Aulas 12 e 13

- Barramentos paralelo *vs* barramentos série
- Barramentos série
  - Princípio de funcionamento
  - Sincronização de relógio entre transmissor e recetor
  - Modos de transmissão de dados: transmissão orientada ao bit, transmissão orientada ao byte
  - Topologias de ligação
  - Elementos de uma ligação série

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Introdução

- Barramentos: interligação dos blocos de um sistema de computação
  - CPU, memória, unidades de I/O
- Tipos de dispositivos ligados a um barramento:
  - **Bus Master** – Dispositivo que pode iniciar e controlar uma transferência de dados (exemplos: Processador, Módulo de I/O com DMA)
  - **Bus Slave** – Dispositivo que só responde a pedidos de transferências de dados, i.e., não tem capacidade para iniciar uma transferência (exemplos: Memória, Módulo de I/O sem DMA)
- **Barramento de um só Master**: só há um dispositivo no barramento com capacidade para iniciar e controlar transferências de informação
- **Barramento Multi-Master**: mais que um dispositivo capaz de iniciar e controlar transferências de informação (exemplos: vários CPUs, 1 ou mais controladores de DMA, um ou mais módulos de I/O com DMA)

Introdução

- **Barramentos paralelo**: os dados são transmitidos em paralelo (através de N Linhas). Incluem:

- **Barramento de dados**:
  - Suporta a transferência de informação entre os blocos
  - O número de linhas (largura do barramento) determina quantos bits podem ser transferidos simultaneamente; a largura do barramento é um fator determinante no desempenho do sistema
- **Barramento de endereços**:
  - Especifica a origem/destino da informação
  - O número de linhas define a dimensão do espaço de endereçamento (determina a capacidade máxima de memória que o sistema pode ter: 2^N palavras, sendo N o número de bits do barramento de endereços)
- **Barramento de controlo**:
  - Conjunto de sinais que especificam operações, sinalizam eventos, efetuam pedidos, …

Introdução

- A transmissão paralela, com relógio comum, a débitos elevados coloca problemas de vária ordem, nomeadamente:
  - Controlo do tempo de "skew" das linhas do barramento
  - Dificuldade em anular a interferência provocada por fontes de ruído externas
  - Interferência mútua, isto é, entre sinais adjacentes ("crosstalk")
  - Elevado número de fios de ligação e custo associado
  - Fichas de ligação volumosas e caras (possivelmente com contactos dourados)

- **Barramentos série**: os dados são serializados no transmissor, ou seja, transmite-se 1 bit de cada vez (tipicamente 1 bit a cada ciclo de relógio)
  - Comunicação série

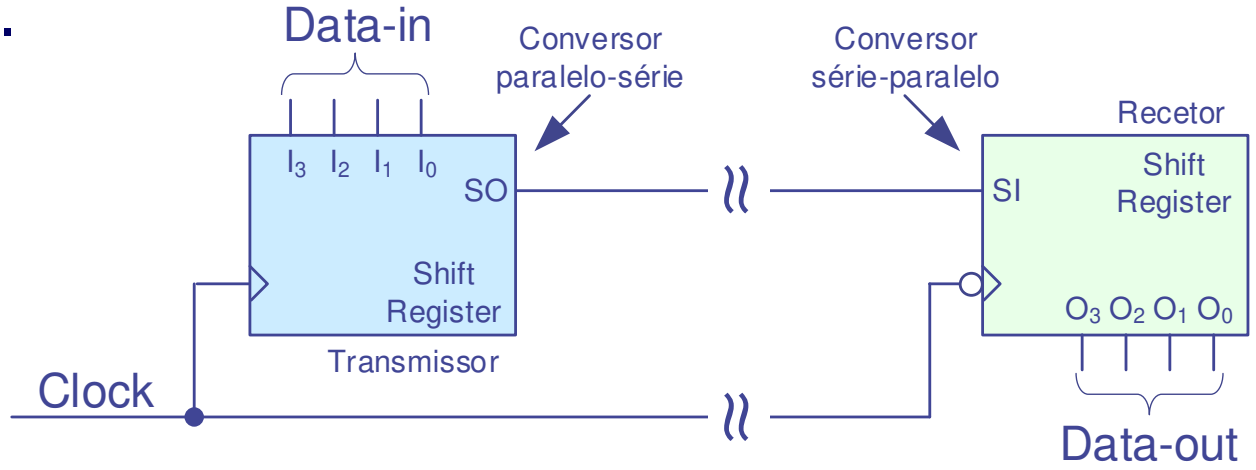
Introdução

- Vantagens dos barramentos série (ao nível físico):
  - Simplicidade de ligação de cablagem
  - Diminuição de custos de interligação
  - Possibilidade de transmissão a distâncias elevadas (em par diferencial)
  - Débito elevado

- Tipos de comunicação série
  - **Simplex**: comunicação apenas num sentido (TX -> RX); usada, por exemplo, em telemetria, para leitura remota de sensores
  - **Half-duplex**: comunicação nos dois sentidos, mas apenas um de cada vez (é usada uma só linha)
  - **Full-duplex**: Comunicação simultânea nos dois sentidos (são usadas duas linhas)

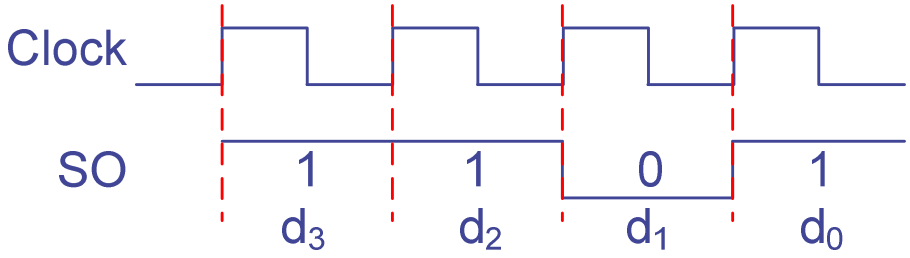
Introdução

- Diz-se que se está na presença de um barramento ou interface série sempre que exista uma só "linha" (suporte) para transferência de dados.



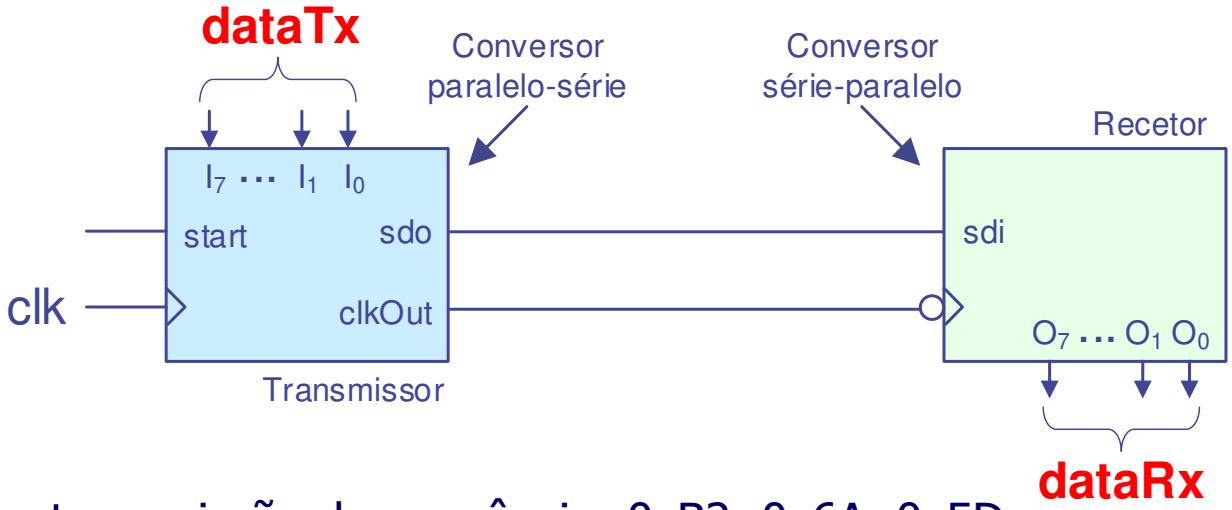
Para transmissão bidirecional podem existir 2 "linhas" separadas, uma para transmissão e outra para receção

- Exemplo (Data-in = 1101)

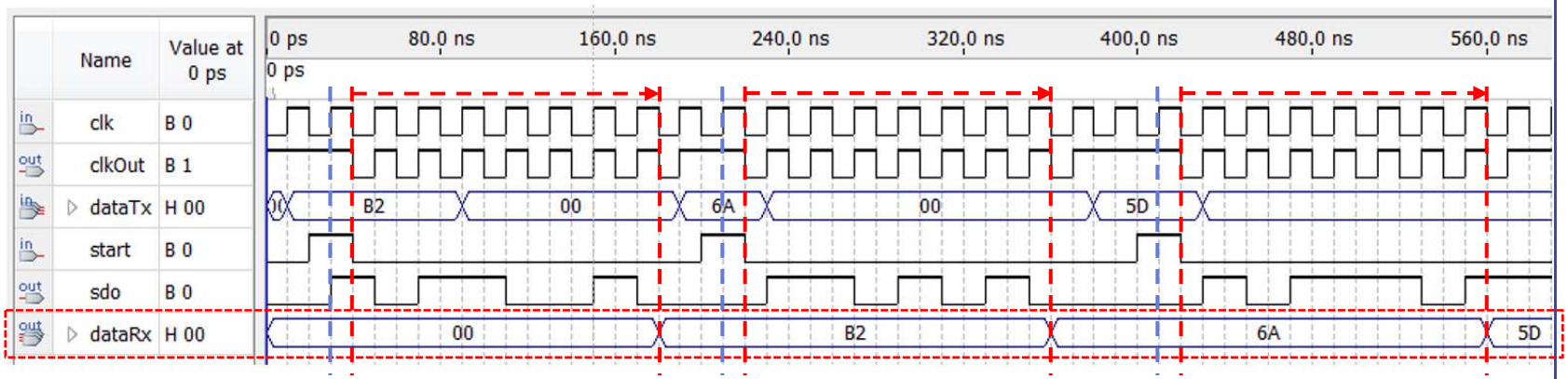


Introdução

- Exemplo em que o transmissor gera o sinal de relógio (o sinal "start" dá início à transmissão)

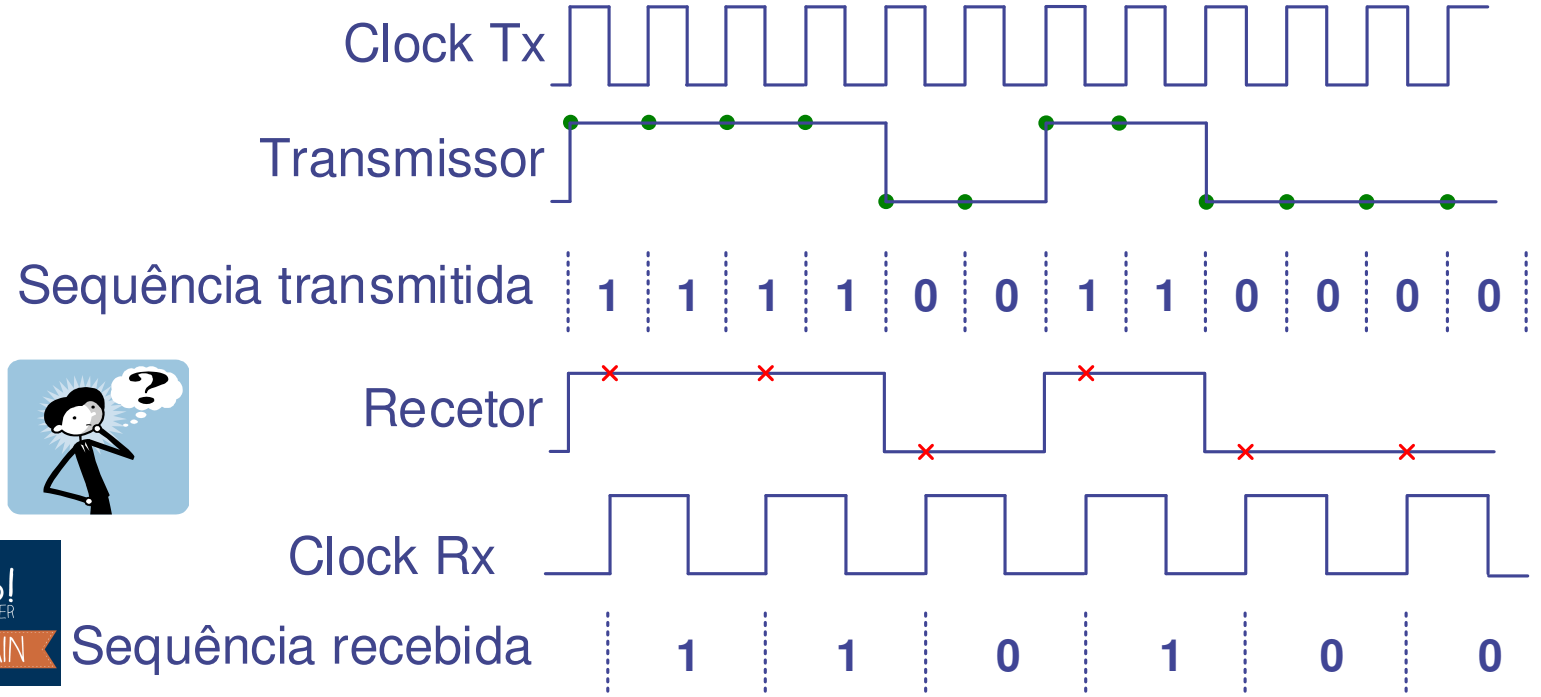


- Exemplo com transmissão da sequência: 0xB2, 0x6A, 0x5D



Sincronização entre transmissor e recetor

- O sincronismo é obtido através da utilização do mesmo relógio no transmissor e no recetor, ou de relógios independentes que terão que estar sincronizados durante a transmissão



