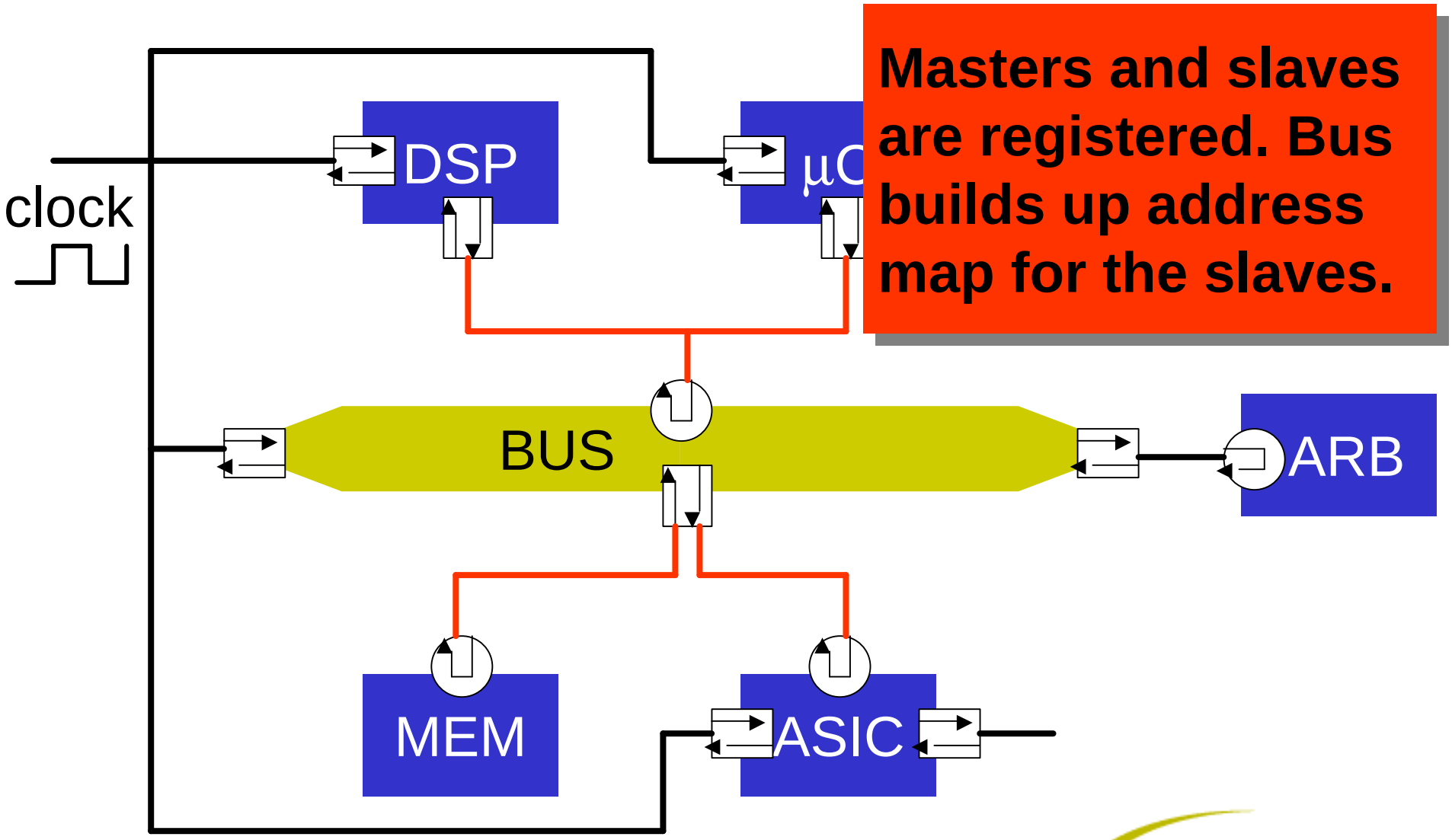
SystemC 2.0 transaction-level model



SystemC 2.0 transaction-level model



