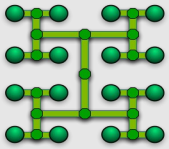
Universidade de Brasília

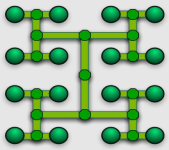
Modelagem em Nível Transacional

Ricardo Jacobi
Departamento de Ciência da Computação
Universidade de Brasília
jacobi@unb.br



LAICO

Universidade de Brasília

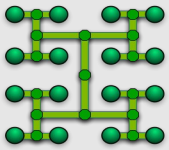


LAICO

Introdução

- Um “sistema complexo” é um conceito que depende do contexto e época de sua utilização
- Nos tempos de SSI (*small scale integration*) circuitos com centenas de portas eram de alta complexidade
- Atualmente, com o advento dos sistemas monolíticos (*SoC – system on chip*), um sistema complexo inclui diversos módulos que interagem, sendo cada um destes por sua vez de complexidade variável,abrangendo processadores risc, DSP´s, memórias e módulos dedicados

Universidade de Brasília



LAICO

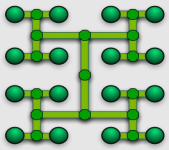
Introdução

- Para se acompanhar a crescente evolução da complexidade as metodologias e ferramentas de projeto devem se desenvolver de forma correspondente
- Com equipes compostas por diversos pesquisadores com perfis variados, o projeto de um SoC requer uma formalização na especificação e modelagem do sistema para minimizar o risco de erro

Introdução

- Sistemas complexos são usualmente modelados primeiro em software (C, C++, Java) para uma primeira verificação de sua funcionalidade
- A descrição em software permite verificar a funcionalidade do sistema e prover uma descrição formal de seu comportamento
- Os módulos em software podem descrever subsistemas a serem desenvolvidos (sintetizados) ou representar módulos de propriedade intelectual a serem reutilizados ou ainda funções a serem compiladas para os processadores do SoC

Universidade de Brasília

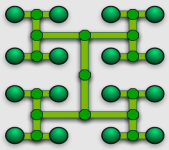


LAICO

Introdução

- As descrições puramente em software carecem de recursos para descrição de aspectos físicos e temporais do hardware, como temporização, conectividade, paralelismo, etc
- Nesse contexto, linguagens como SystemC provêm recursos que permitem incorporar aspectos do hardware em descrições de nível de abstração mais alto

Universidade de Brasília

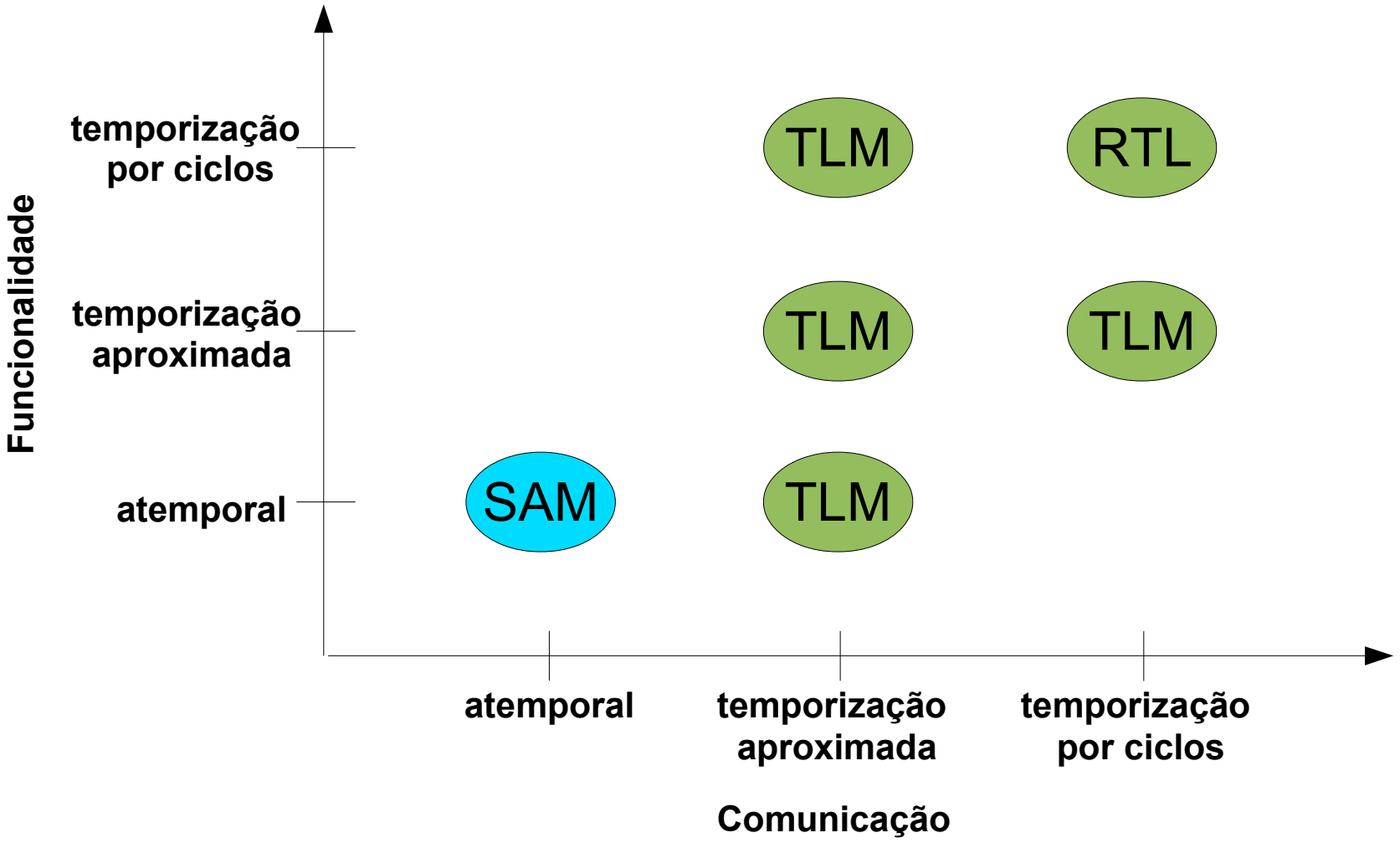


LAICO

Introdução

- A modelagem em nível de transações (TLM – *Transaction Level Modeling*) é uma metodologia que está sendo proposta como alternativa para os primeiros passos do projeto de SoCs
- Não há uma definição precisa do que seja TLM
- Algumas classificações foram propostas na literatura, como Lukai e Gajski em CODES´03
- Um conceito básico em TLM é a separação entre computação x comunicação

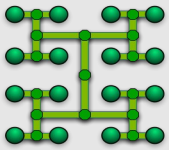
Classificação TLM



TLM

- Uma descrição de um sistema em nível transacional abstrai detalhes de comunicação entre os módulos, focando na funcionalidade das transferências de dados e não em sua implementação
- Uma **transação** é um conjunto completo de informações necessárias a execução de uma operação. Por exemplo:
  - Um bloco de memória em operação de acesso
  - Um pacote *ethernet*
  - Um bloco de pixels para uma DCT

Universidade de Brasília



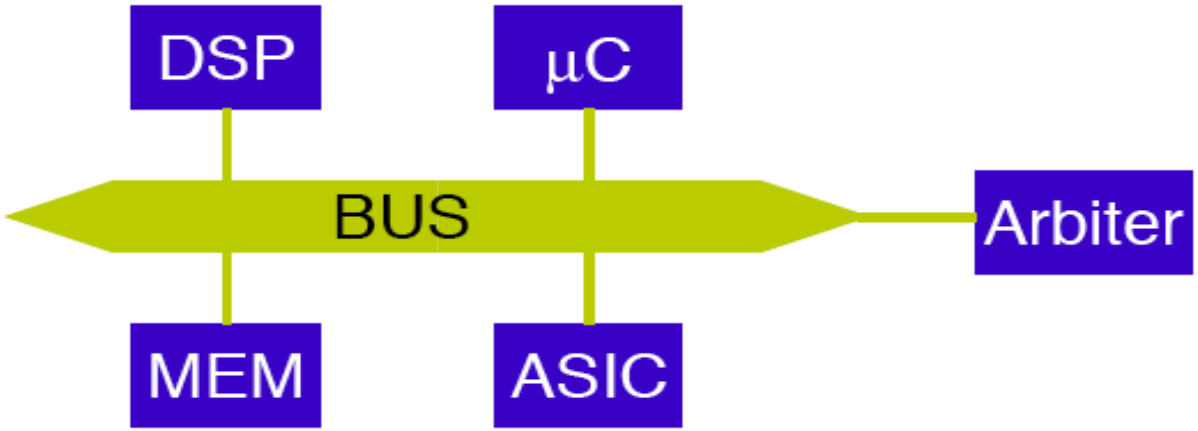
LAICO

Transações

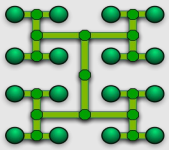
- Em SystemC, transações são definidas como métodos declarados em interfaces e implementados nos canais de comunicação
- Abstrai-se o *handshaking* detalhado dos sinais, sincronizando-se as operações através de operações de E/S bloqueantes e não-bloqueantes