- Caso sejam distintos, os relógios do Tx e do Rx têm de estar sincronizados para que a amostragem do sinal seja realizada nos instantes corretos

Sincronização entre transmissor e recetor

- **Transmissão Síncrona**
  - O sinal de relógio é transmitido de forma explícita através de um sinal adicional, ou na codificação dos dados
  - Os relógios do transmissor e do recetor têm de se manter sincronizados
  - Quando o relógio não é explicitamente transmitido, o relógio do recetor é recuperado a partir das transições de nível lógico na linha de dados

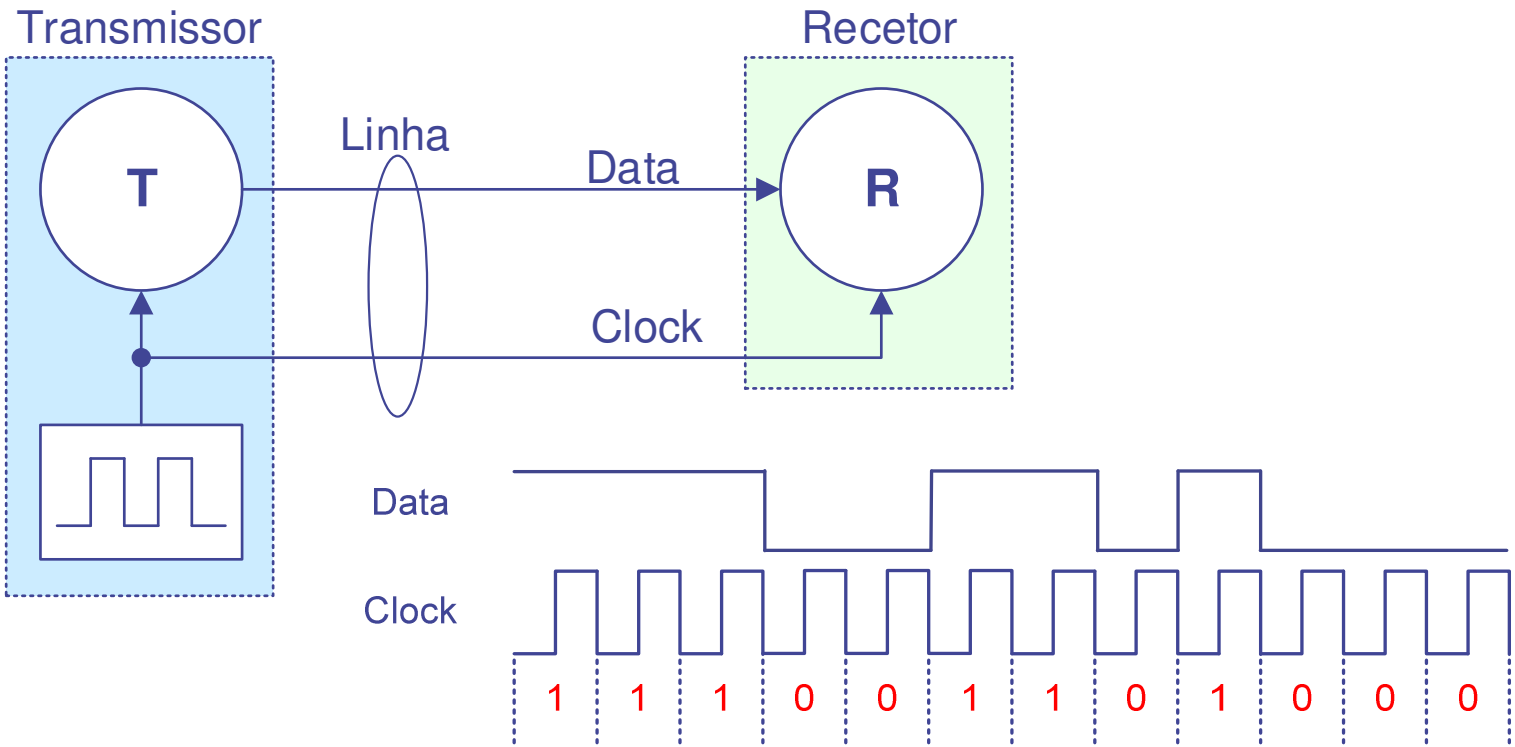
- **Transmissão Assíncrona**
  - Não é usado relógio na transmissão, nem há recuperação do relógio na receção
  - É necessário acrescentar bits para sinalizar o princípio e o fim da transmissão (e.g. start bit, stop bit), que permitam ao recetor proceder à amostragem do sinal recebido, com o menor erro temporal possível

Técnicas de sincronização do relógio

- Transmissão síncrona
  - **Relógio explícito do transmissor**
    - Exemplo: SPI
  - **Relógio explícito do recetor**
  - **Relógio explícito mutuamente-sincronizado**
    - Exemplo: I2C
  - **Relógio codificado ("self-clocking")**
    - Exemplo: USB, Ethernet
- Transmissão assíncrona
  - **Relógio implícito**
    - Exemplo: RS-232, CAN

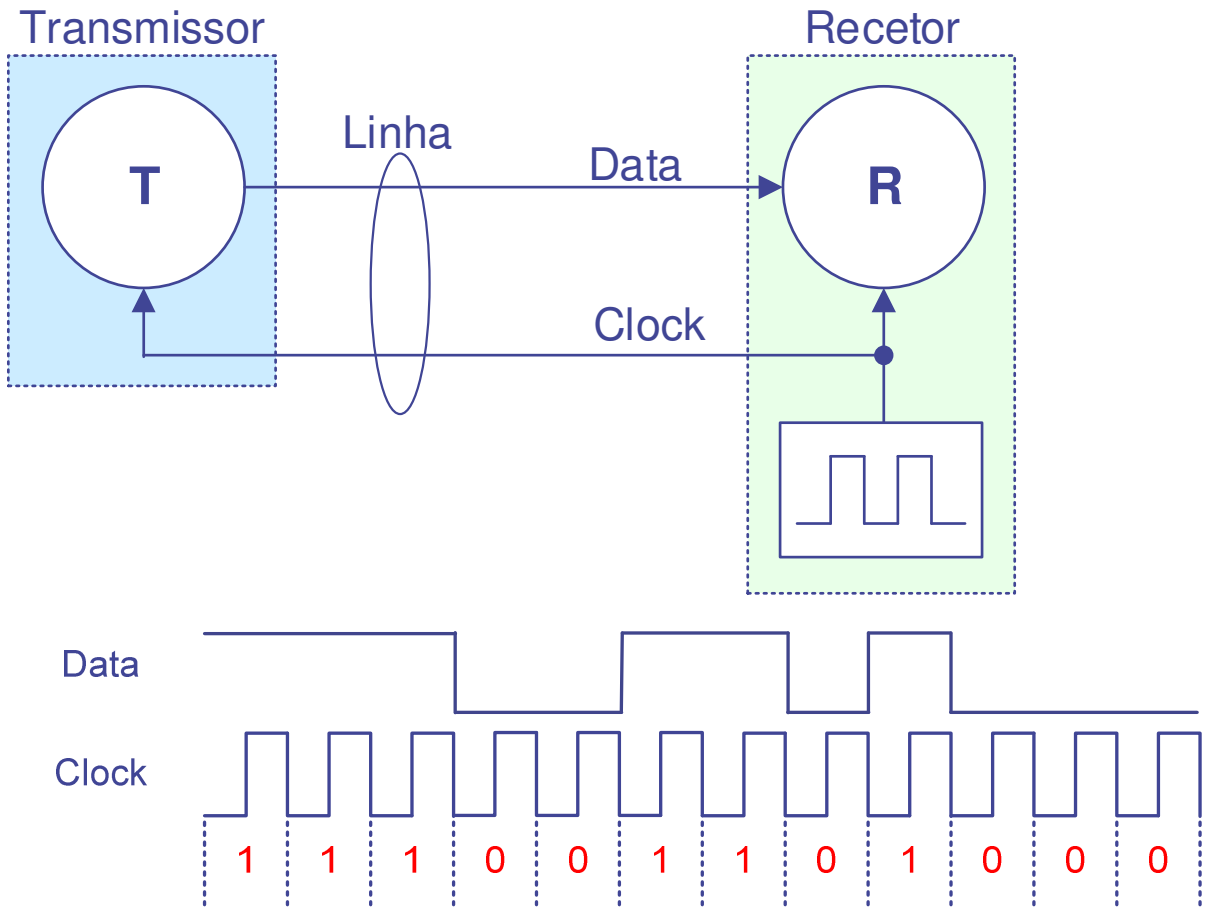
Sincronização de relógio

- **Relógio explícito do transmissor**
- O transmissor envia os dados e informação de relógio em linhas separadas
- A linha de relógio pode ser vista como um sinal "Valid"



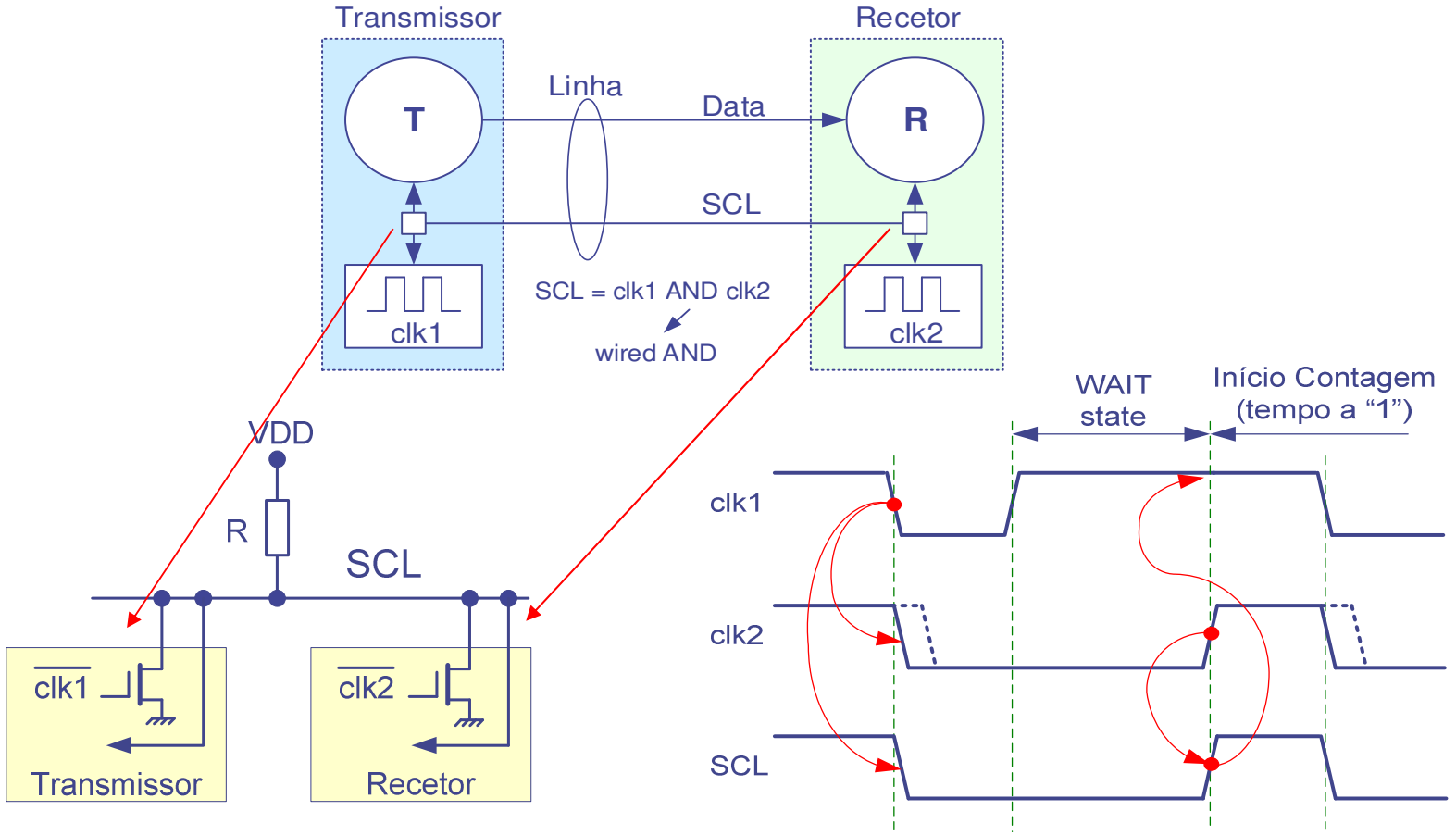
Sincronização de relógio

- **Relógio explícito do recetor**
- Semelhante ao anterior, sendo o recetor a gerar o sinal de relógio