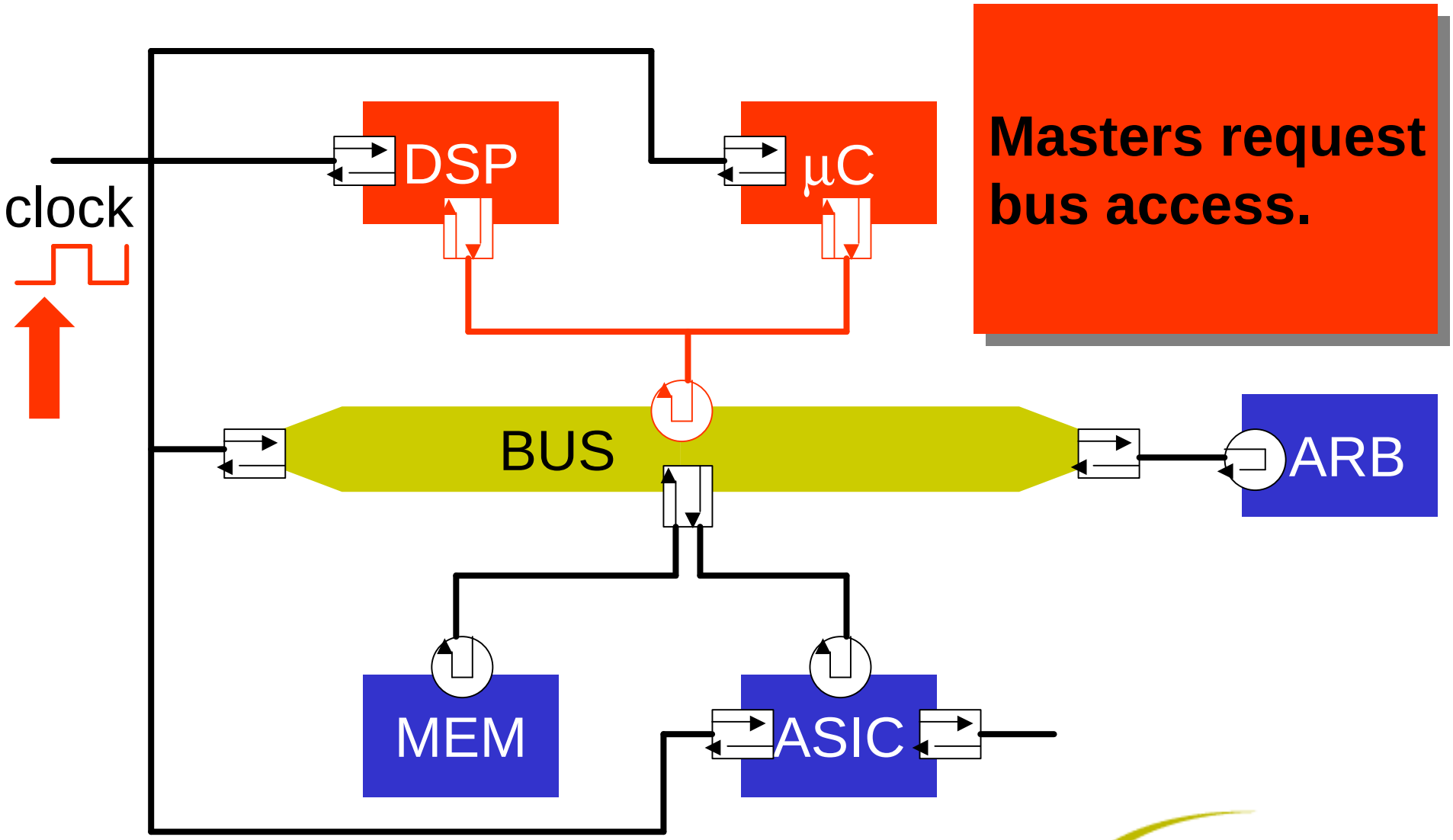
Initialization



Masters and slaves are registered. Bus builds up address map for the slaves.