Vantagens TLM

- Relativamente fácil de desenvolver, compreender e estender
- Simulação mais rápida do que descrições RT ou comportamentais
- Modelagem de hardware e software
- Rápido e preciso o suficiente para validar software antes da implementação detalhada do hardware



Universidade de Brasília

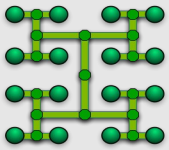


LAICO

Barramento Muito Simples

```
class very_simple_bus_if : virtual public sc_interface {
    public:
        virtual void burst_read(char *data,
                                unsigned addr,
                                unsigned length) = 0;
        virtual void burst_write(char *data,
                                unsigned addr,
                                unsigned length) = 0;
};

class very_simple_bus : public very_simple_bus_if, public sc_channel
{
 public:
    very_simple_bus(sc_module_name nm, unsigned ms, sc_time ct) :
        sc_channel(nm), _ct(ct) {
            _mem = new char [ms]; // memoria = array incluido
            memset(_mem, 0, ms);
    }
```

Universidade de Brasília

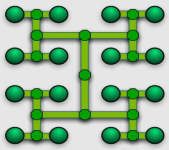


LAICO

Barramento Muito Simples

```
~very_simple_bus() { delete [] _mem; }
virtual void burst_read(char *data, unsigned addr, unsigned len){
   _bus_mutex.lock();          // acesso exclusivo
   wait(len * _ct);            // simula atraso de acesso
   memcpy(data, _mem + addr, len);   // transfere dados
   _bus_mutex.unlock();              // libera acesso
}

virtual void burst_write(char *data, unsigned addr, unsigned len){
   _bus_mutex.lock();          // acesso exclusivo
   wait(len * _ct);            // simula atraso de acesso
   memcpy(_mem + addr, data, len);   // transfere dados
   _bus_mutex.unlock();              // libera acesso
 }
 protected:
 char *_mem; sc_time _ct; sc_mutex _bus_mutex;
};
```

Universidade de Brasília

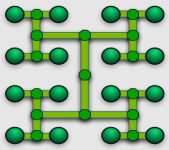


LAICO

Um Barramento Simples

- *simple_bus* é um exemplo fornecido com a distribuição do SystemC
- objetivo: modelar um sistema com múltiplos módulos interligados através de barramento
- modelo transacional preciso em nível de ciclo
- simplificação:
  - sem pipeline
  - sem *wait states*
  - sem transferências particionadas