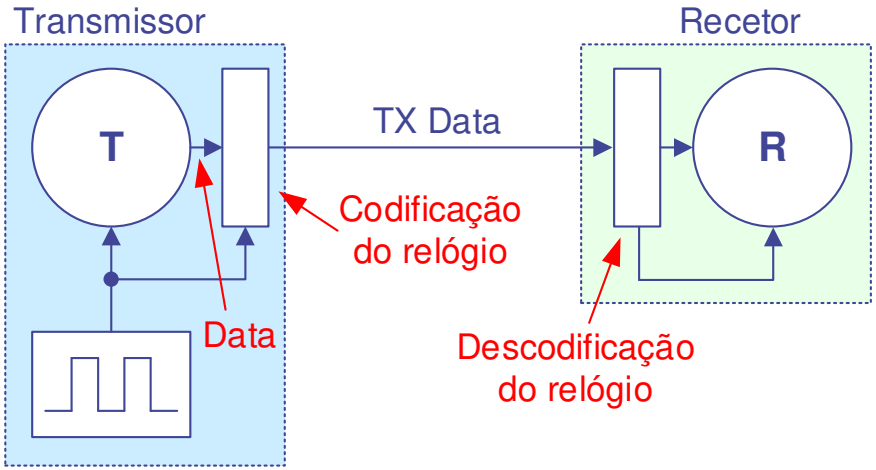
Sincronização de relógio

- **Relógio explícito mutuamente-sincronizado ("clock stretching")**
- Transmissor e Recetor sincronizam-se mutuamente

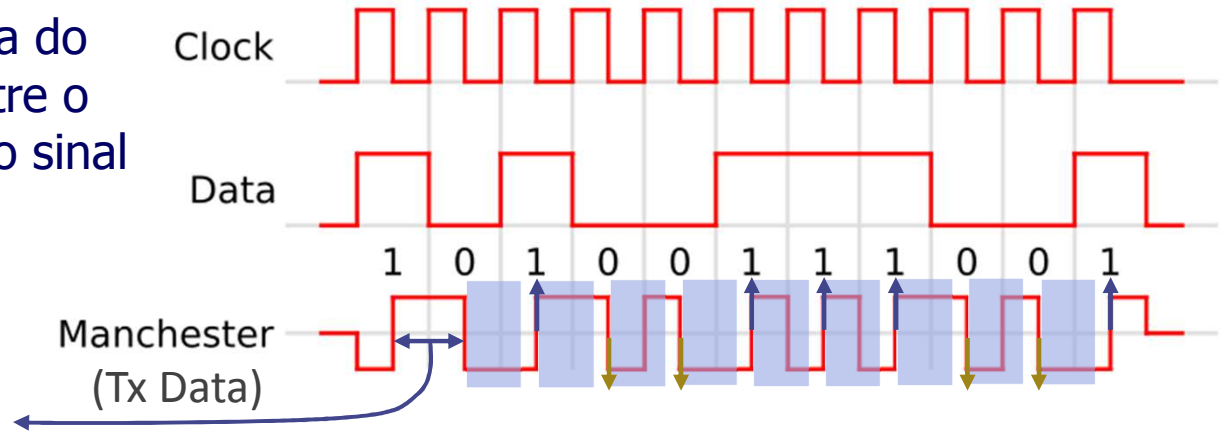


Sincronização de relógio

- **Relógio codificado**
- Relógio enviado, em forma codificada, conjuntamente com os dados

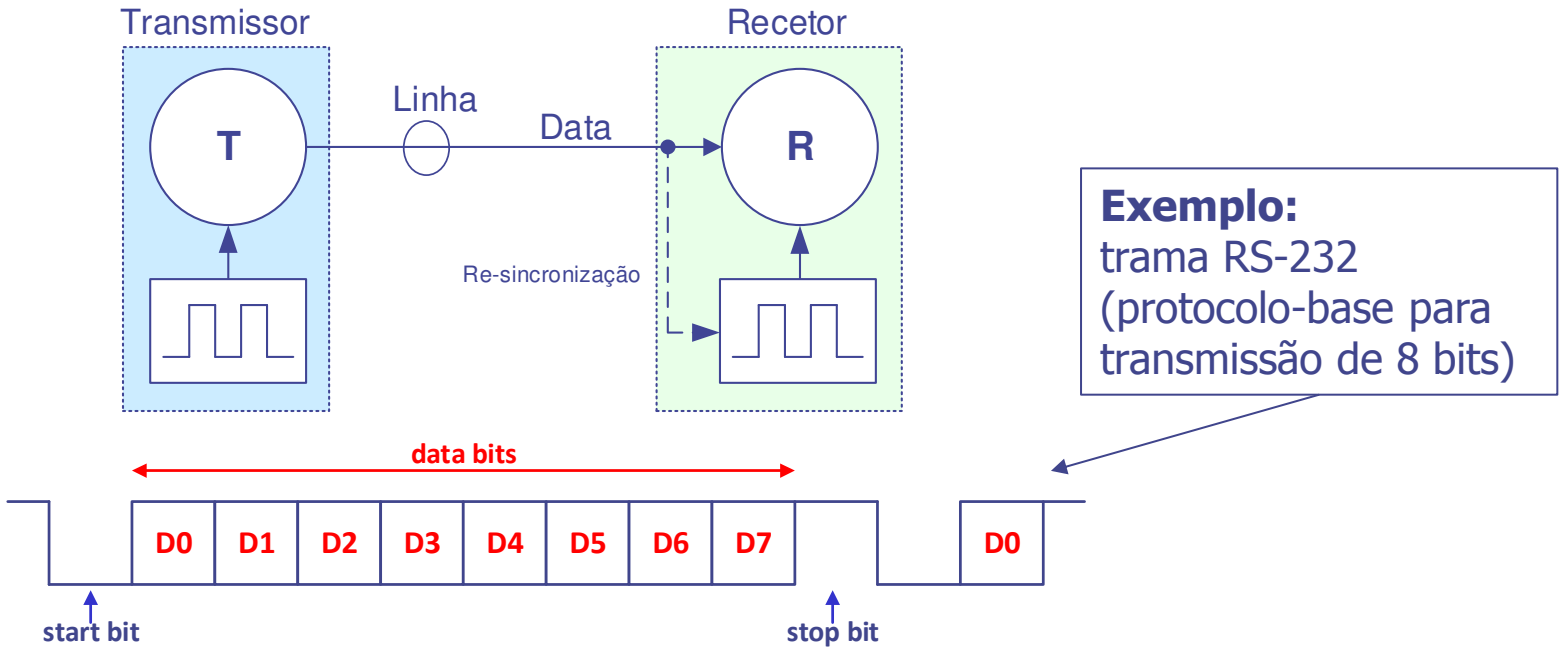


- Exemplo: codificação Manchester
- A informação transmitida resulta do "ou exclusivo" entre o bit a transmitir e o sinal de relógio



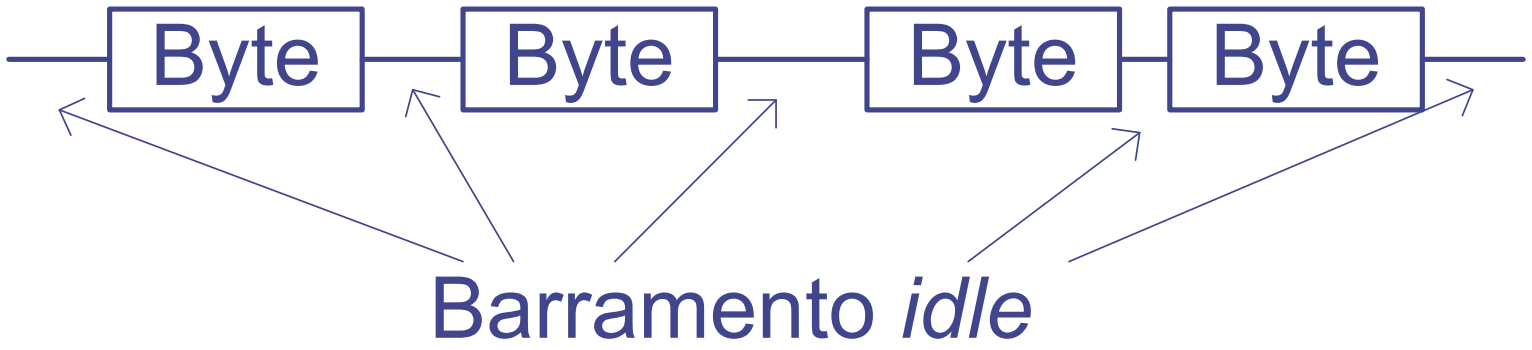
Sincronização de relógio (transmissão assíncrona)

- **Relógio implícito**
- Os relógios são locais (i.e. não há comunicação do relógio)
- O relógio do recetor é sincronizado ocasionalmente com o do transmissor por meio da receção de símbolos específicos
- Entre instantes de sincronização o desvio dos relógios depende da estabilidade/precisão dos relógios do transmissor e do recetor



Transmissão de dados – transmissão orientada ao Byte

- O envio de um byte é a operação atómica (indivisível) do barramento
- Cada byte é encarado como independente dos restantes
- Não há restrições temporais para a transmissão em sequência de 2 bytes



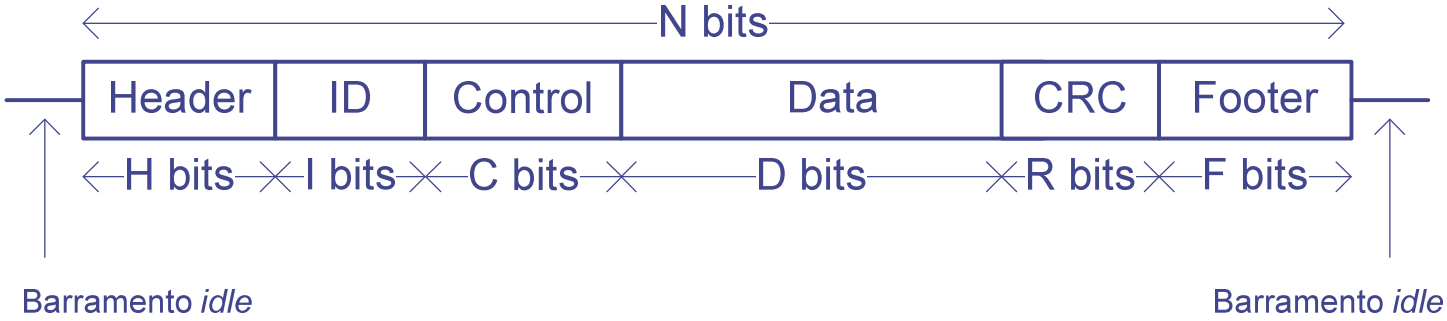
- Alguns bytes podem estar reservados para estruturar a informação
- Exemplo de transmissão orientada ao byte: RS232

Transmissão de dados – transmissão orientada ao bit

- A informação é organizada em tramas (sequência de bits intercalada entre duas situações de meio livre)
- As tramas são constituídas por um símbolo de sincronização (delimitador, constituído por 1 ou mais bits) seguido por uma sequência de bits de comprimento arbitrário
- As tramas podem conter campos com diferentes funções:
  - Sincronização: sinalização de início e de fim da trama
  - Arbitragem de acesso ao meio (em barramentos multi-master)
  - Identificação. Diversas formas possíveis:
    - Quem produz
    - Qual o destino
    - Identificação da informação que circula na trama
    - ...
  - Quantidade de informação transmitida
  - Dados
  - Deteção de erros de transmissão

Transmissão de dados – transmissão orientada ao bit

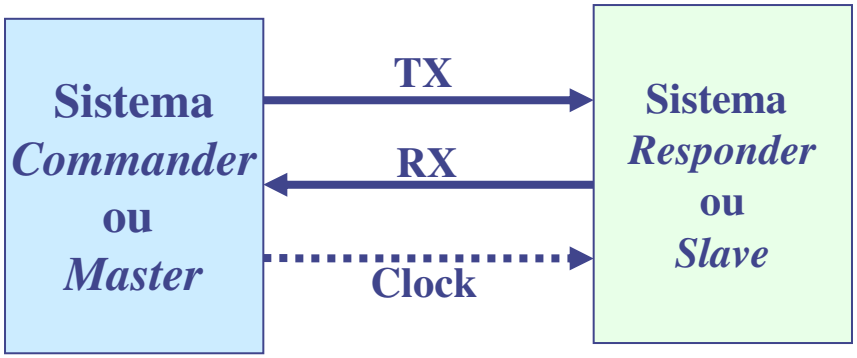
- Exemplo de estrutura de uma trama



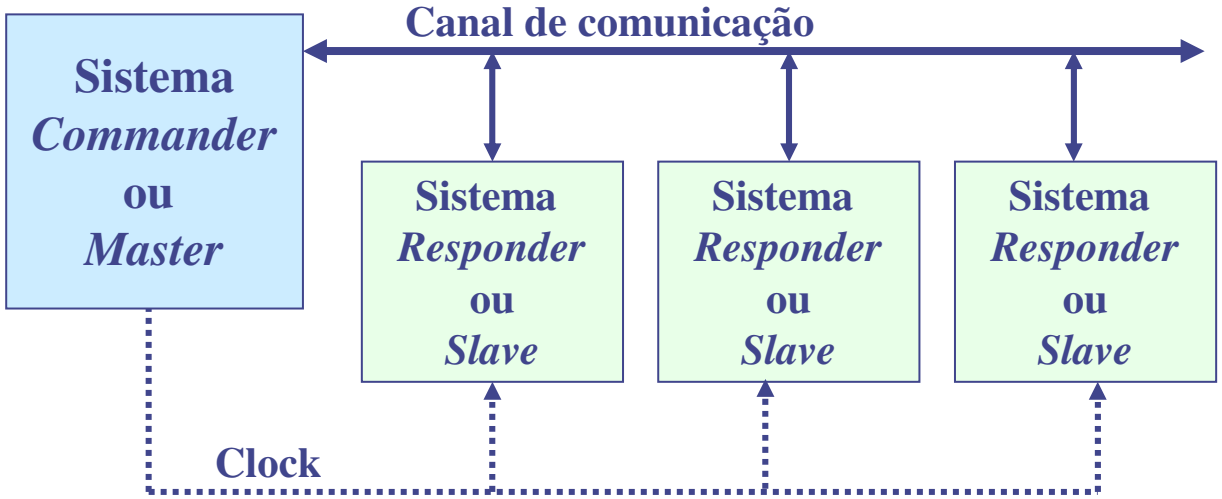
- Exemplo de transmissão orientada ao bit: barramento CAN ("Controller Area Network")
- "Header" e "footer": delimitadores de início e fim de trama
- Data: campo de dados
- CRC ("cyclic redundancy check"): código usado para detetar, no recetor, erros na comunicação
  - Uma forma simples de CRC consiste em somar todos os bytes transmitidos (soma truncada com R bits) – *checksum*
  - O recetor soma todos os bytes recebidos e compara com a soma recebida