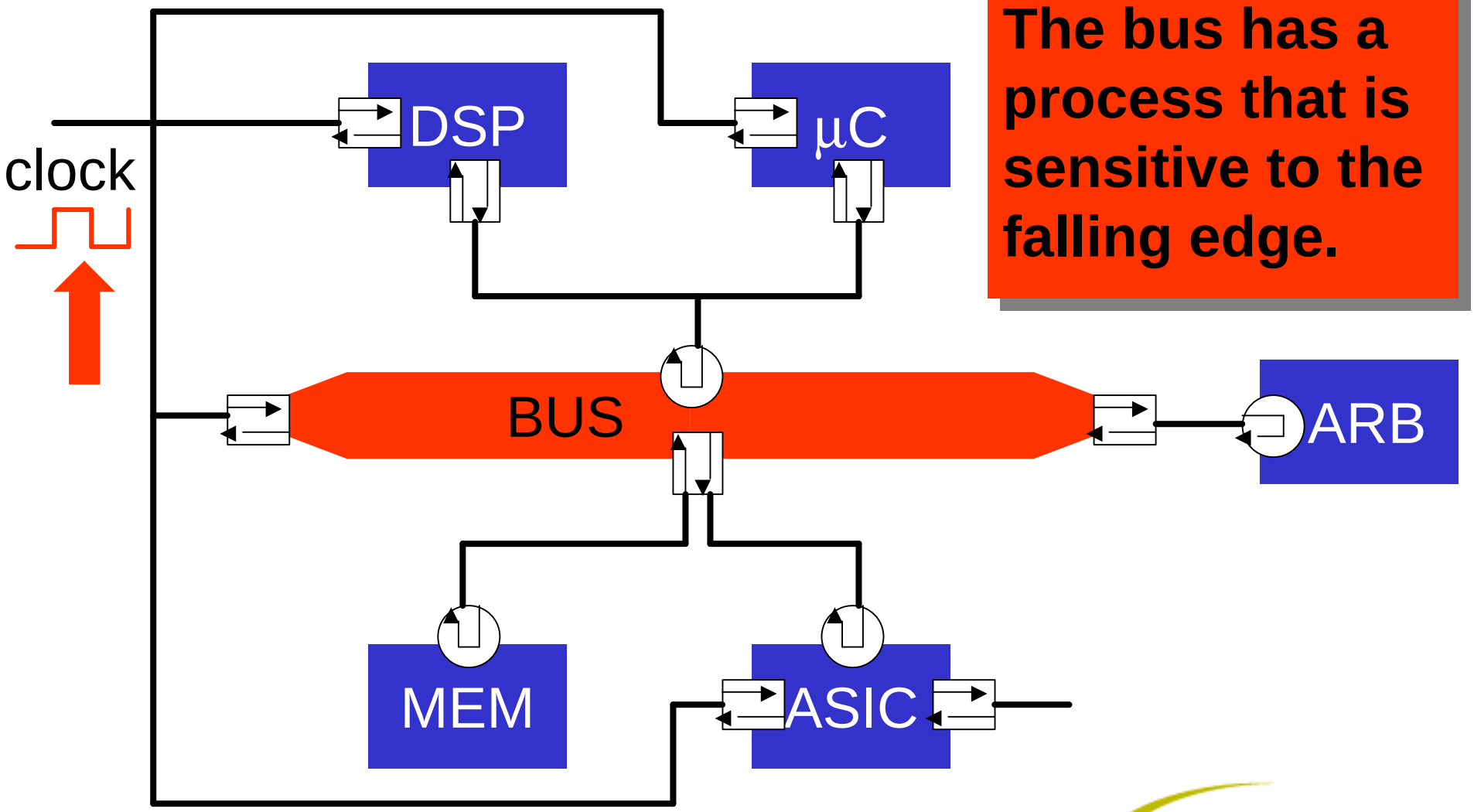
SYSTEMC™

Rising clock edge



SYSTEMC™

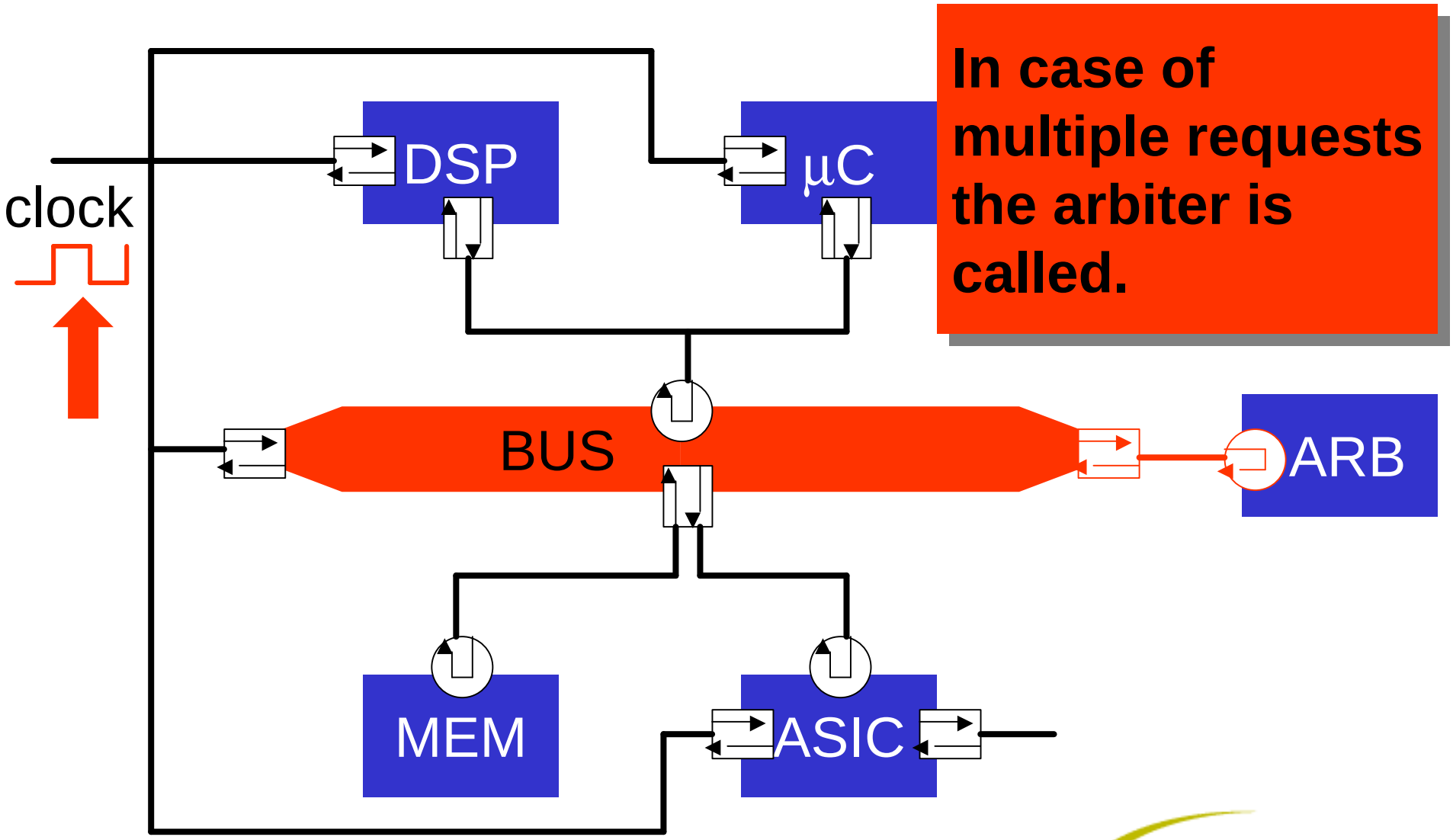