Universidade de Brasília

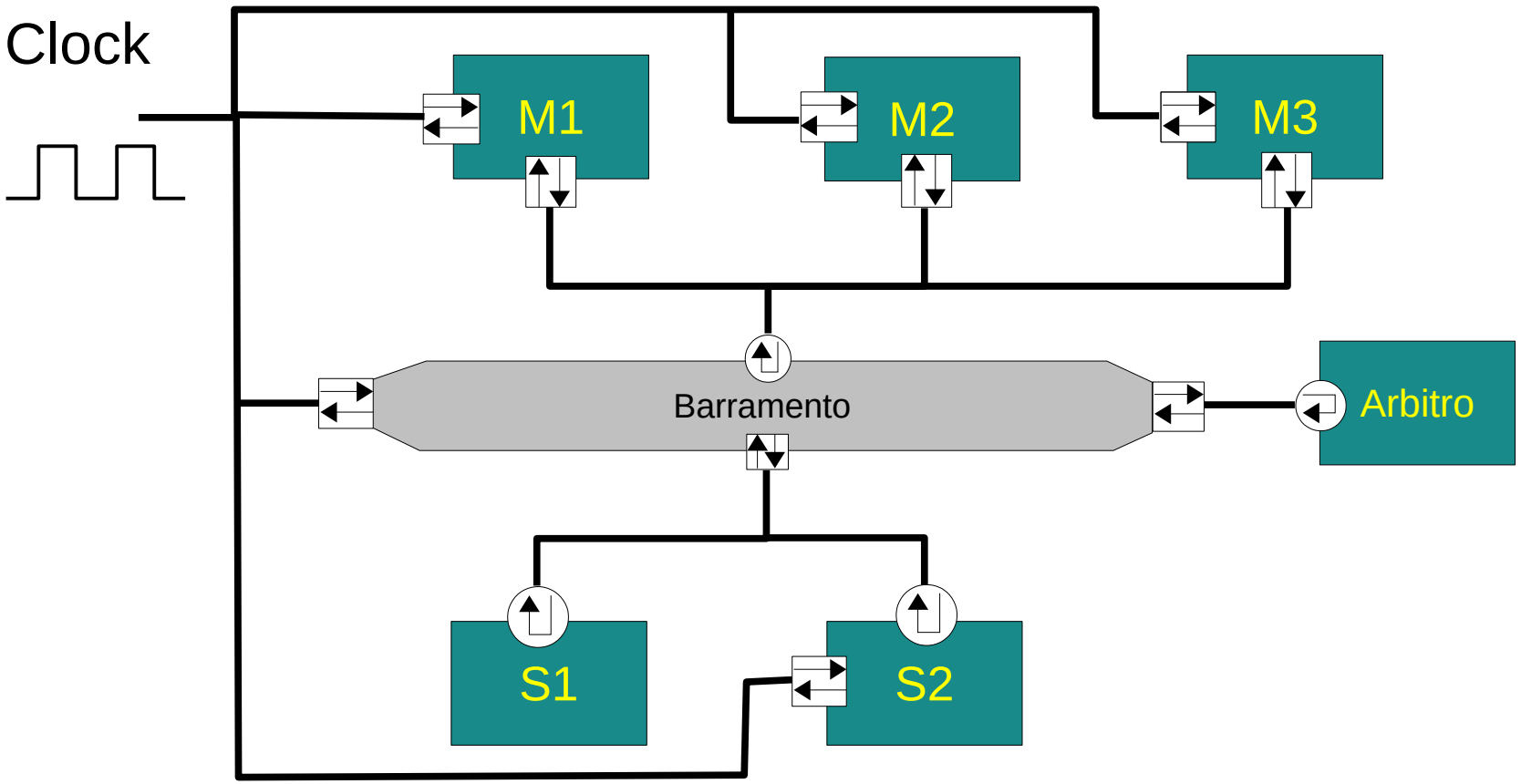


LAICO

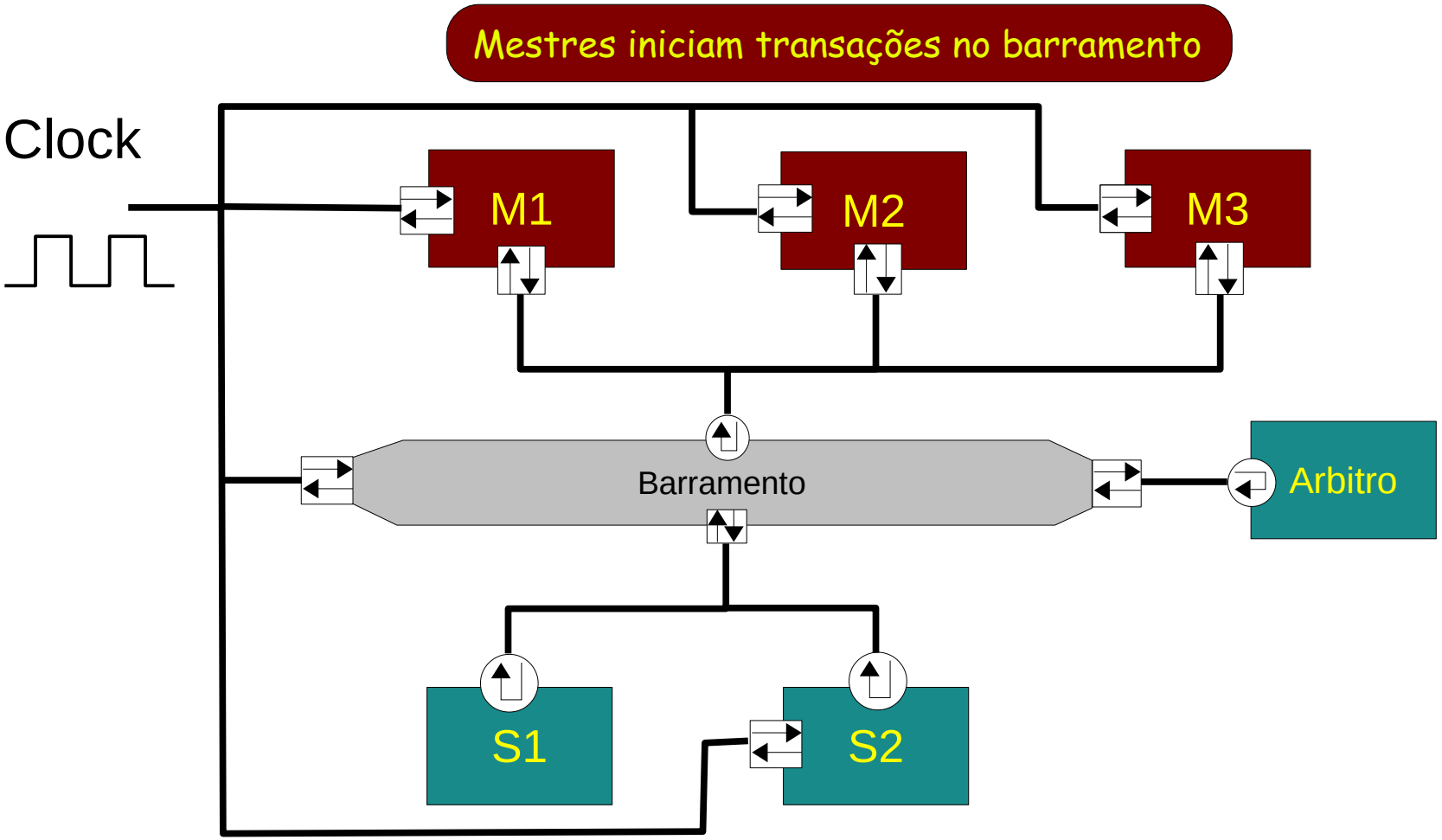
Estrutura do Barramento

- O barramento contém:
  - *Mestres*: CPU's, DSP's, microcontroladores...
  - *Barramento*: permite a interconexão entre os modulos através de transações