Topologias

- Comunicação ponto a ponto ("half-duplex" ou "full-duplex"):

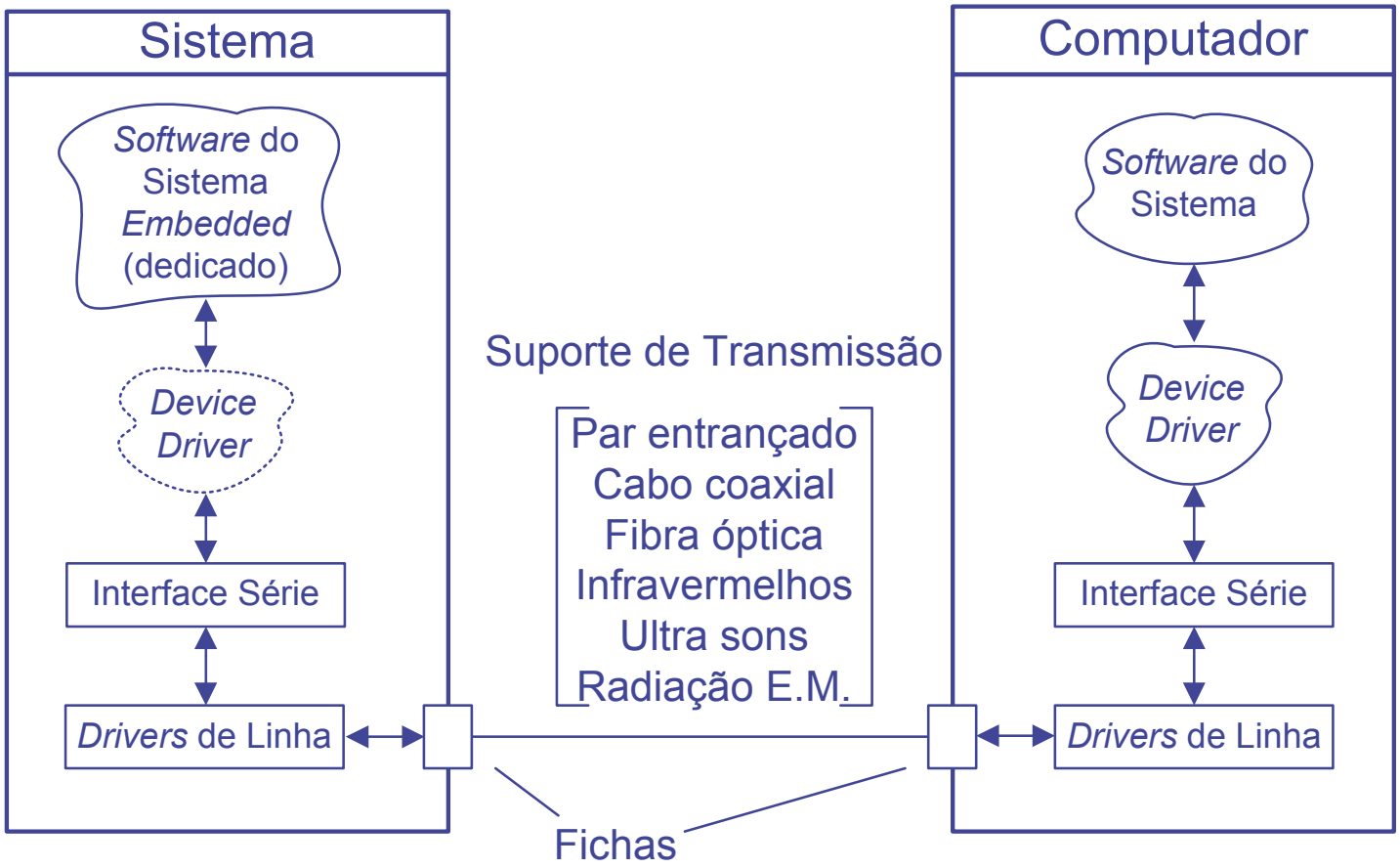


- Comunicação multiponto ("half-duplex"):



Elementos de uma ligação série

- Exemplo de uma ligação série entre um sistema embutido ("embedded" ou dedicado) e um computador de uso geral (PC)



Aula 14

- A interface SPI (*Serial Peripheral Interface*)
- Sinalização
- Sequência de operação
- Arquiteturas de ligação
- Tipos de transferências
- Passos de configuração de um *master* SPI

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Introdução

- SPI – sigla para "Serial Peripheral Interface"
- Interface definida inicialmente pela Motorola (Microwire da National Semiconductor é um *subset* do protocolo SPI)
- O SPI é utilizado para comunicar com uma grande variedade de dispositivos:
  - Sensores de diverso tipo: temperatura, pressão, etc.
  - Cartões de memória (MMC / SD)
  - Circuitos: memórias, ADCs, DACs, Displays LCD (e.g. telemóveis), comunicação entre corpo de máquinas fotográficas e as lentes, ...
  - Comunicação entre microcontroladores
- Ligação a curtas distâncias (dezenas de cm)

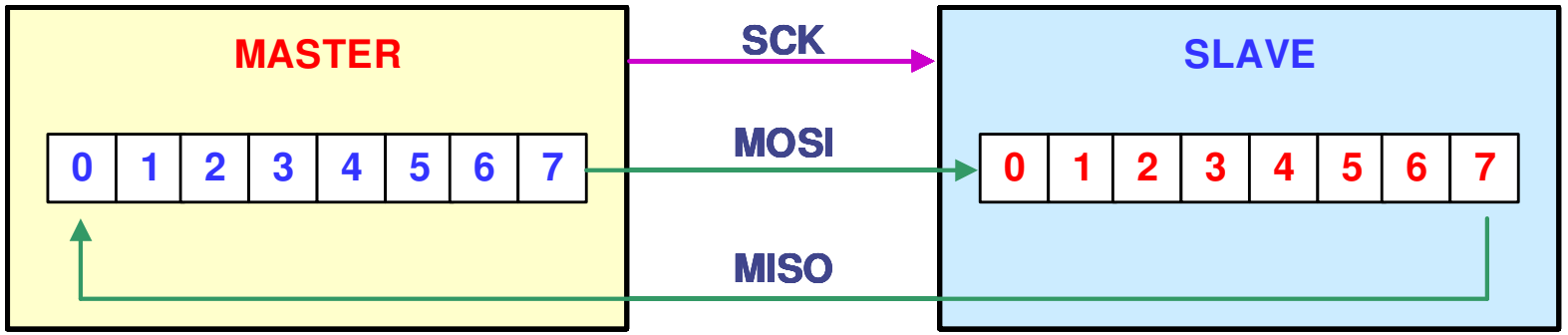
Descrição geral

- Arquitetura "Master-Slave" com ligação ponto a ponto
- Comunicação bidirecional "full-duplex"
- Comunicação síncrona (relógio explícito do *master*)
  - Relógio é gerado pelo *master* que o disponibiliza para todos os *slaves*
  - Não é exigida precisão ao relógio - os bits vão sendo transferidos a cada transição de relógio. Isto permite utilizar um oscilador de baixo custo no *master* (não é necessário um cristal de quartzo)
- Fácil de implementar por hardware ou por software
- Não são necessários "line drivers" (ou "transceivers") - circuitos de adaptação ao meio de transmissão. Os níveis lógicos correspondem aos da diferença de potencial de alimentação dos dispositivos (e.g. 3.3V)

Descrição geral

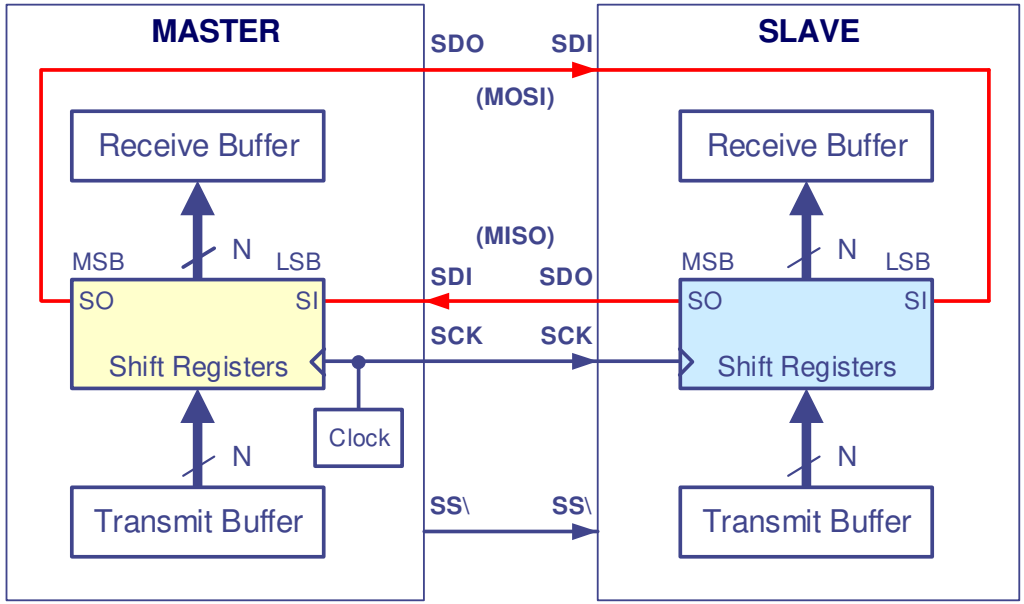
- Arquitetura "Master-Slave"
  - O sistema só pode ter um *master*
  - O *master* é o único dispositivo no sistema que pode controlar o relógio
- Um *master* pode estar ligado a vários *slaves*: para cada comunicação, apenas 1 *slave* é selecionado pelo *master* (daí ligação ponto a ponto)
- O *master* inicia e controla a transferência de dados
- Sinalização:
  - **SCK** – clock
    - Relógio gerado pelo *master* que sincroniza a transmissão/receção de dados
  - **MOSI** – Master Output Slave Input (SDO no *master*)
    - Linha do *master* para envio de dados para o *slave*
  - **MISO** – Master Input Slave Output (SDI no *master*)
    - Linha do *slave* para enviar dados para o *master*
  - **SS** – Slave select
    - Linha do *master* que seleciona o *slave* com quem vai comunicar

Descrição geral – esquema de princípio



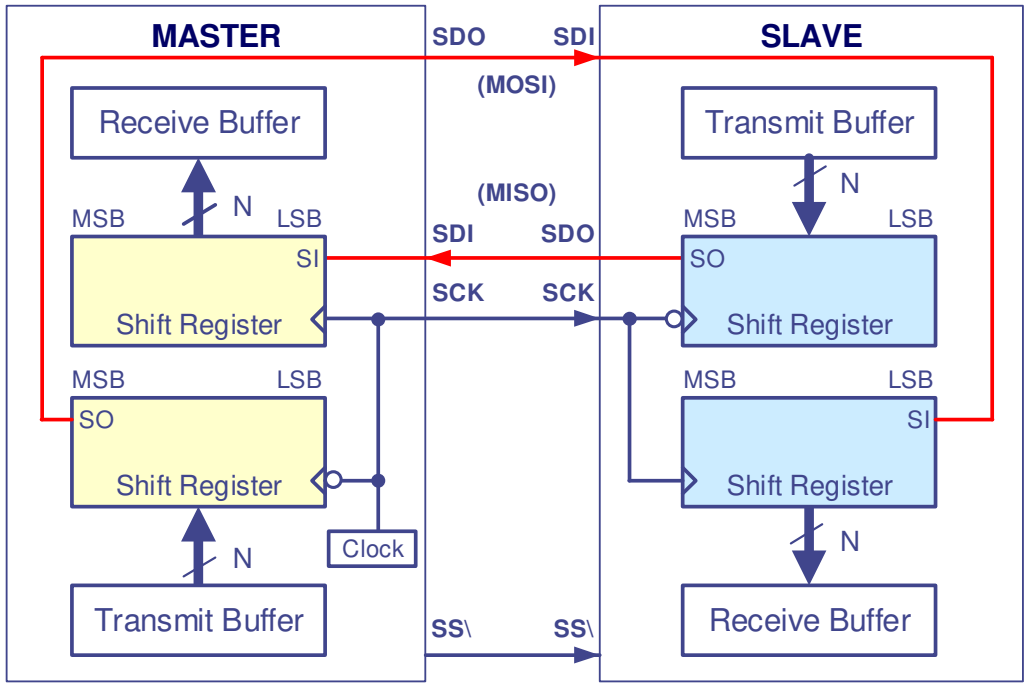
- Transmissão "full-duplex" baseada em dois *shift-registers* (um no *master* e outro no *slave*)
- Em cada ciclo de relógio:
  - O *master* coloca 1 bit na linha MOSI e o *slave* recebe-o
  - O *slave* coloca 1 bit na linha MISO e o *master* recebe-o
- Ao fim de N ciclos de relógio o *master* enviou uma palavra de N bits e recebeu do *slave* uma palavra com a mesma dimensão – "Data Exchange"
- Esta sequência é realizada mesmo quando é pretendida uma comunicação unidirecional