Falling clock edge



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

SYSTEMC™

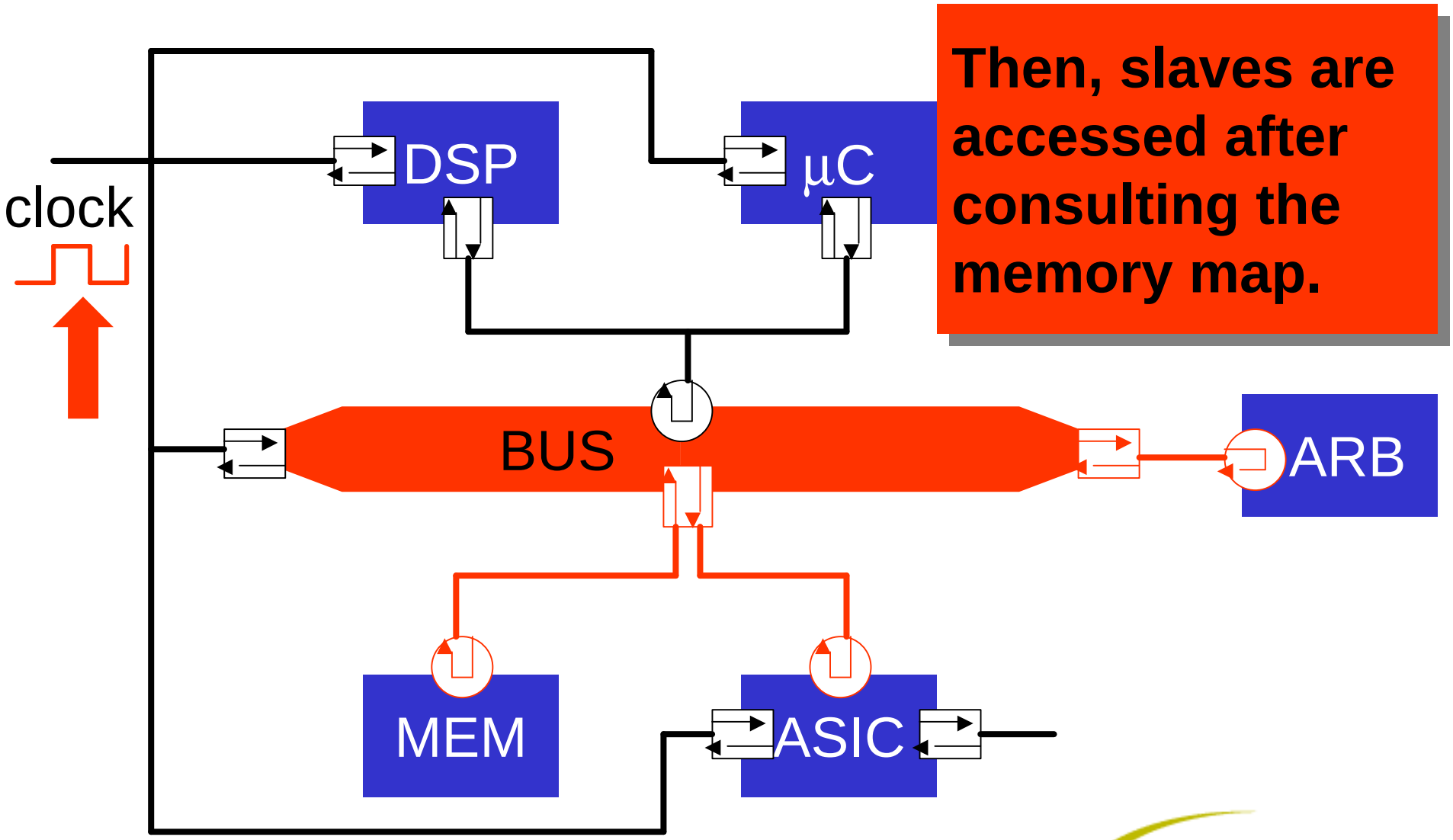
Falling clock edge



In case of multiple requests the arbiter is called.