  - *Escravos*: ROM's, RAM's, módulos de E/S e ASIC's
  - *Árbitro*: seleciona a requisição a ser atendida
  - *Gerador de relógio*: gera um sinal de relógio para sincronização entre blocos

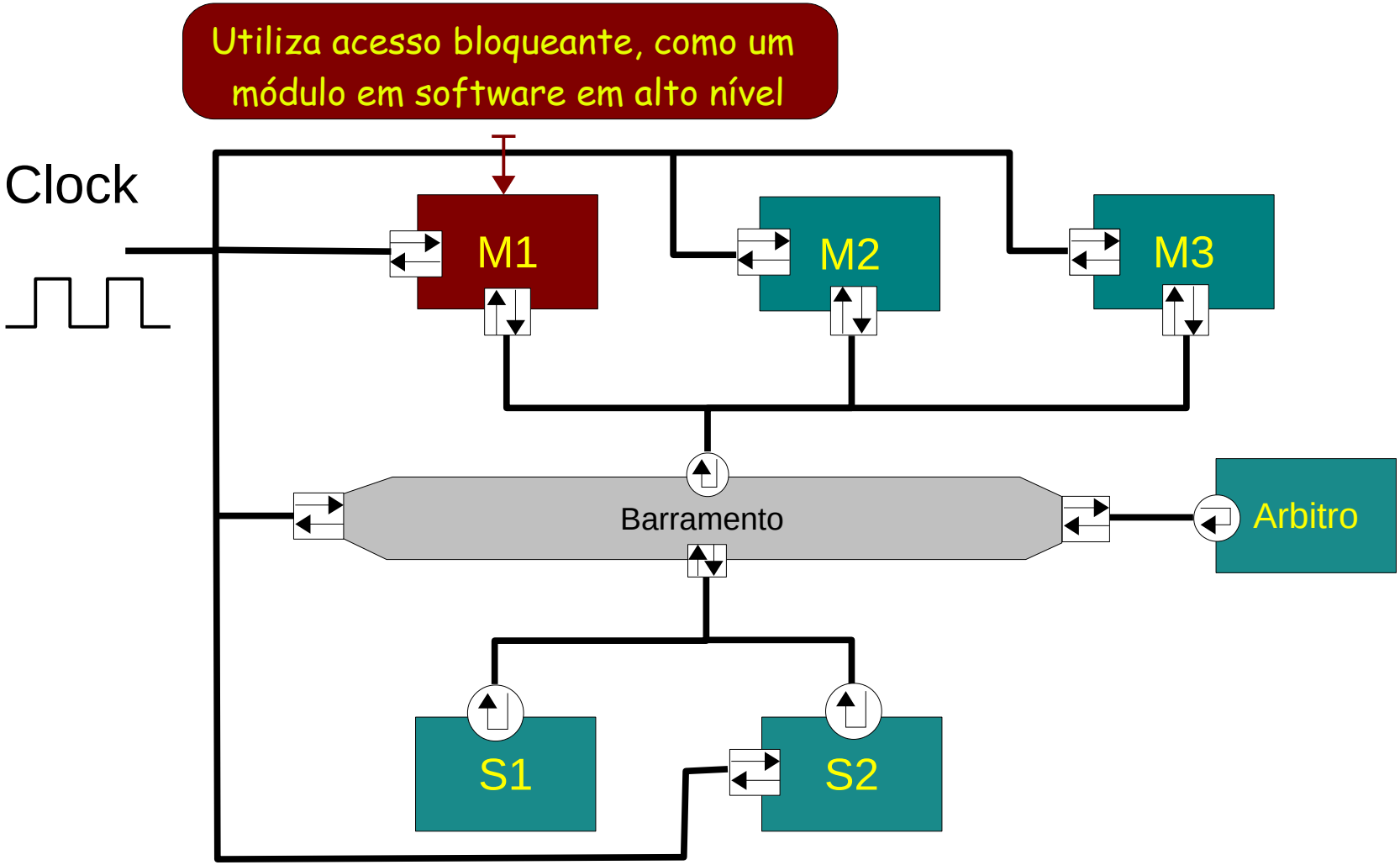
Estrutura do Barramento ...



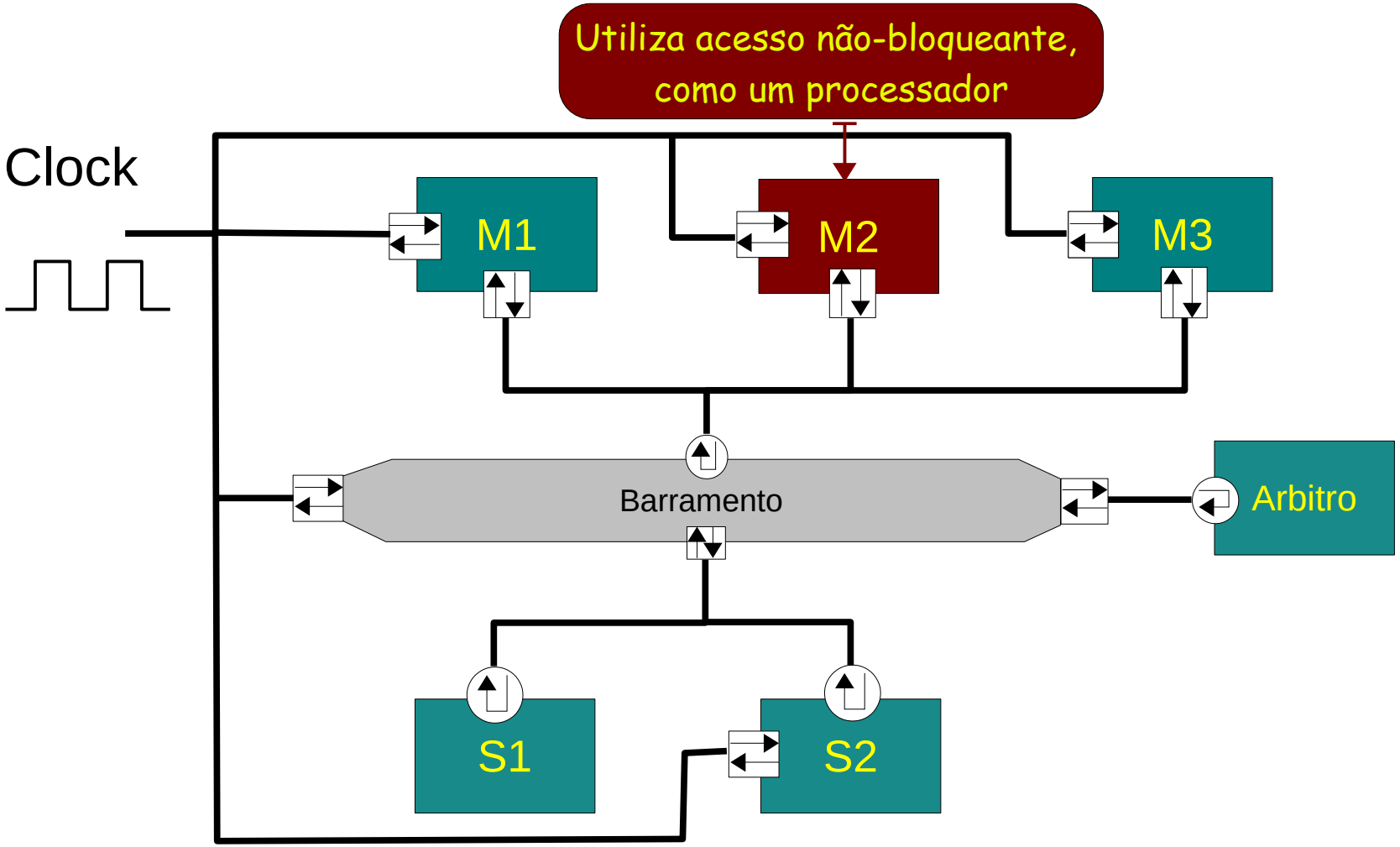
Estrutura do Barramento ...