Sinalização



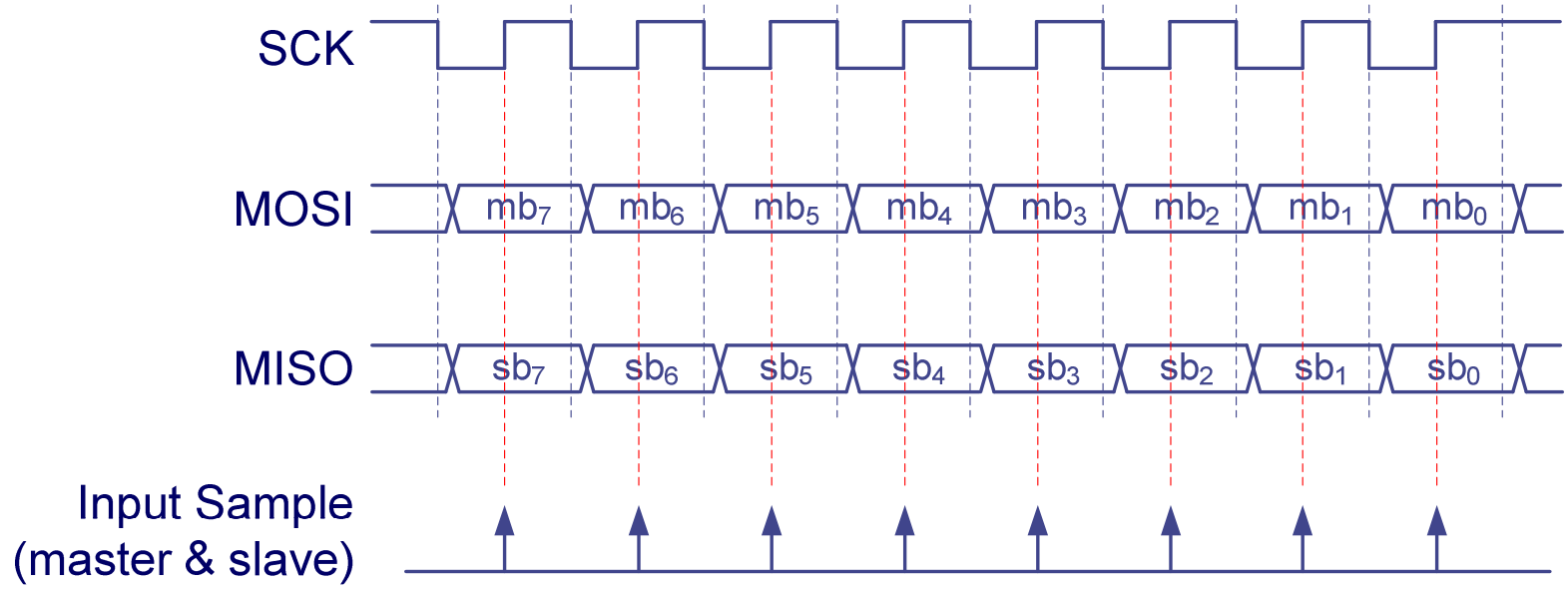
- Dados:
  - MOSI – Master Output Slave Input (SDO – serial data out no *master*)
  - MISO – Master Input Slave Output (SDI – serial data in no *master*)
- Controlo:
  - SS\ – Slave select (sinal ativado pelo *master* para selecionar o *slave* com quem vai comunicar)
  - SCK – serial clock

Sinalização



- O sinal de relógio tem um "duty-cycle" de 50%
- No exemplo da figura:
  - *master* e *slave* usam a transição negativa do relógio para colocarem 1 bit na linha (*master* na linha MOSI, *slave* na linha MISO)
  - Na transição positiva seguinte, o *master* armazena o valor presente na linha MISO e o *slave* armazena o valor que se encontra na linha MOSI

Operação – exemplo

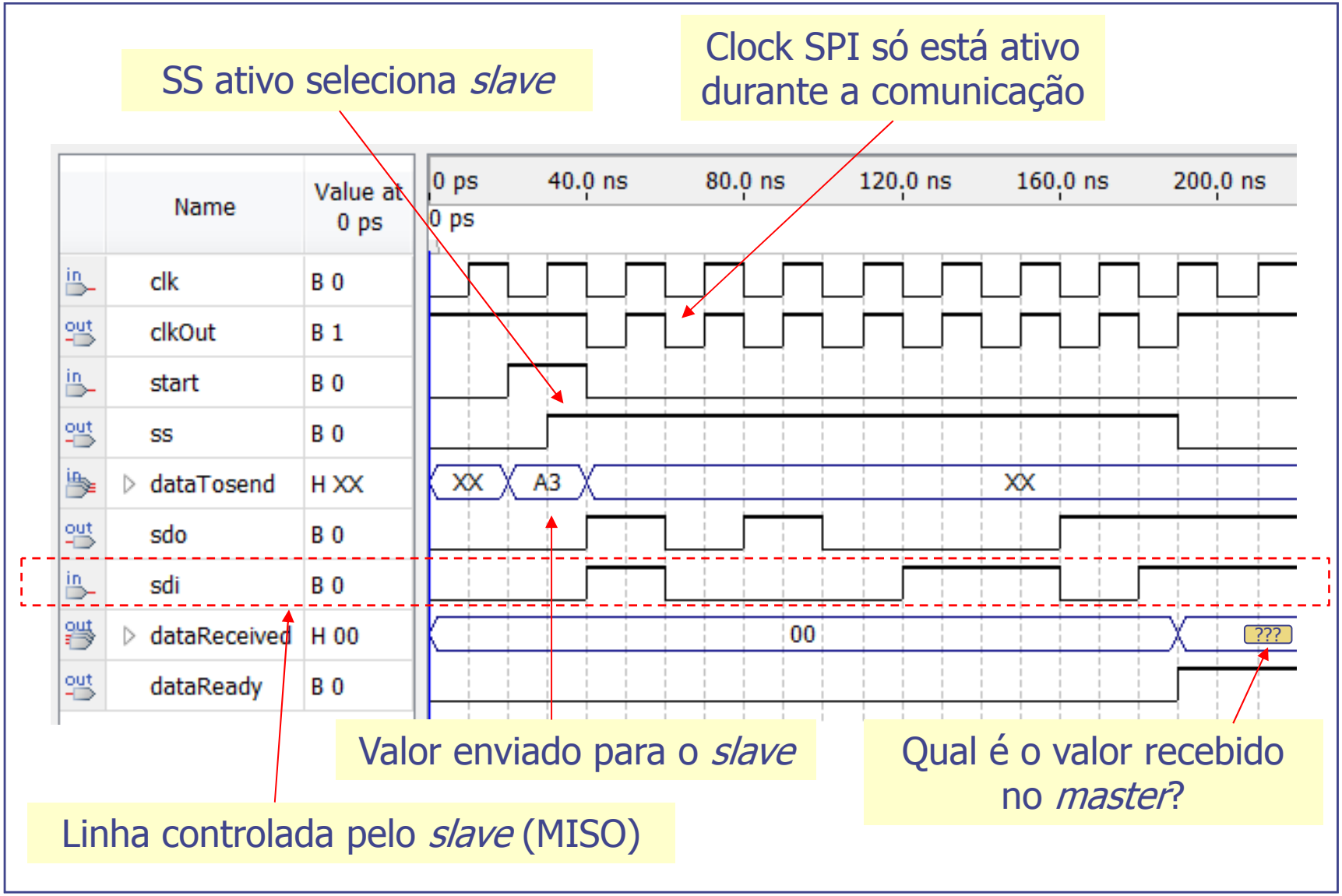


- A transição negativa do relógio é usada pelo *master* e pelo *slave* para colocar na respetiva linha de saída um bit de informação
- A transição positiva seguinte é usada pelo *master* e pelo *slave* para armazenar o bit presente na respetiva linha de entrada
- Ao fim de oito ciclos de relógio:
  - o valor inicialmente armazenado no *shift-register* do *master* foi transferido para o *shift-register* do *slave*
  - o valor inicialmente armazenado no *shift-register* do *slave* foi transferido para o *shift-register* do *master*

Operação

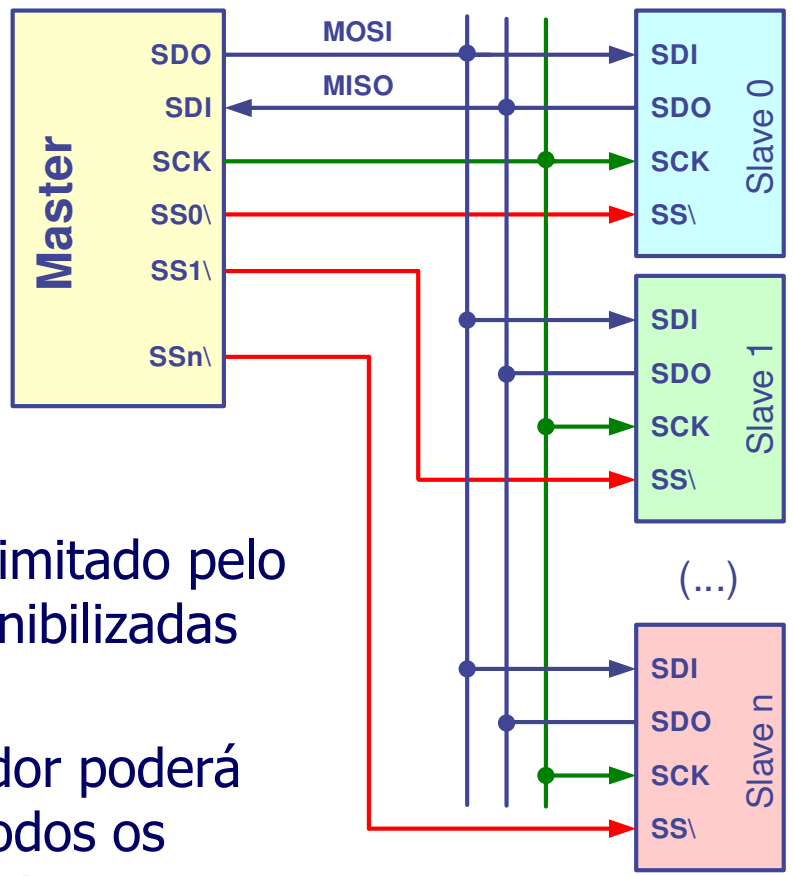
- O *master* ativa a linha SS\ do *slave* com que vai comunicar
- O *master* ativa o relógio que vai ser usado para sincronizar a troca de informação com o *slave* com quem vai comunicar
- Em cada ciclo do relógio, por exemplo na transição positiva
  - O *master* coloca na linha MOSI um bit de informação que é lido pelo *slave* na transição de relógio oposta seguinte
  - O *slave* coloca na linha MISO um bit de informação que é lido pelo *master* na transição de relógio oposta seguinte
- O *master* desativa a linha SS\ e desativa o relógio (que fica estável, por exemplo, no nível lógico 1)
  - Só há relógio durante o tempo em que se processa a transferência
- No final, o *master* e o *slave* trocaram o conteúdo dos seus *shift-registers*

Simulação de um *master* SPI



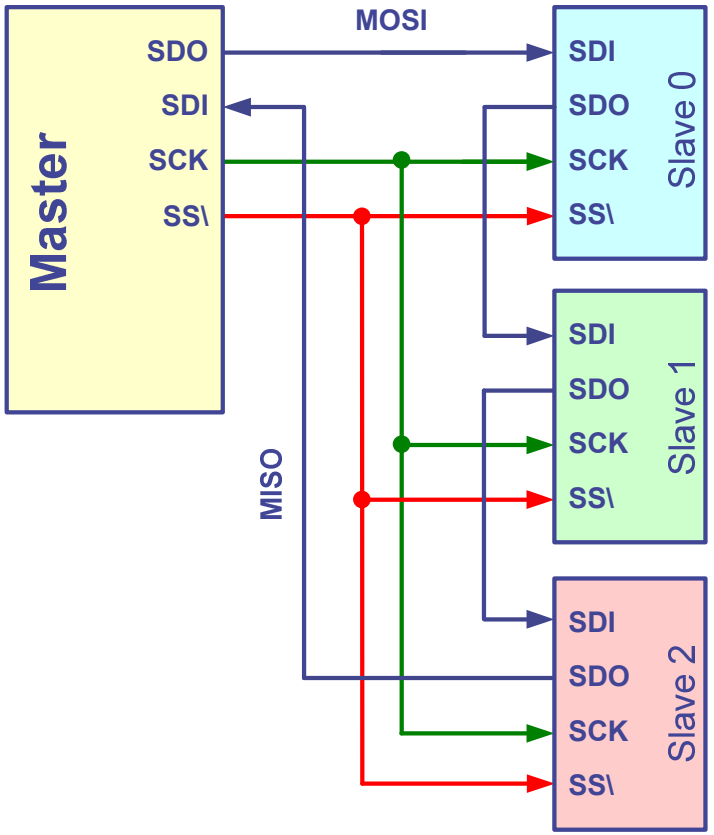
Arquiteturas de ligação – *slaves* independentes

- Sinais de seleção ("slave select") independentes
- Em cada instante apenas um SSx\ está ativo, isto é, apenas 1 *slave* está selecionado
- Os sinais SDO dos *slaves* (MISO) não selecionados estão em alta impedância
- O número máximo de *slaves* está limitado pelo número de linhas de seleção disponibilizadas pelo *master*
- Alternativamente, o microcontrolador poderá gerar, através de portos digitais, todos os sinais SSx\ necessários para comunicar com os *slaves*, ultrapassando a limitação anterior



Arquiteturas de ligação – Daisy Chain (cascata)