SYSTEMC™

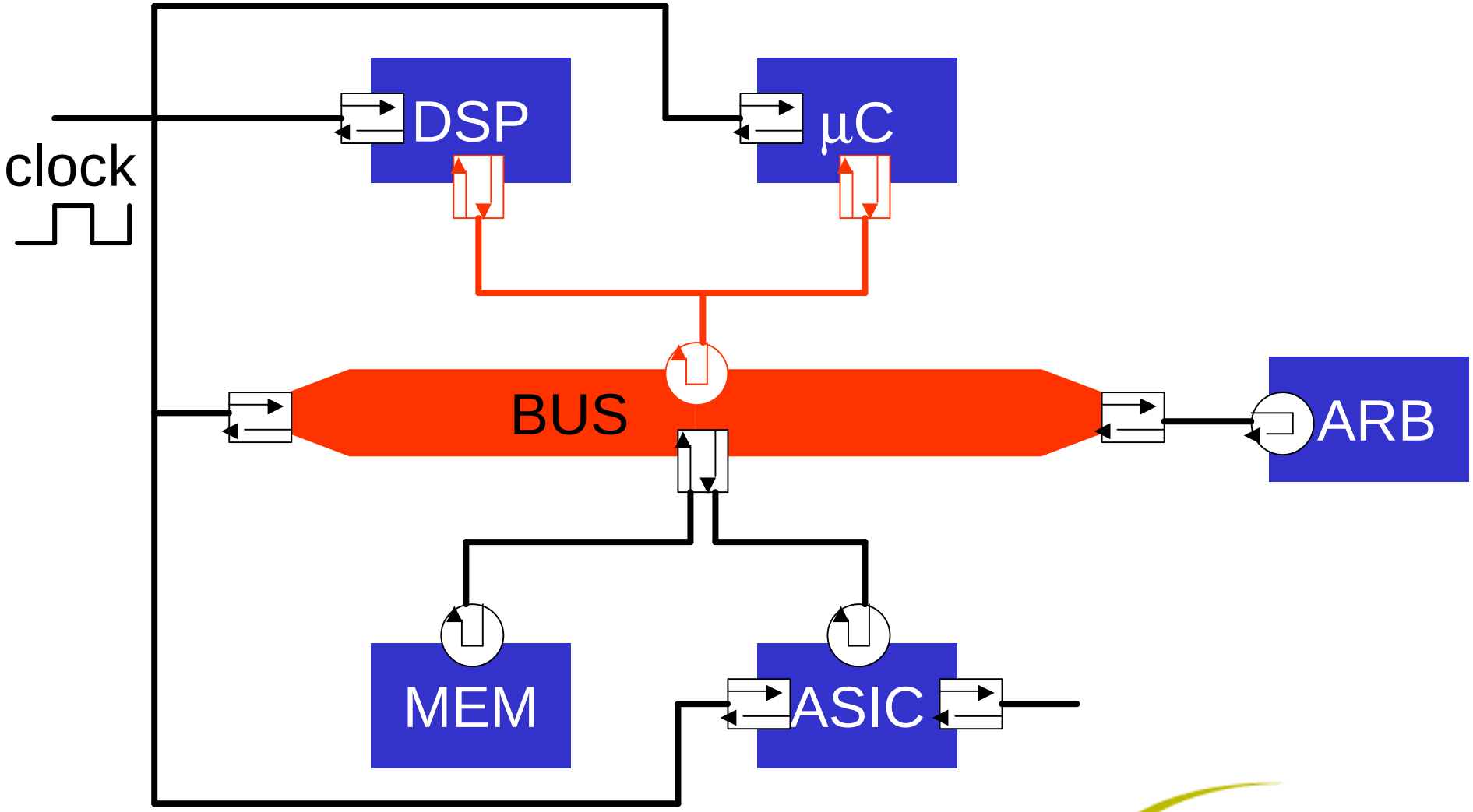
Falling clock edge



Then, slaves are accessed after consulting the memory map.