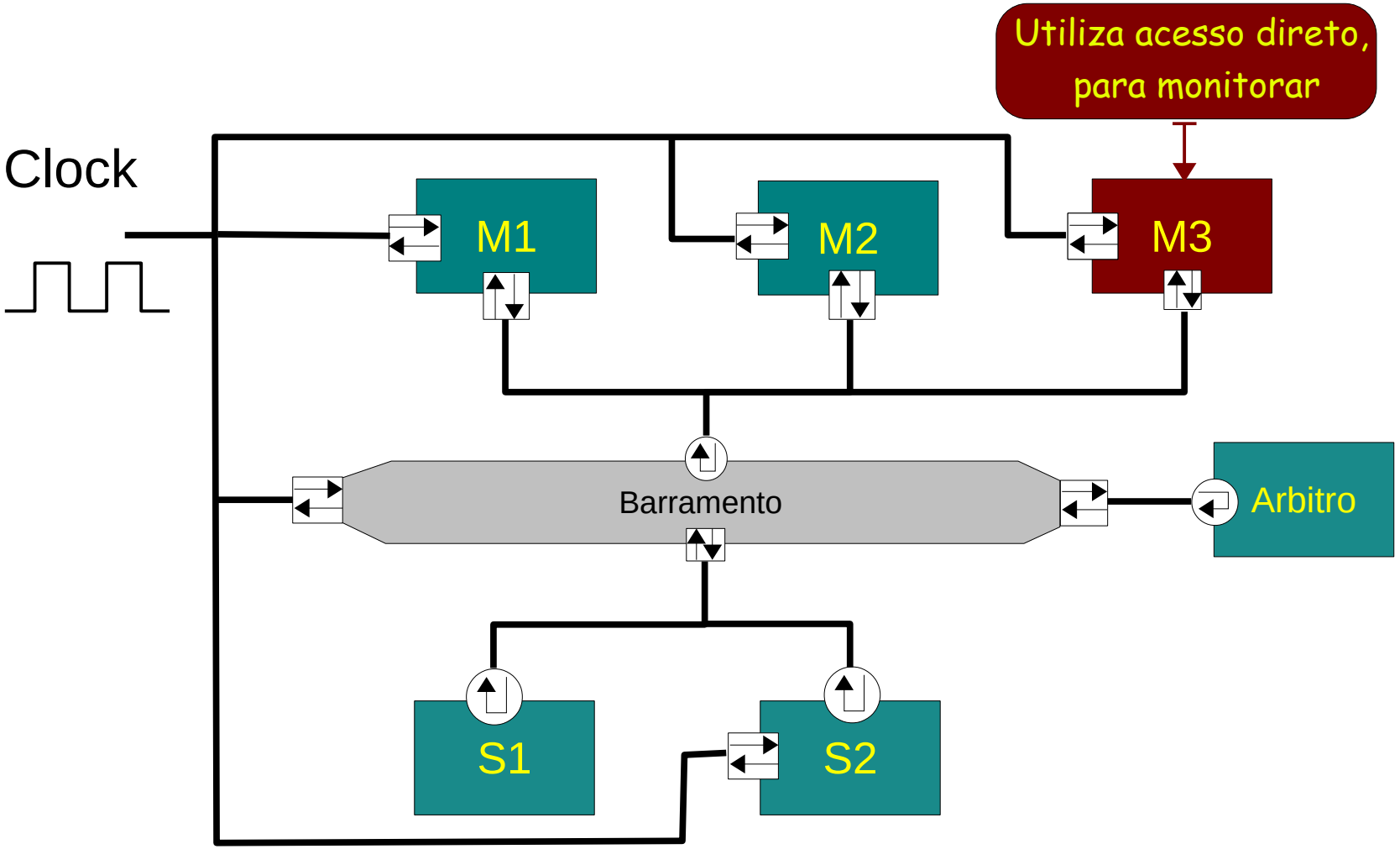
Primeiro Mestre



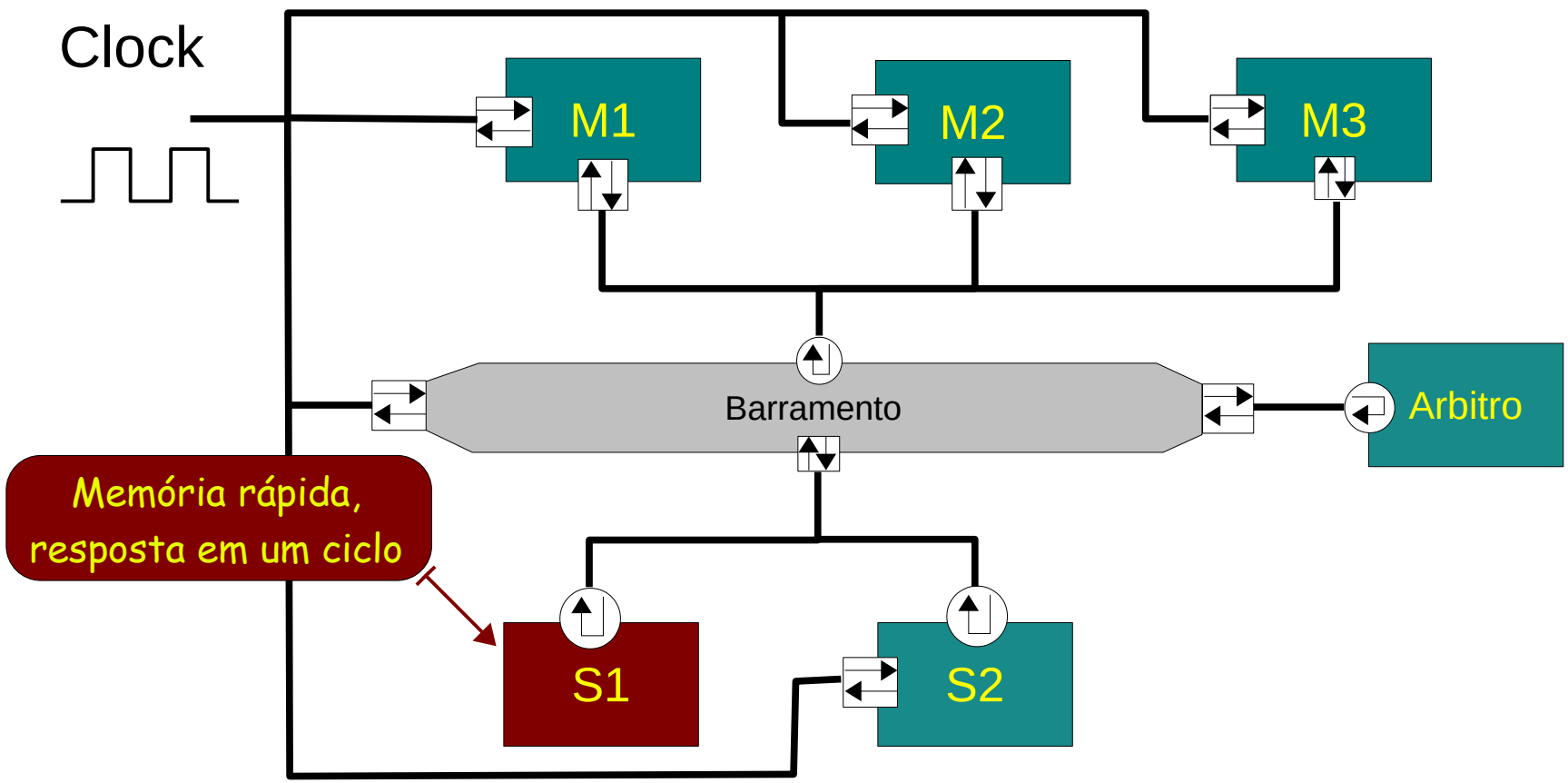
Segundo Mestre



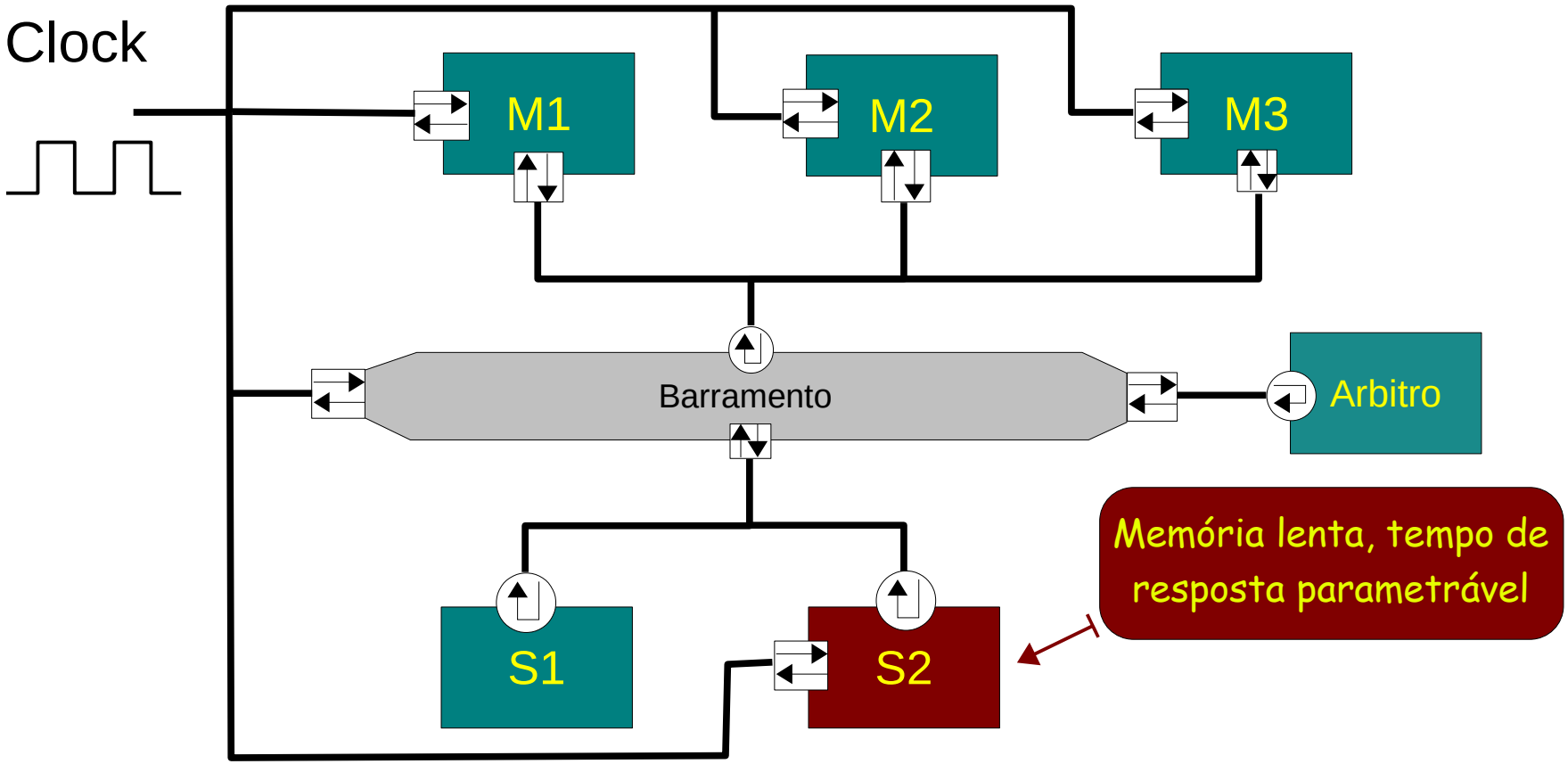
Terceiro Mestre



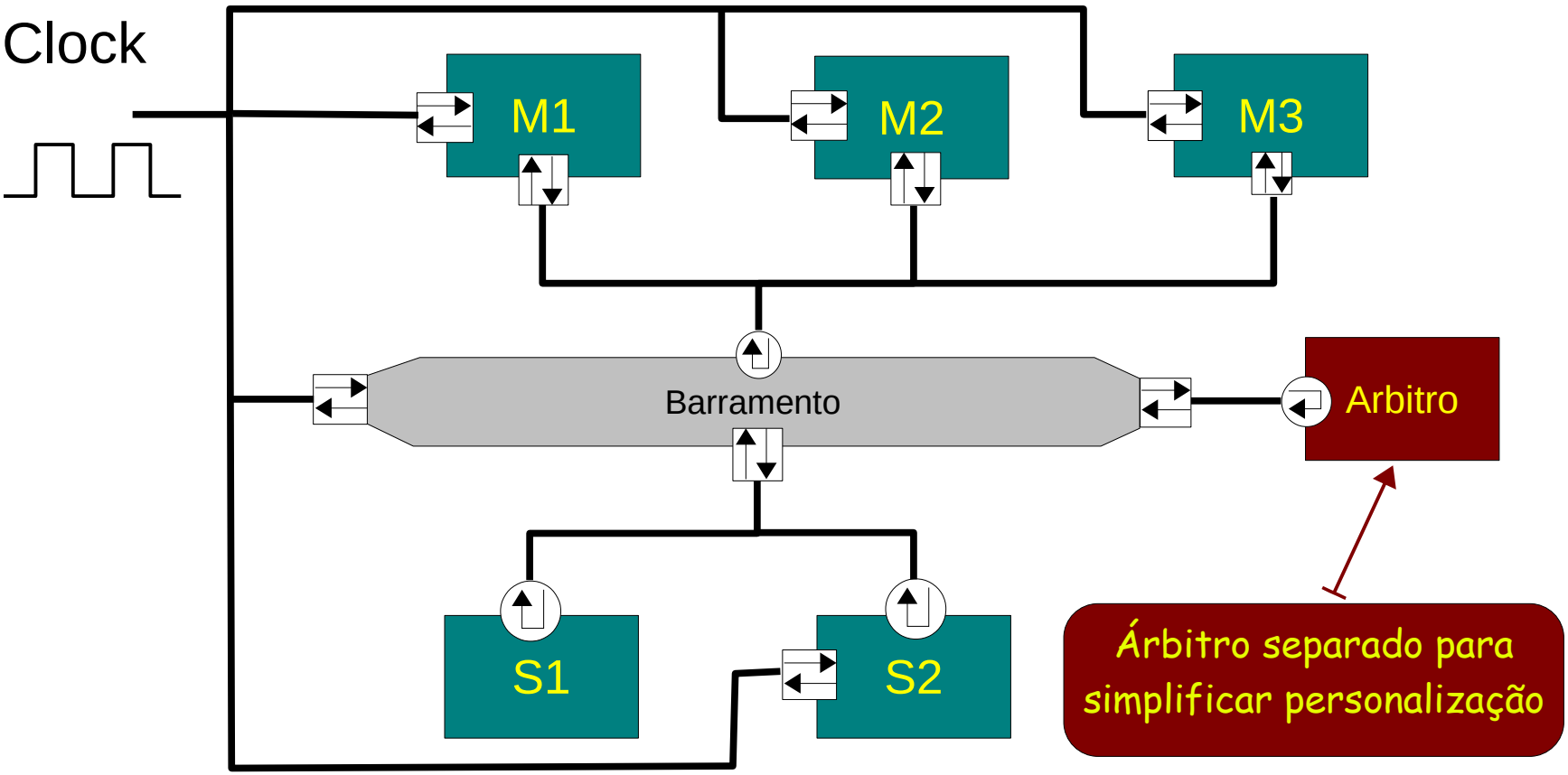
Primeiro Escravo



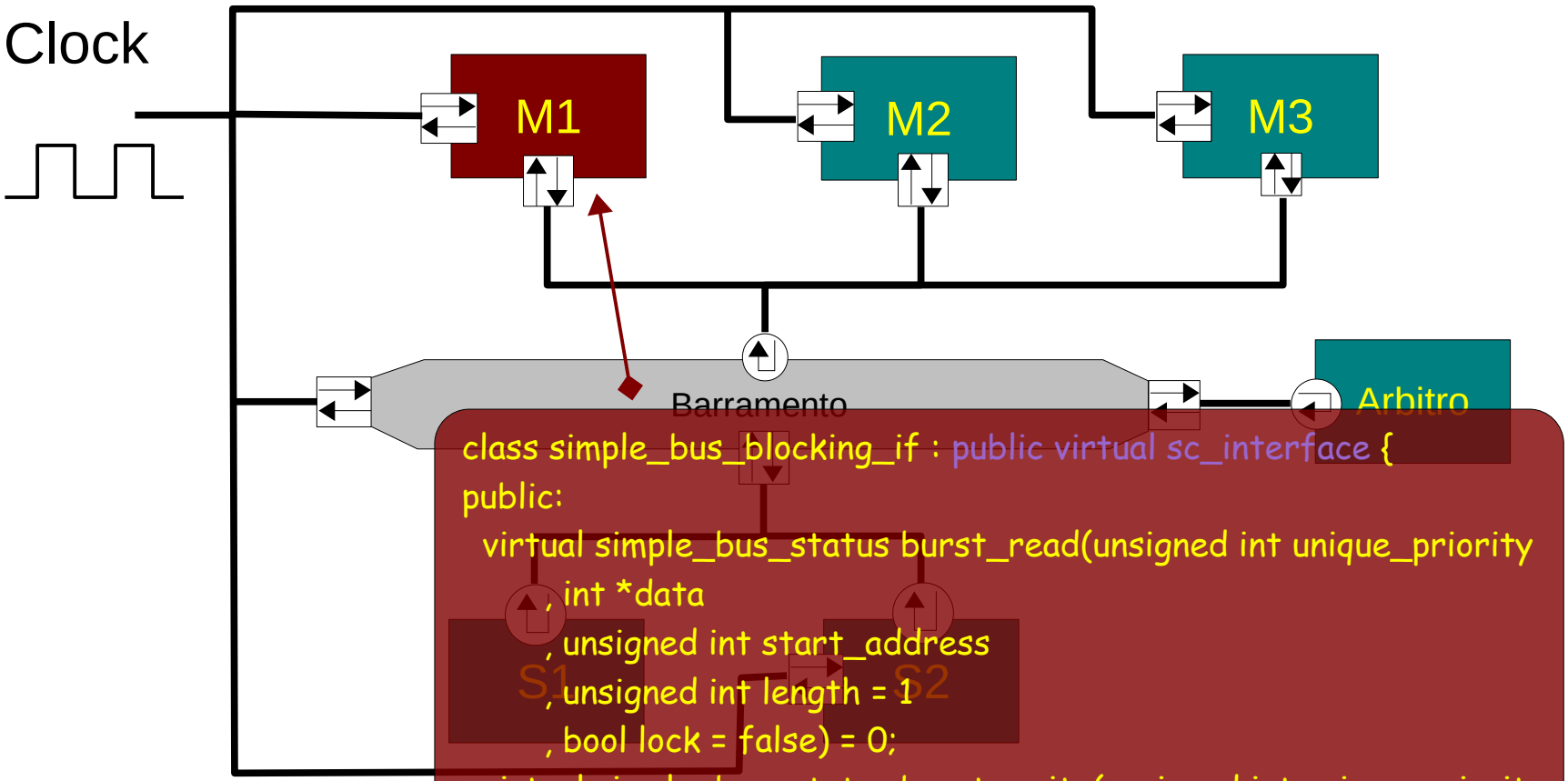
Segundo Escravo



Terceiro Escravo

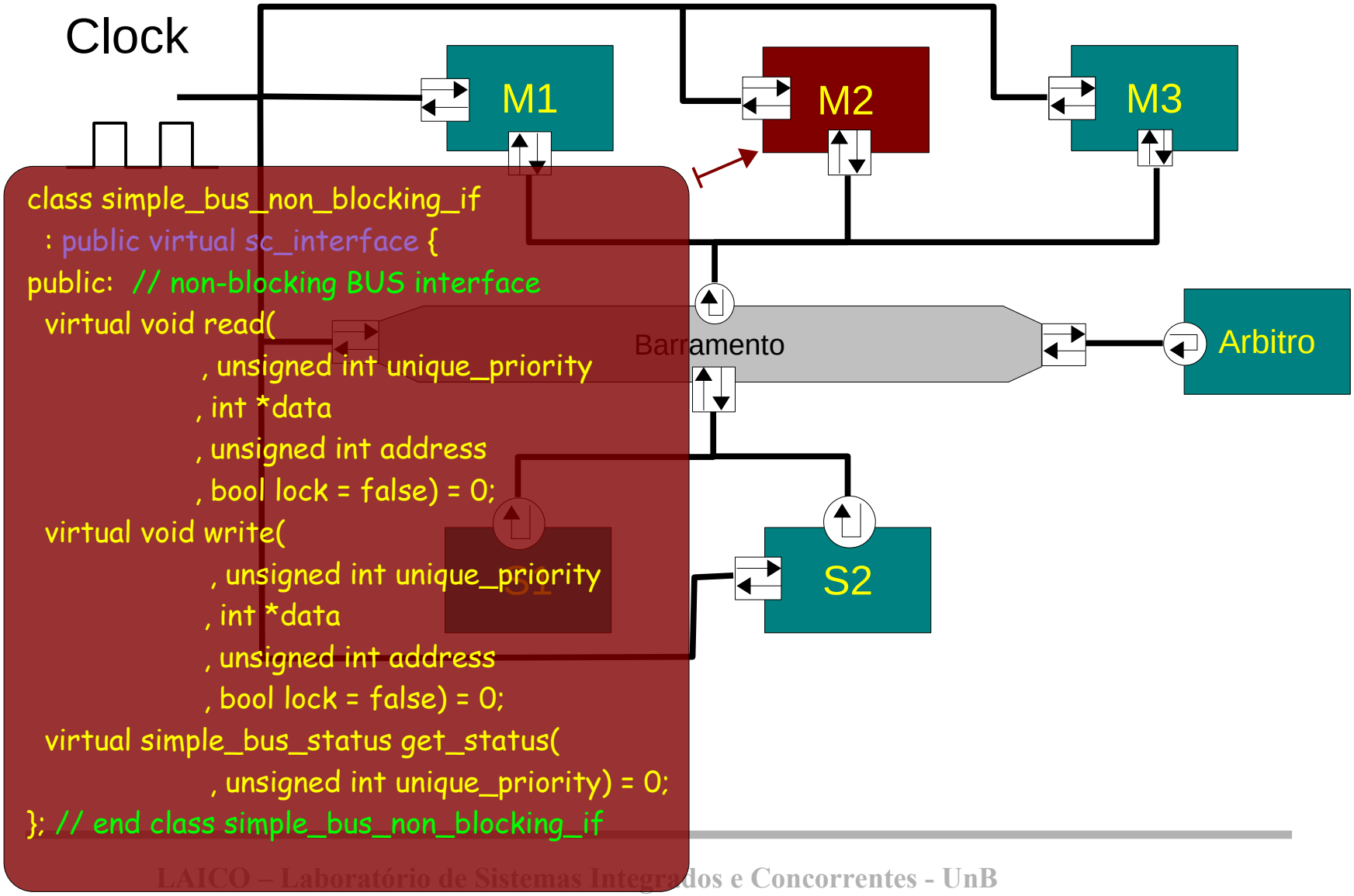


Interface Bloqueante



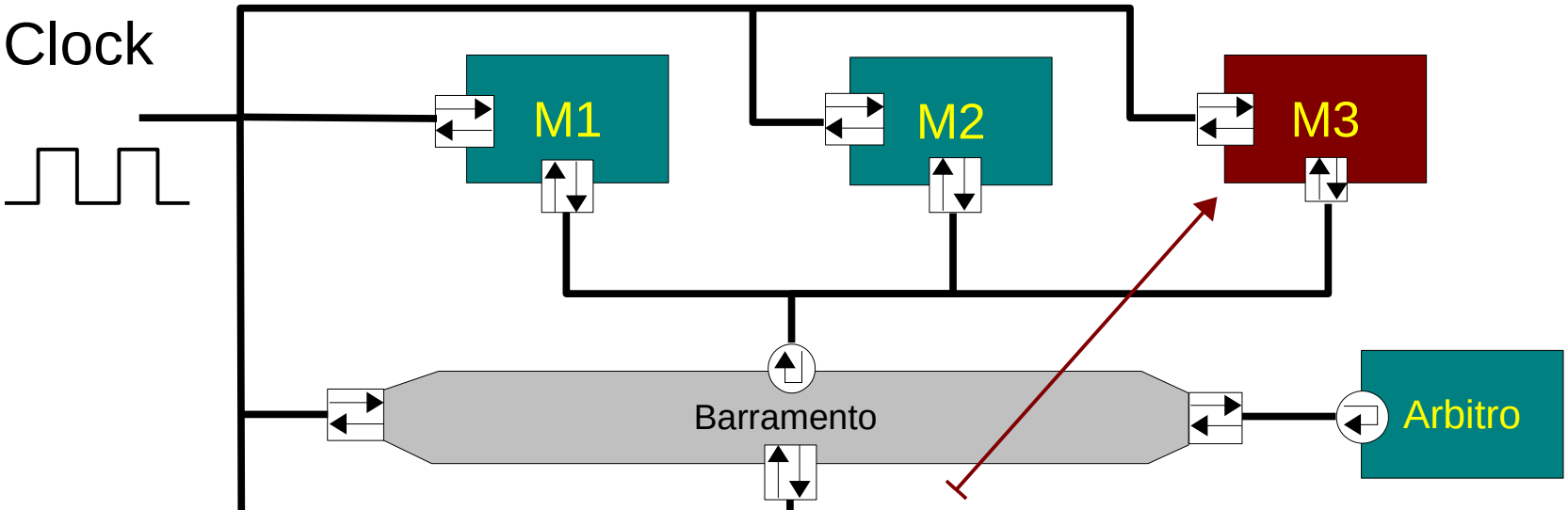

```
class simple_bus_blocking_if : public virtual sc_interface {
public:
  virtual simple_bus_status burst_read(unsigned int unique_priority
      , int *data
      , unsigned int start_address
      , unsigned int length = 1
      , bool lock = false) = 0;
  virtual simple_bus_status burst_write(unsigned int unique_priority
      , int *data
      , unsigned int start_address
      , unsigned int length = 1
      , bool lock = false) = 0;
};
```

Interface Não-Bloqueante




```
class simple_bus_non_blocking_if
  : public virtual sc_interface {
public:  // non-blocking BUS interface
  virtual void read(
                  , unsigned int unique_priority
                  , int *data
                  , unsigned int address
                  , bool lock = false) = 0;
  virtual void write(
                  , unsigned int unique_priority
                  , int *data
                  , unsigned int address
                  , bool lock = false) = 0;
  virtual simple_bus_status get_status(
                  , unsigned int unique_priority) = 0;
}; // end class simple_bus_non_blocking_if
```