- Sinal "slave select" comum, SDO/SDI ligados em cascata
- Todos os *slaves* recebem o mesmo sinal de relógio gerado pelo *master*
- A saída de dados de cada *slave* liga à entrada de dados do seguinte
- Para que esta arquitetura funcione o *slave* tem de ser capaz de armazenar uma sequência de N bits enviados durante 1 ciclo de comando e enviar para a sua saída a mesma sequência de N bits durante o ciclo de comando seguinte
  - Enquanto o SS estiver ativo o *slave* ignora o comando recebido e envia-o para a saída DO no ciclo de comando seguinte
  - O *slave* apenas executa o comando quando o sinal SS é desativado



Tipos de transferências

- O SPI funciona sempre em modo "data exchange", isto é, o processo de comunicação envolve sempre a troca do conteúdo dos *shift-registers* do *master* e do *slave*
- Cabe aos dispositivos envolvidos na comunicação usar ou descartar a informação recebida
- Podem considerar-se os seguintes cenários de transferência:
  - **Bidirecional**: são transferidos dados válidos em ambos os sentidos (master → slave e slave → master)
  - **Master → slave (operação de escrita)**: *master* transfere dados para o *slave*, e ignora/descarta os dados recebidos
  - **Slave → master (operação de leitura)**: *master* pretende ler dados do *slave*; para isso transfere para o *slave* uma palavra com informação irrelevante (por exemplo 0); o *slave* ignora/descarta os dados recebidos

Configuração de um *master* SPI

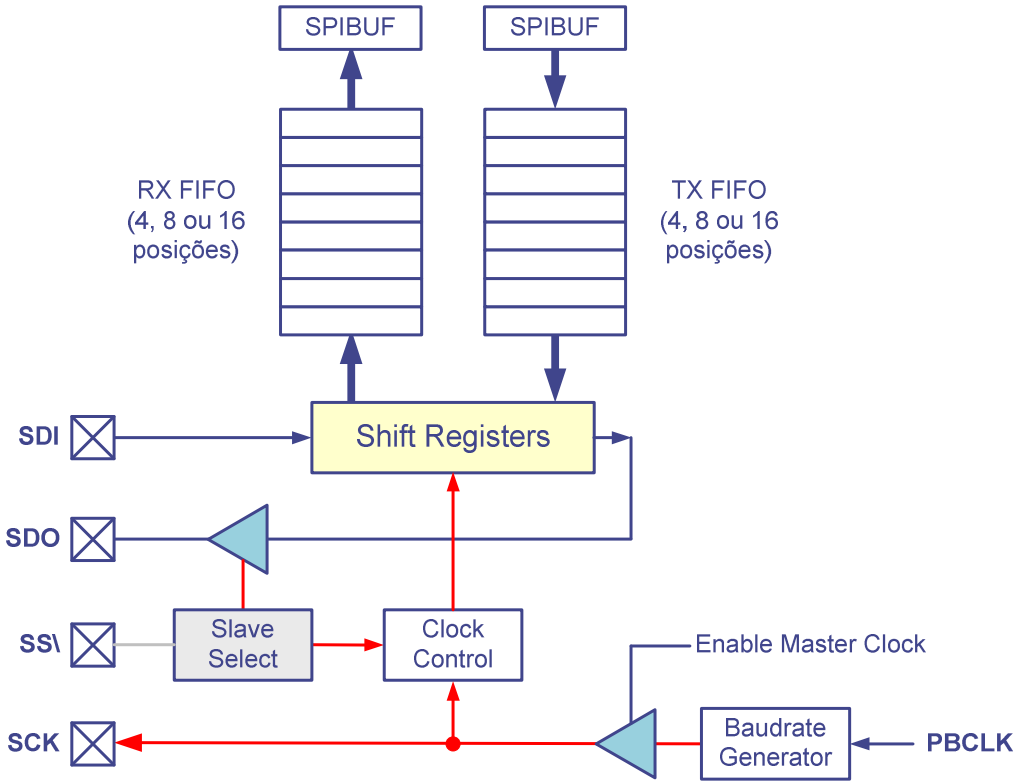
- Antes de iniciar a transferência há algumas configurações que são efetuadas no *master* (através do seu modelo de programação) para adequar os parâmetros que definem a comunicação às características do *slave* com que vai comunicar:
  1. Configurar a frequência de relógio
  2. Configurar o nível lógico de repouso ("idle") do sinal de relógio
  3. Especificar qual o flanco do relógio usado para a transmissão (a receção é efetuada no flanco oposto). Esta configuração é feita em função das características do *slave* com o qual o *master* vai comunicar:
     - Transmissão no flanco ascendente (consequentemente, a receção é feita no flanco descendente)
     - Transmissão no flanco descendente (consequentemente, a receção é feita no flanco ascendente)

Interface SPI no PIC32

S

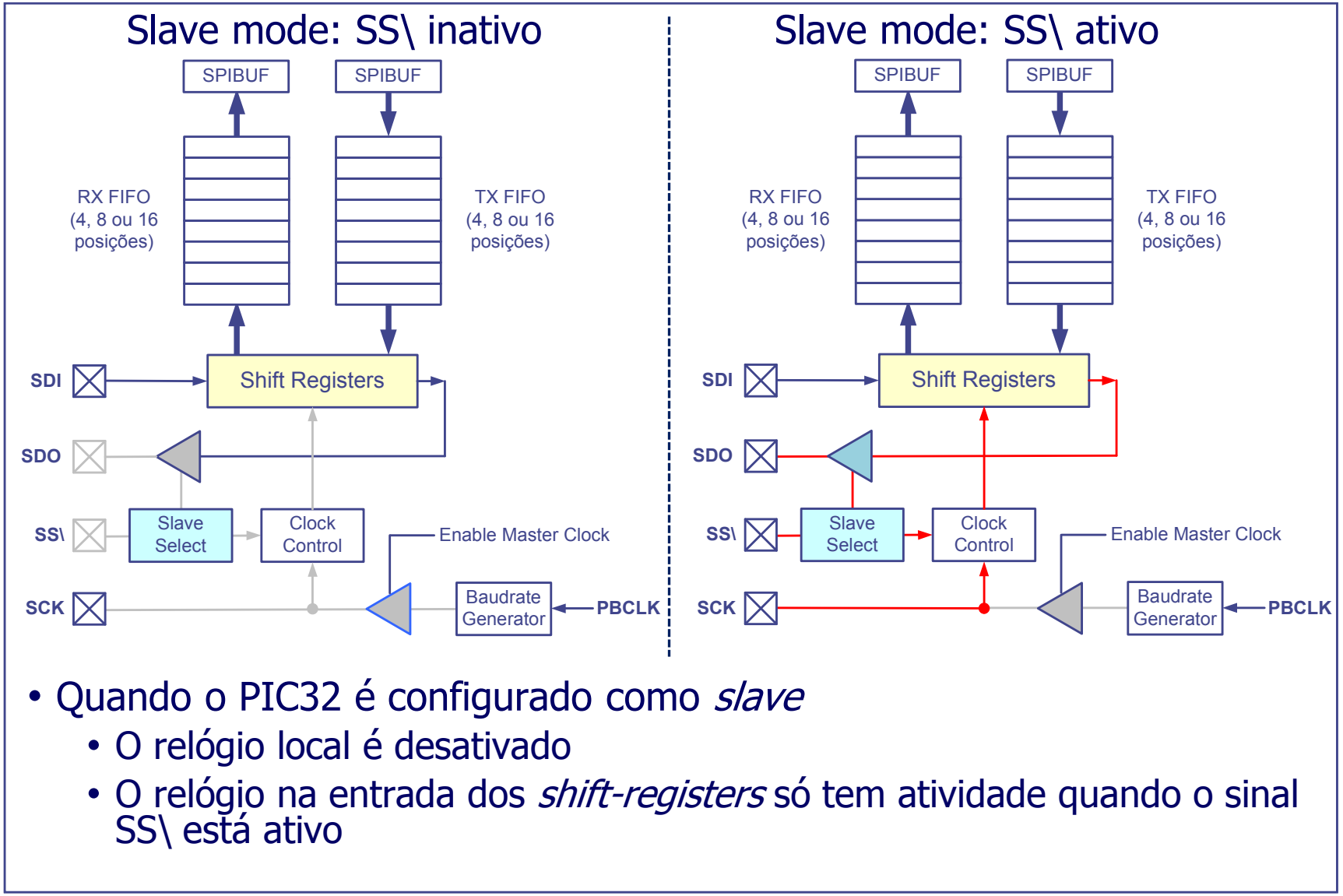
- O PIC32MX795F512H disponibiliza 3 módulos de comunicação SPI
- Cada um dos módulos pode ser configurado para funcionar como *master* ou como *slave*
- Comprimento de palavra configurável: 8, 16 ou 32 bits
- *Shift-registers* separados para receção e transmissão
- Os registos de receção e transmissão são FIFOS:
  - 16 posições se o comprimento de palavra for 8 bits
  - 8 posições se o comprimento de palavra for 16 bits
  - 4 posições se o comprimento de palavra for 32 bits
- Cada uma dos módulos pode ser configurado para gerar interrupções em função da ocupação dos FIFOS (e.g. TX FIFO tem, pelo menos, 1 posição livre; RX FIFO tem, pelo menos, 1 palavra disponível para ser lida)

Interface SPI no PIC32



- Quando o PIC32 é configurado como *master*
  - O relógio local é ativado durante cada transmissão
  - O buffer 3state do SDO está sempre ativo
  - A entrada SS\ é ignorada

Interface SPI no PIC32

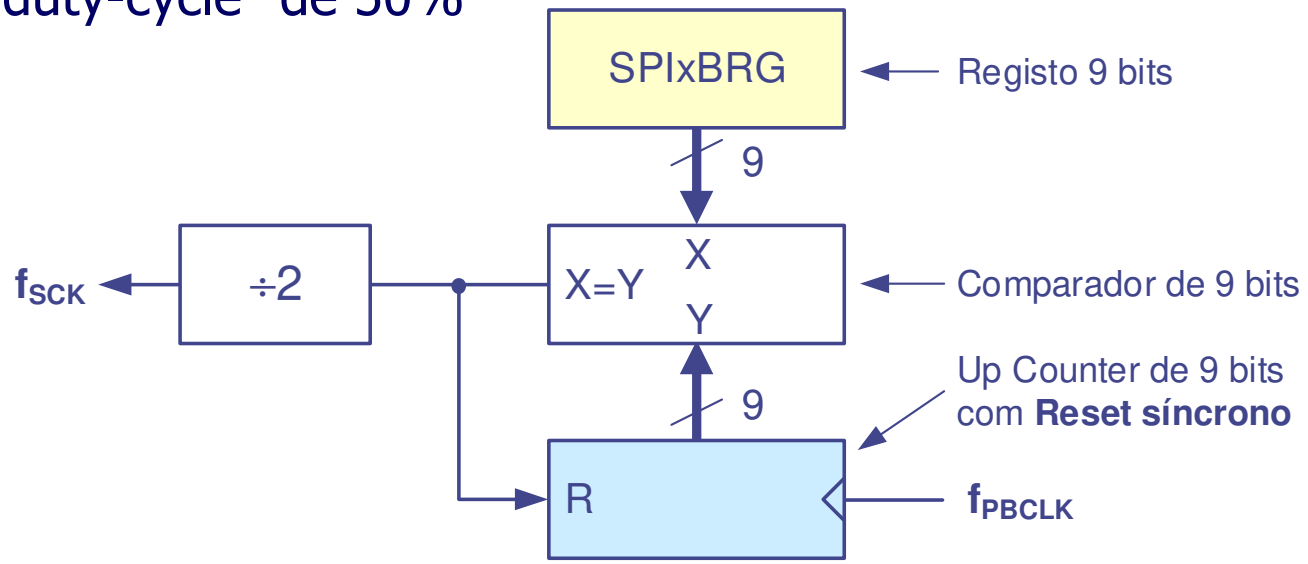


- Quando o PIC32 é configurado como *slave*
  - O relógio local é desativado
  - O relógio na entrada dos *shift-registers* só tem atividade quando o sinal SS\ está ativo

Interface SPI no PIC32 – gerador de relógio

S

- Utiliza uma arquitetura semelhante à de um timer, em que o sinal de relógio de entrada é o Peripheral Bus Clock (20 MHz na placa DETPIC32).
- Com a divisão por 2 à saída do comparador obtém-se um relógio com "duty-cycle" de 50%



- **$f_{SCK} = f_{PBCLK} / (2 * (SPIxBRG + 1))$**, em que SPIxBRG representa a constante armazenada no registo com o mesmo nome

Aula 15

- O barramento CAN (*Controller Area Network*)
- Características fundamentais
- Aplicações
- Topologia da rede e codificação
- Tipos de tramas
- Deteção de erros
- Filtros de aceitação de mensagens
- Arbitragem