SYSTEMC™

Master interfaces



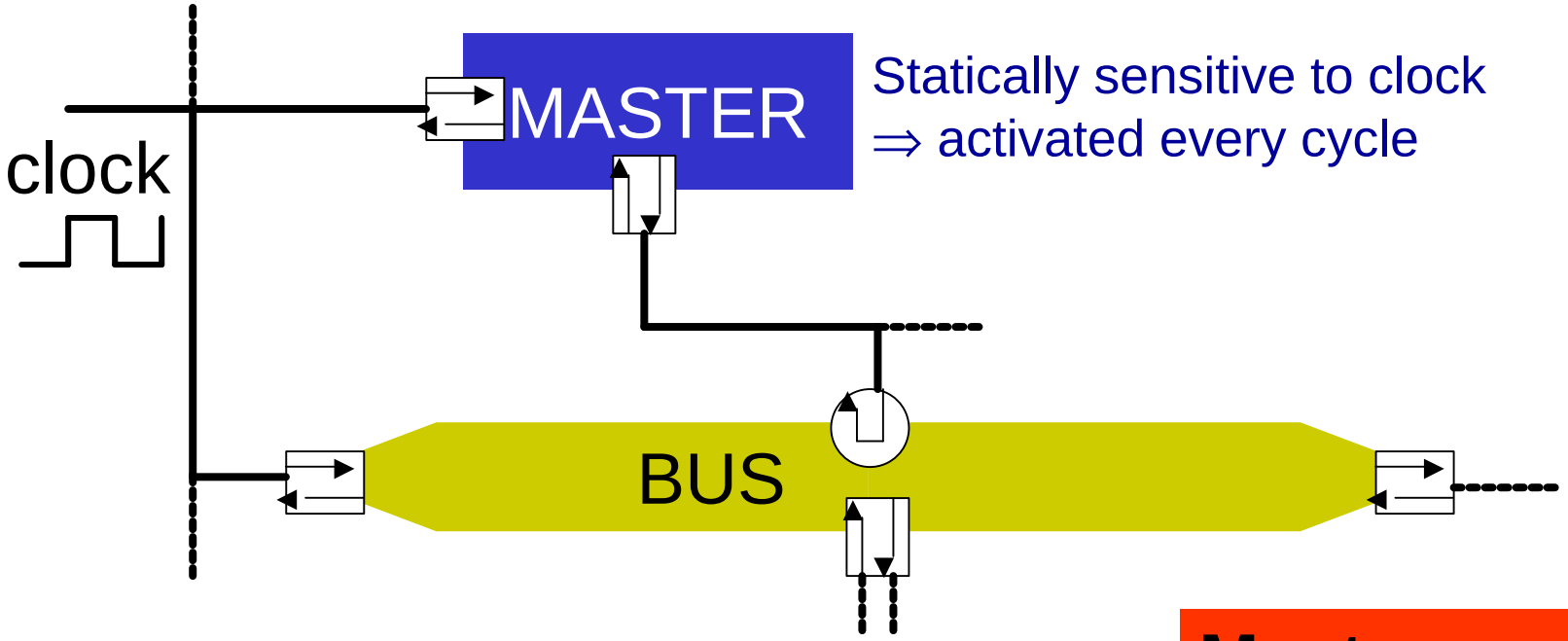
Master interfaces of the bus

- **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(data*, start_address,
                    length = 1, priority = 1,
                    lock = false);
  ```
- ```
  status burst_write(data*, start_address,
                     length = 1, priority = 1,
                     lock = false);
  ```
- **“Blocking” because call returns only after complete transmission is finished.**

Dynamic sensitivity



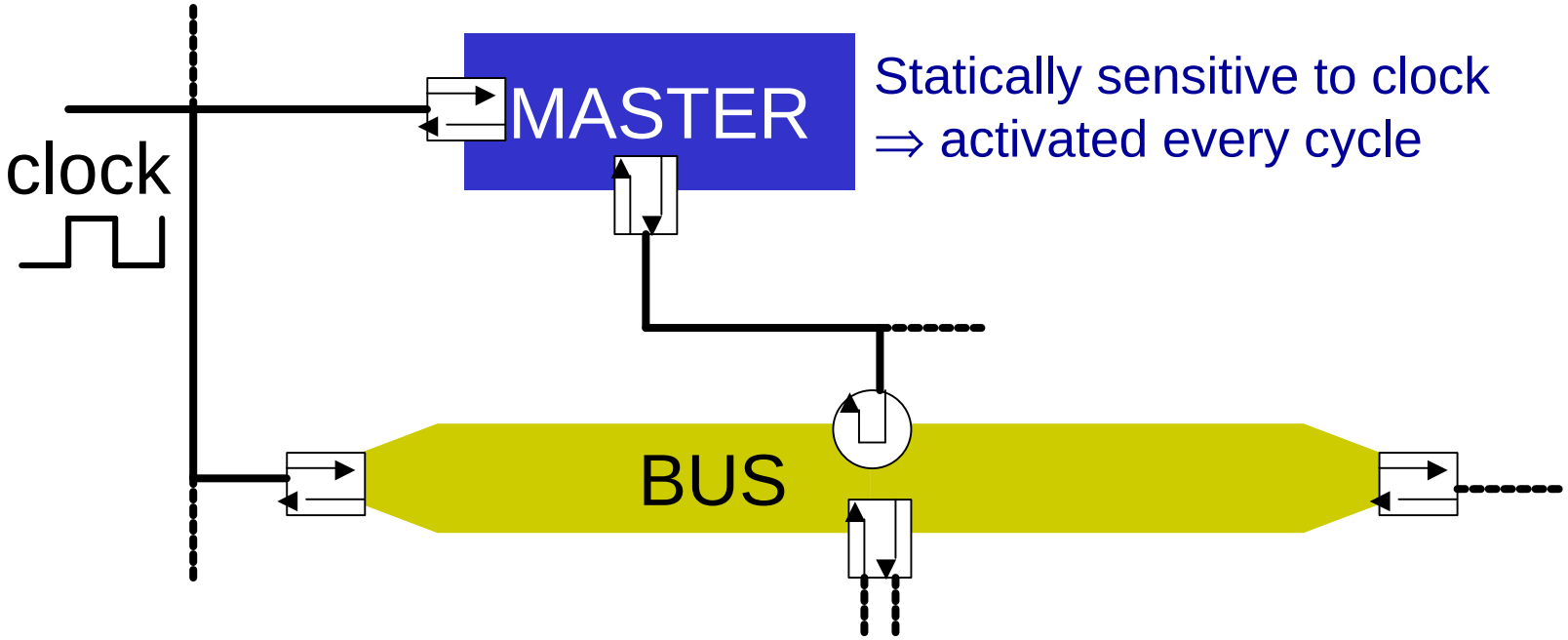
Statically sensitive to clock
⇒ activated every cycle