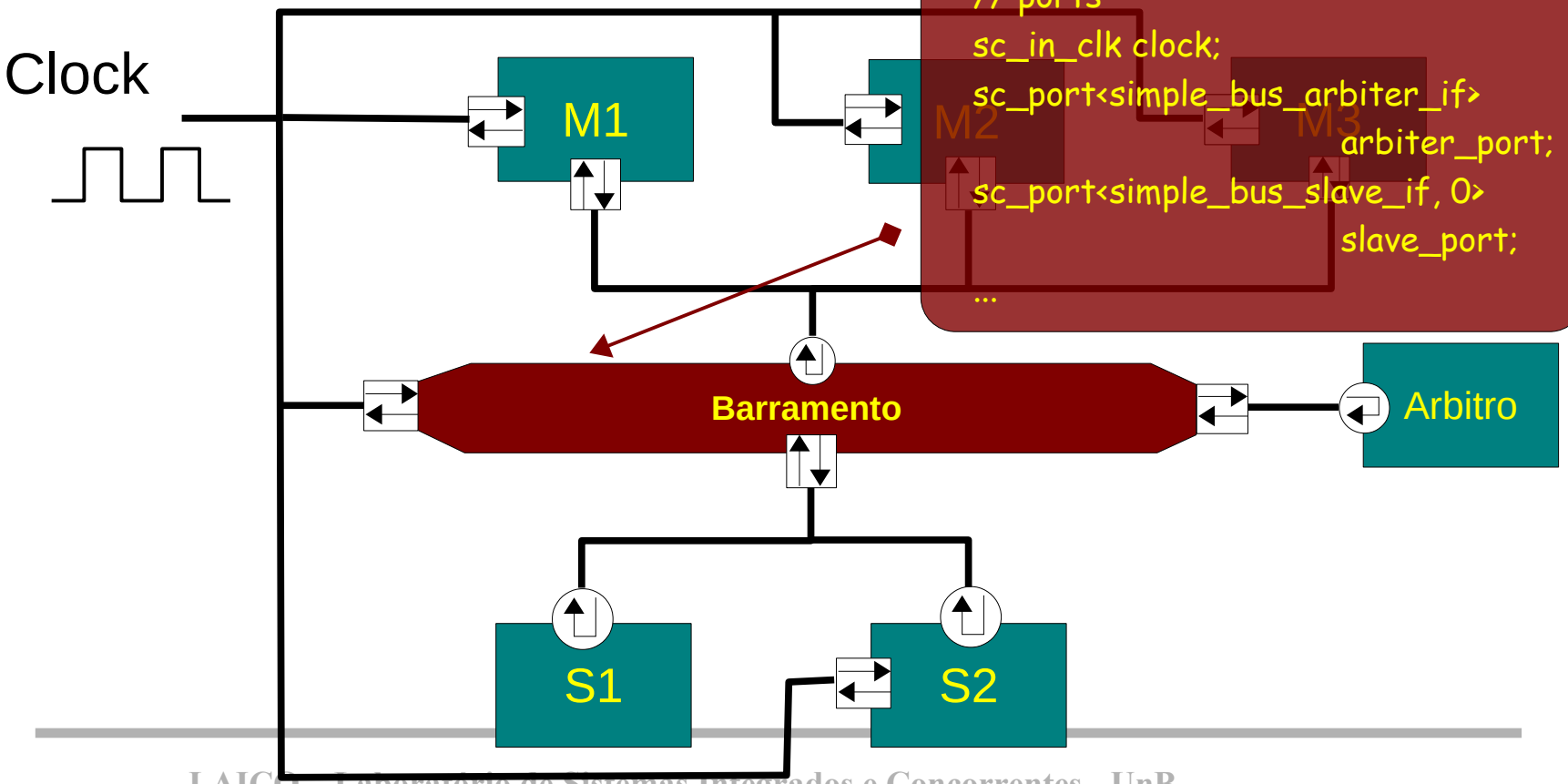
Interface Direta



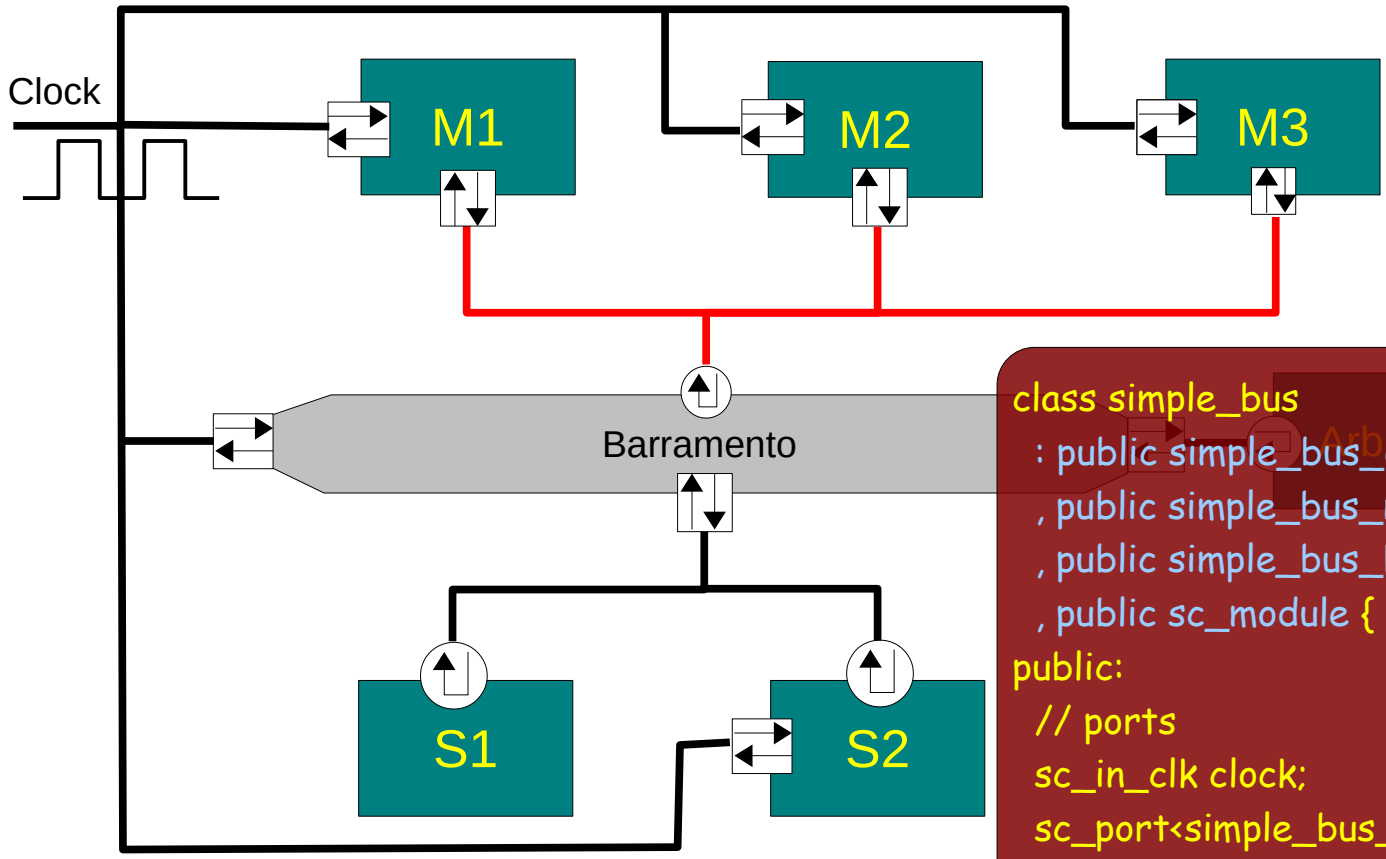

```
class simple_bus_direct_if
  : public virtual sc_interface
{
public:
  // direct BUS/Slave interface
  virtual bool direct_read(int *data, unsigned int address) = 0;
  virtual bool direct_write(int *data, unsigned int address) = 0;

}; // end class simple_bus_direct_if
```

Barramento Implementa Interfaces

```
class simple_bus
  : public simple_bus_direct_if
  , public simple_bus_non_blocking_if
  , public simple_bus_blocking_if
  , public sc_module {
public:
  // ports
  sc_in_clk clock;
  sc_port<simple_bus_arbiter_if>
                        arbiter_port;
  sc_port<simple_bus_slave_if, 0>
                        slave_port;
  ...
```

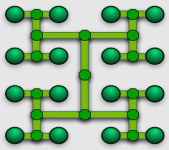


Na Subida do Relógio as Requisições




```
class simple_bus
  : public simple_bus_direct_if
  , public simple_bus_non_blocking_if
  , public simple_bus_blocking_if
  , public sc_module {
public:
  // ports
  sc_in_clk clock;
  sc_port<simple_bus_arbiter_if>
                          arbiter_port;
  sc_port<simple_bus_slave_if, 0>
                          slave_port;
  ...
```

Universidade de Brasília

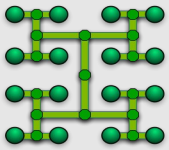


LAICO

Modelagem para Alto Desempenho

- Baseada em *transações*:
  - chamada de método onde os dados e controle são passados para outro processo
- Tipos de dados de alto nível:
  - sempre que possível tipos primitivos de C++
- Uso de ponteiros para transmissão de dados
- Uso de sentividade dinâmica para eliminar acionamento desnecessário de processos

Universidade de Brasília

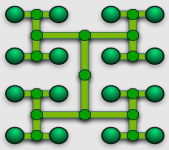


LAICO

Modelagem para Alto Desempenho...

- Alguns módulos não tem nenhum processo
  - fast_mem, arbiter
  - muito simples, implementam métodos usuais C++
- Preferir SC_METHOD no lugar de SC_THREAD
  - blocos ativados mais frequentemente (*bus*, *arbiter*, *fast_mem* e *slow_mem*) usam SC_METHOD ou métodos comuns
- Simplificar os processos ativados frequentemente

Universidade de Brasília



LAICO

Funcionamento

- Na subida do relógio os mestres enviam requisições ao barramento, onde são armazenadas
- O barramento mantém todas as requisições ainda não completadas
- Na descida do relógio, o barramento chama o árbitro para selecionar uma requisição a ser atendida
- O barramento a seguir determina qual escravo está sendo acessado em função do endereço

Funcionamento

- O escravo é acessado pelo barramento usando o método `read()/write()`
- O método retorna imediamente indicando se o escravo emitiu um *wait state*
- Havendo *wait states*, o barramento impede outros acessos, reeditando a requisição incompleta no próximo ciclo
- Quando a transferência é completada, o estado da requisição original é atualizado

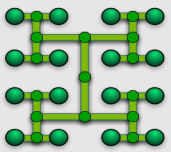
Funcionamento

- Se a requisição é tipo *burst (rajada)*, ela é mantida para ciclos subsequentes
- Acessos em rajada podem ser interrompidos por acessos de maior prioridade, a menos que o *lock* esteja ativado
  - Neste caso, a rajada não é interrompida
  - Se o mesmo mestre está realizando um novo acesso depois de um *lock*, ele é sempre selecionado

Universidade de Brasília

Simple_bus_blocking_if

```
class simple_bus_blocking_if
  : public virtual sc_interface
{
public:
  // blocking BUS interface
  virtual simple_bus_status burst_read(unsigned int unique_priority
        , int *data
        , unsigned int start_address
        , unsigned int length = 1
        , bool lock = false) = 0;
  virtual simple_bus_status burst_write(unsigned int unique_priority
, int *data
, unsigned int start_address
, unsigned int length = 1
, bool lock = false) = 0;

}; // end class simple_bus_blocking_if
```



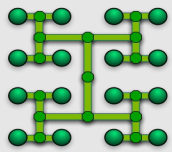
LAICO

#endif

Universidade de Brasília

Simple_bus_non_blocking_if

```
class simple_bus_non_blocking_if
  : public virtual sc_interface
{
public:
  // non-blocking BUS interface
  virtual void read(unsigned int unique_priority
    , int *data
    , unsigned int address
    , bool lock = false) = 0;
  virtual void write(unsigned int unique_priority
     , int *data
     , unsigned int address
     , bool lock = false) = 0;

  virtual simple_bus_status get_status(unsigned int unique_priority) = 0;

};
```

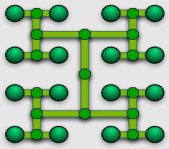


LAICO

Universidade de Brasília

Simple_bus_direct_if

```
class simple_bus_direct_if
  : public virtual sc_interface
{
public:
  // direct BUS/Slave interface
  virtual bool direct_read(int *data, unsigned int address) = 0;
  virtual bool direct_write(int *data, unsigned int address) = 0;

};
```

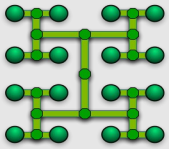


LAICO

Universidade de Brasília

Simple_bus_slave_if

```
class simple_bus_slave_if
  : public simple_bus_direct_if
{
public:
  // Slave interface
  virtual simple_bus_status read(int *data, unsigned int address) = 0;
  virtual simple_bus_status write(int *data, unsigned int address) = 0;

  virtual unsigned int start_address() const = 0;
  virtual unsigned int end_address() const = 0;

};
```

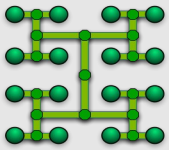


LAICO

Universidade de Brasília

Simple_bus_arbiter_if

```
class simple_bus_arbiter_if
  : public virtual sc_interface
{
public:
  virtual simple_bus_request *
    arbitrate(const simple_bus_request_vec &requests) = 0;

};
```



LAICO