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Introdução

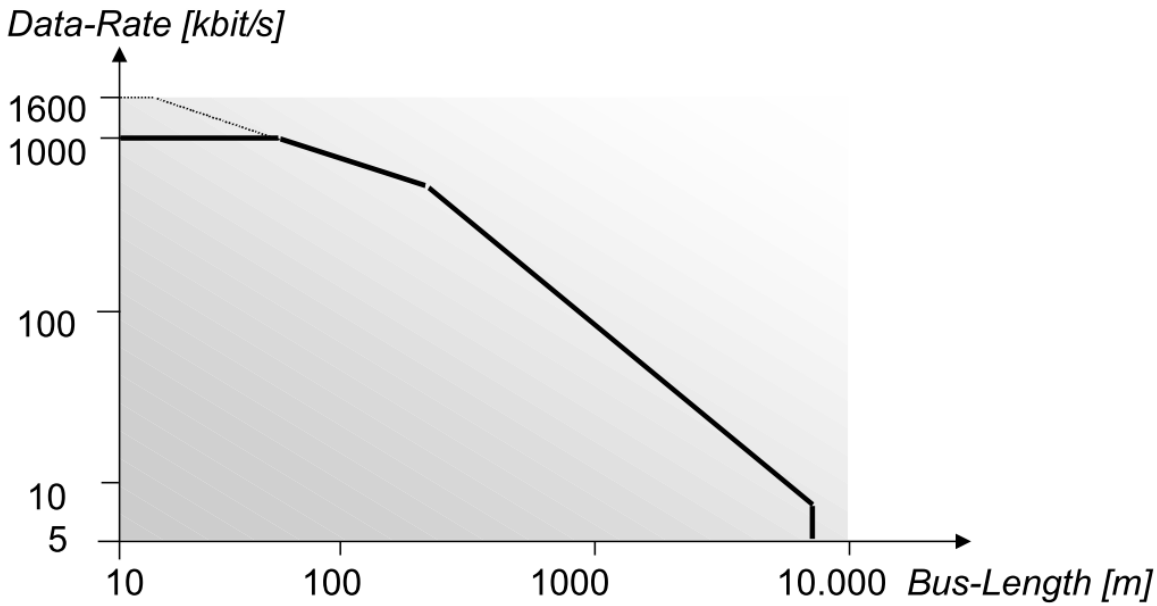
- Desenvolvido em 1991 (versão 2.0) pela Bosch para simplificar as cablagens nos automóveis

  (http://esd.cs.ucr.edu/webres/can20.pdf)
- Utiliza comunicação diferencial em par entrançado
- Taxas de transmissão até 1 Mbit/s
- Adequado a aplicações de segurança crítica; elevada robustez
  - Tolerância a interferência eletromagnética
  - Capacidade de detetar diferentes tipos de erros
  - Baixa probabilidade de não deteção de um erro de transmissão (4.7×10^{-11})
- Atualmente usado num leque muito variado de aplicações
  - Comunicação entre subsistemas de um automóvel
  - Aviónica, Aplicações industriais, Domótica, Robótica
  - Equipamentos médicos, ...

Introdução

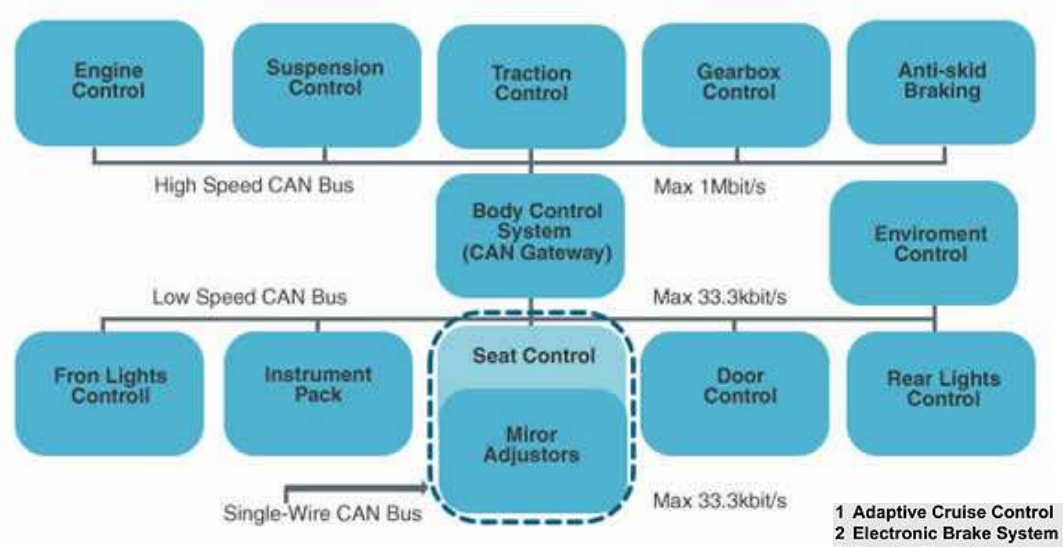
- Transmissão em "broadcast": a informação enviada pelo transmissor pode ser recebida por todos os nós ao mesmo tempo
- Comunicação bidirecional "half-duplex"
- A informação produzida é encapsulada em tramas
- O CAN é um barramento "multi-master": qualquer nó do barramento pode produzir informação e iniciar uma transmissão
- Uma vez que dois ou mais nós podem querer aceder simultaneamente ao barramento para transmitir, tem que haver uma forma de arbitrar o acesso ao meio
- No CAN cada mensagem tem um ID único que identifica a natureza do seu conteúdo; esse ID determina também a prioridade da mensagem e, consequentemente, a prioridade no acesso ao barramento

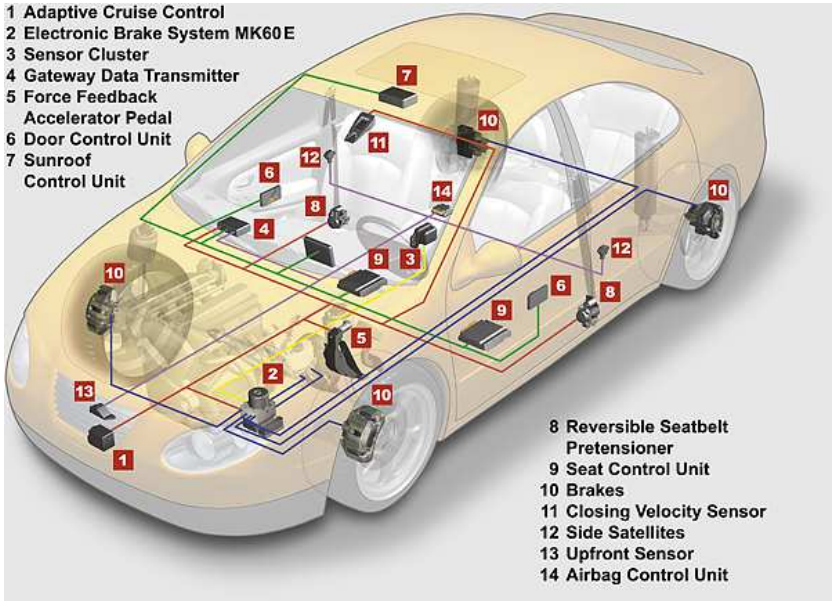
Comprimento máximo do barramento



Bit Rate	Bus Length	Nominal Bit-Time
1 Mbit/s	30 m	1 µs
800 kbit/s	50 m	1,25 µs
500 kbit/s	100 m	2 µs
250 kbit/s	250 m	4 µs
125 kbit/s	500 m	8 µs
62,5 kbit/s	1000 m	20 µs
20 kbit/s	2500 m	50 µs
10 kbit/s	5000 m	100 µs

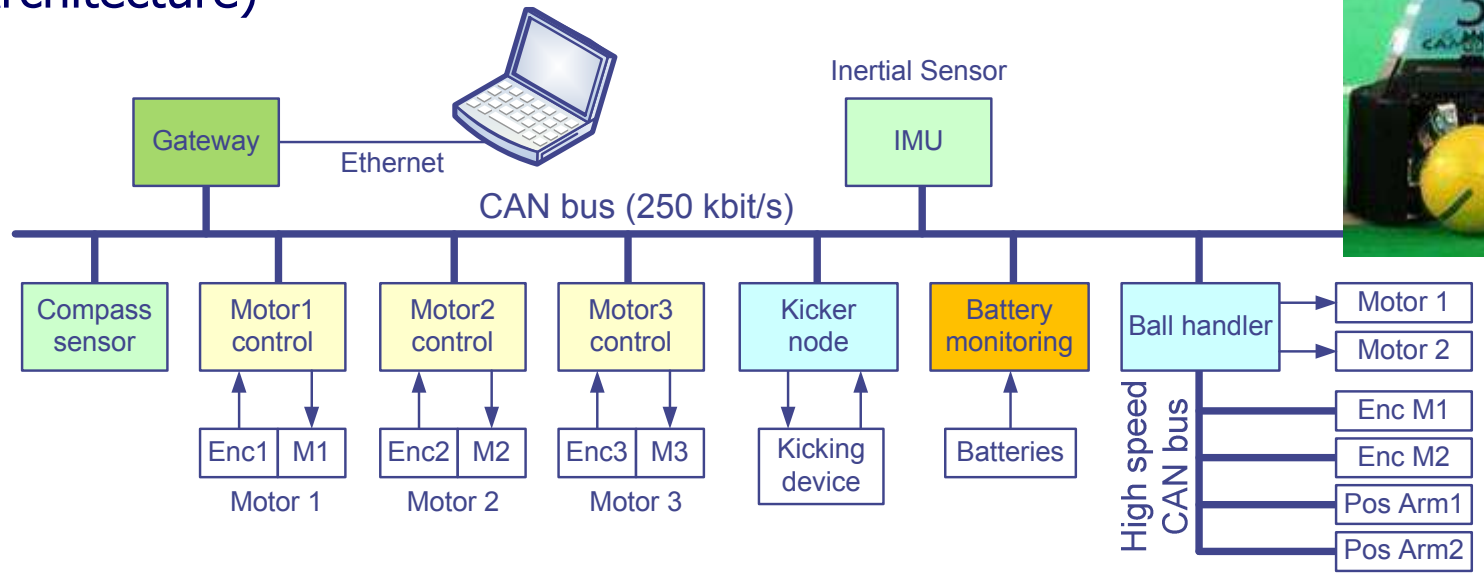
Exemplos de aplicação





Exemplos de aplicação

- Infraestrutura sensorial e de atuação dos robots da equipa de futebol robótico do DETI: CAMBADA (**C**ooperative **A**utonomous **M**obile ro**B**ots with **A**dvanced **D**istributed **A**rchitecture)

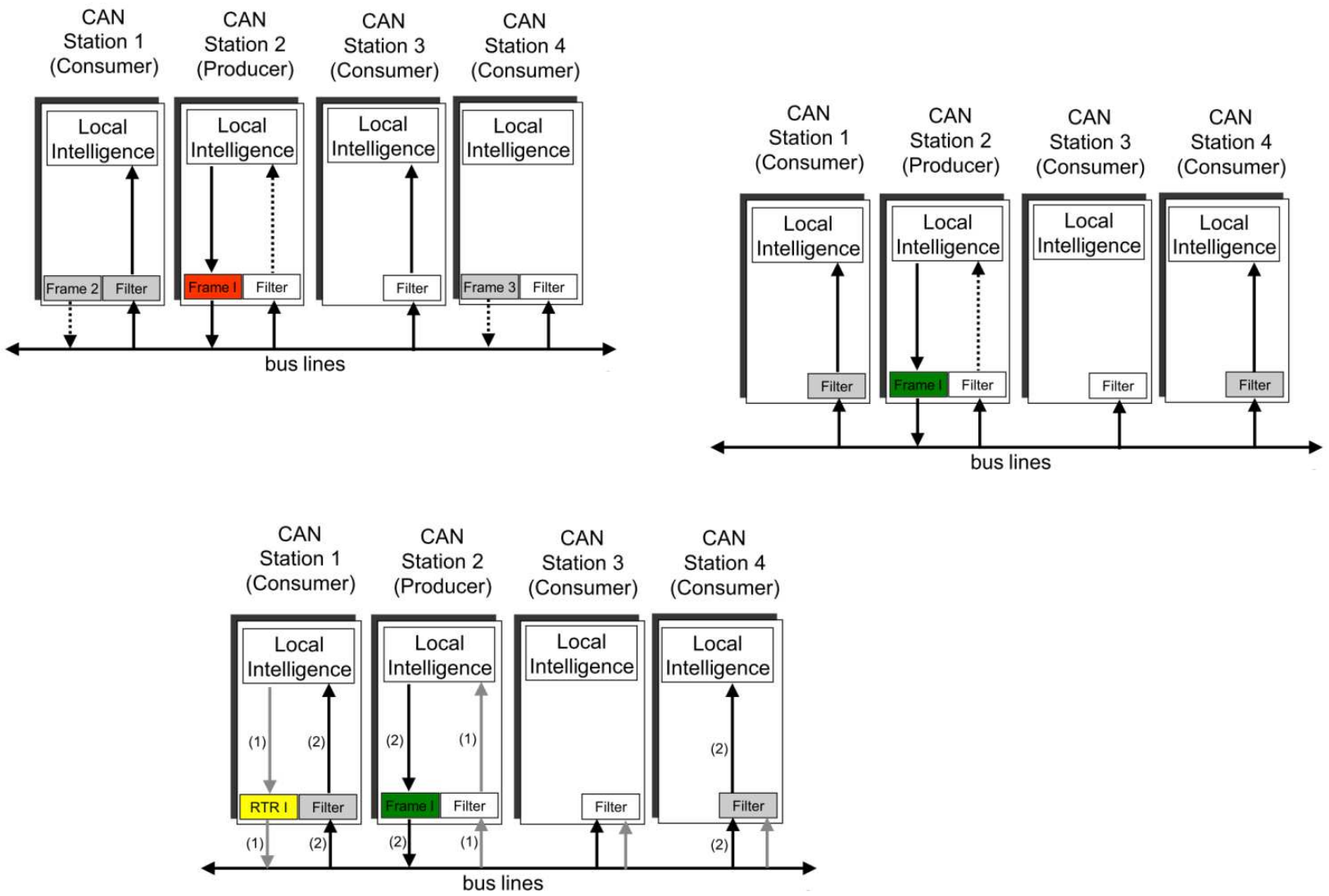


- Arquitetura distribuída em que cada nó desempenha uma tarefa ou conjunto de tarefas relacionadas
- O sistema é facilmente alterável; por exemplo, acrescentar um novo sensor não implica qualquer alteração na estrutura existente (basta ligar o novo nó ao barramento CAN)