```
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**

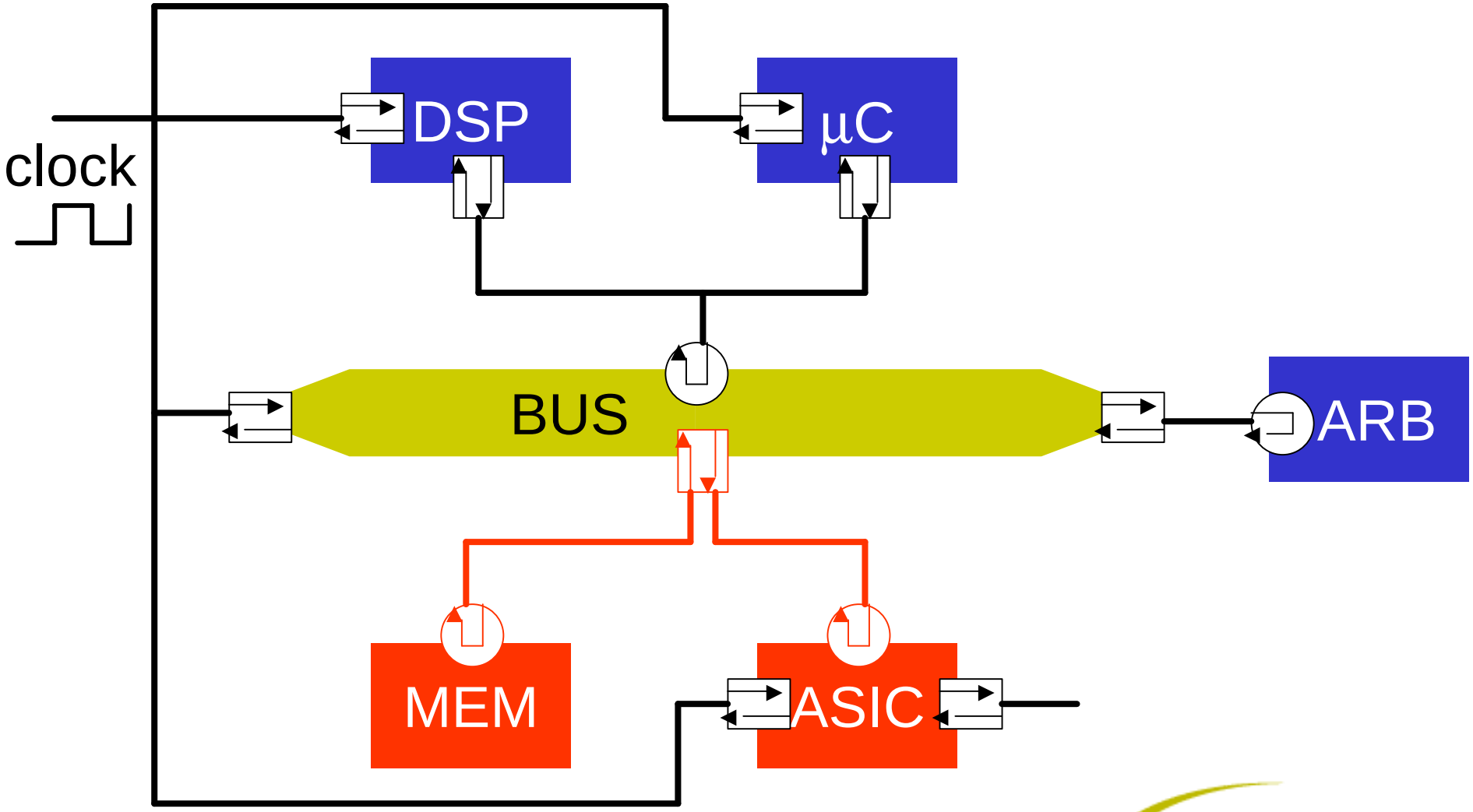
Non-blocking master interface

- `status get_status();`
- `status read(data*, address, priority = 1, lock = false);`
- `status write(data*, address, priority = 1, 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 interface

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

Slave interface



Slave interfaces

- `status read(data*, address);`
- `status write(data*, address);`
- `void address_range(start_address*, end_address*);`
  `// used to build up address map`

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

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

They are ...

- relatively easy to develop and extend
- easy to use
- fast
  - use of IMC Þ function calls instead of HW signals and control FSMs
  - use of dynamic sensitivity Þ 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

Benefits of SystemC v2.0

- **Enables, fast smooth system design**
  - **Communication can modeled and refined independent of function**
- **Supports virtually all system modeling needs**
  - **Flexible semantic foundation additions support most models of computation within one environment**
  - **Leverages all existing v1.0 and v1.1beta capabilities**
- **Broadly applicable solution**
  - **Designed by 12 experts from six different EDA and System IC companies**
  - **Tuned for both EDA tool and IP use**

Summary

- SystemC is moving forward:
  - New Version 2.0 especially supports System-Level Modeling:
    - functional models
    - transaction-level platform models
    - high-level architecture models
- More and more real life applications done with SystemC
- Join ESCUG for up-to-date information