Características fundamentais

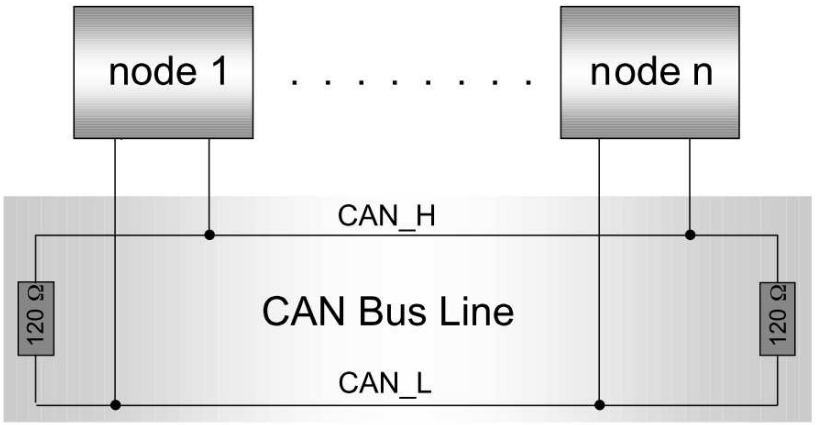
- Sincronização de relógio:
  - **Relógio implícito** (comunicação assíncrona, i.e. não há transmissão do relógio - o transmissor e o recetor têm relógios locais independentes)
- Transmissão orientada ao bit
- Barramento série "multi-master"
  - Diversos nós trocam mensagens encapsuladas em tramas
- Paradigma produtor-consumidor / Transmissão em "broadcast"
  - Identificação do conteúdo da mensagem (não existe identificação do nó de origem ou de destino)
- Capacidade de Remote Transmission Request
- Correção de erros baseada em retransmissão

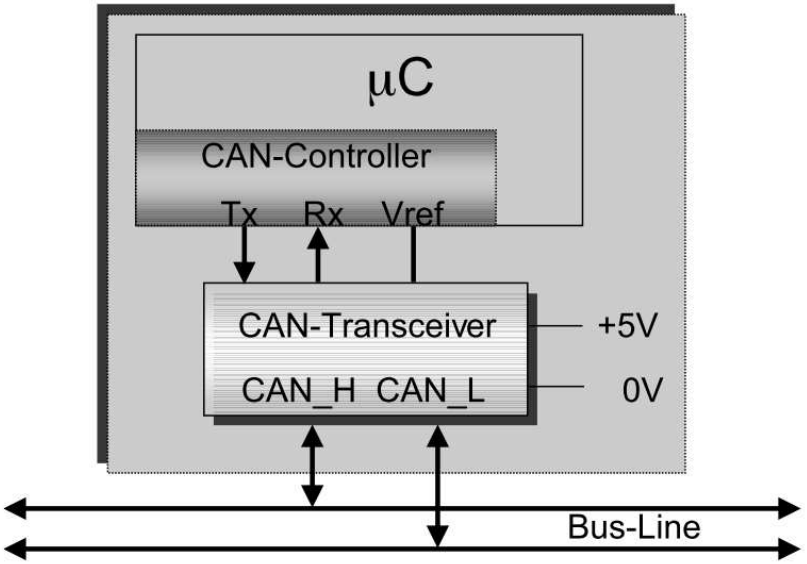
Características fundamentais



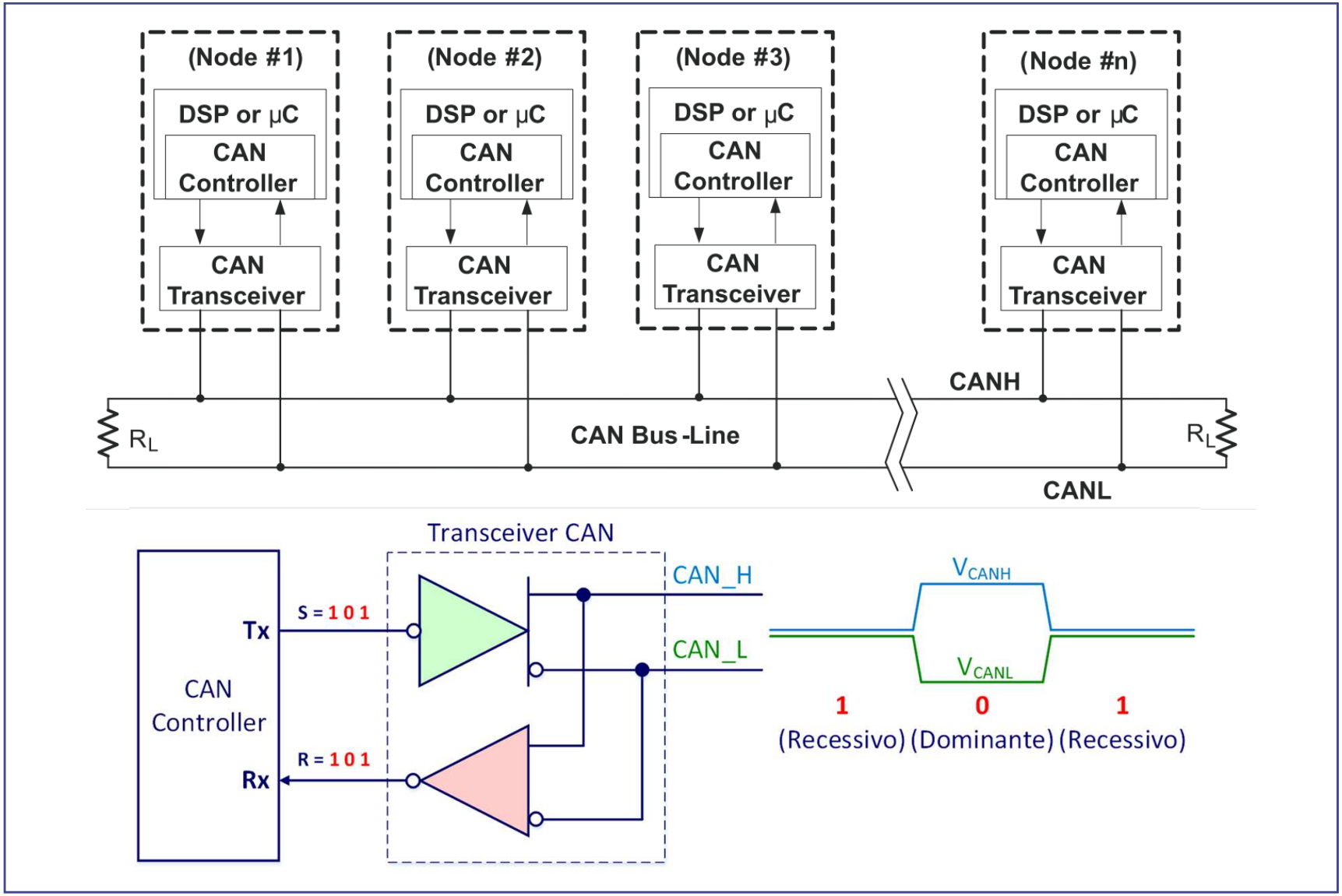
Topologia da rede e estrutura de um nó

- Comunicação **diferencial**, **par entrançado**
- Na transmissão, o "transceiver" transforma o nível lógico presente na linha Tx em duas tensões e coloca-as nas linhas **CAN_H** e **CAN_L**
- Na receção, o "transceiver" discrimina o nível lógico pela **diferença de tensão entre CAN_H e CAN_L** e o resultado é enviado através da linha Rx para o controlador CAN

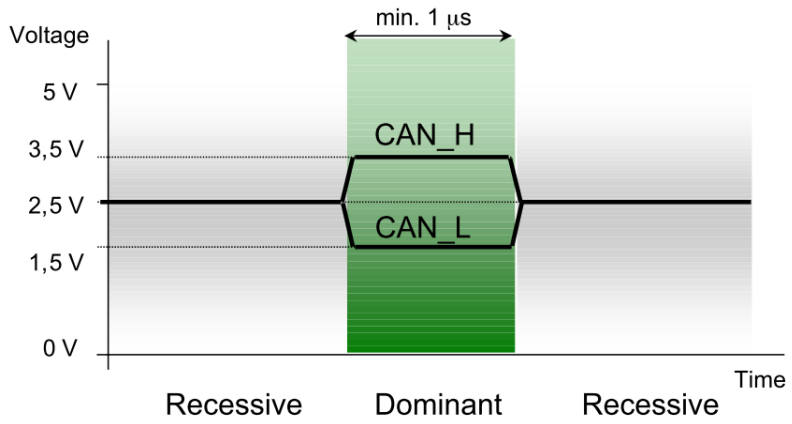


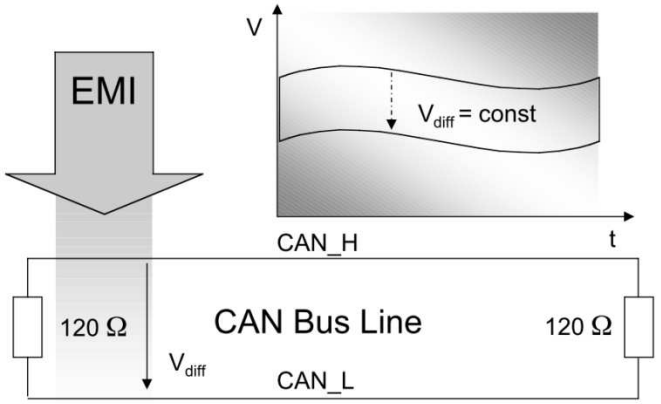


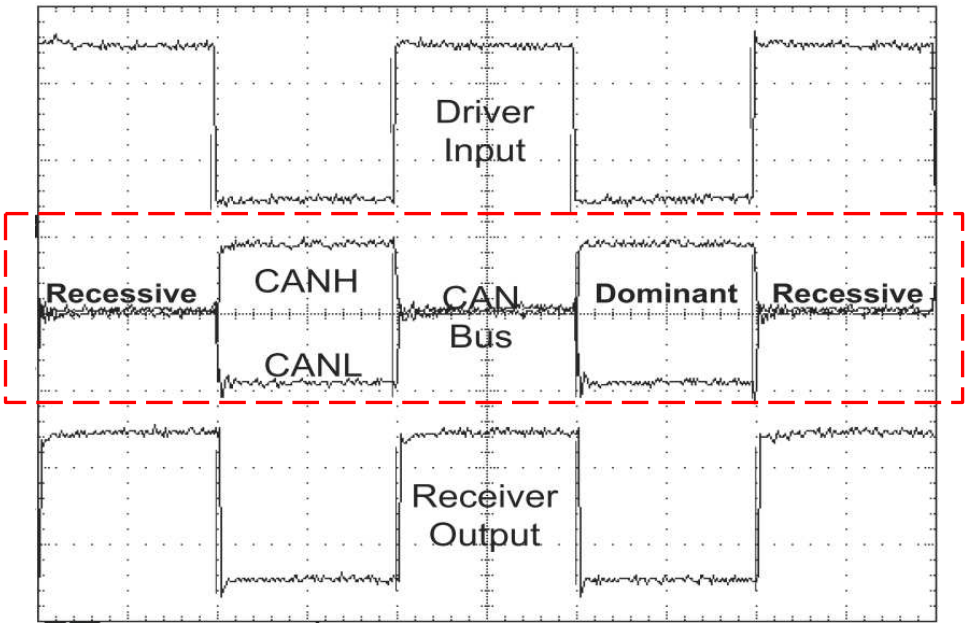
Topologia da rede e estrutura de um nó



Transmissão diferencial



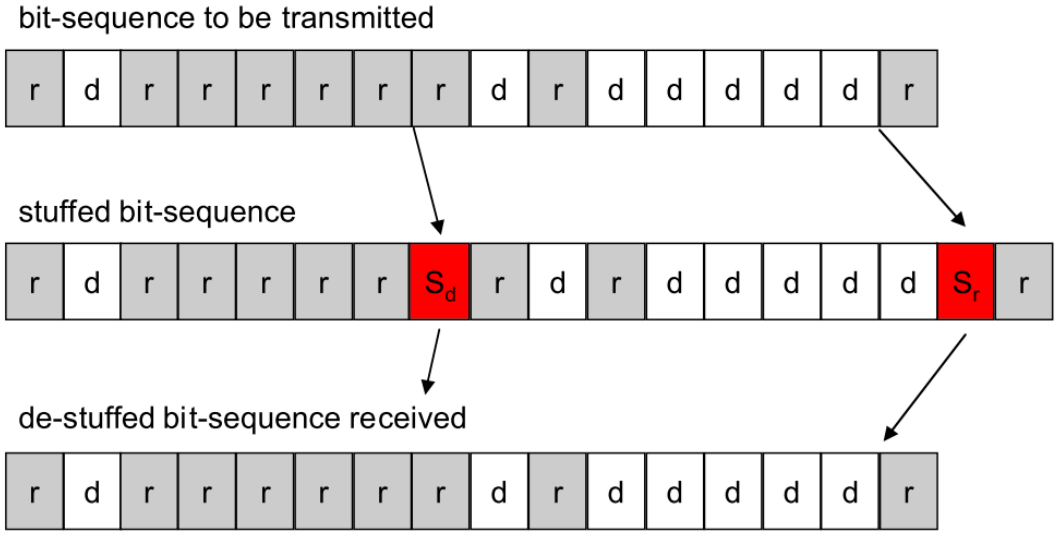




Codificação

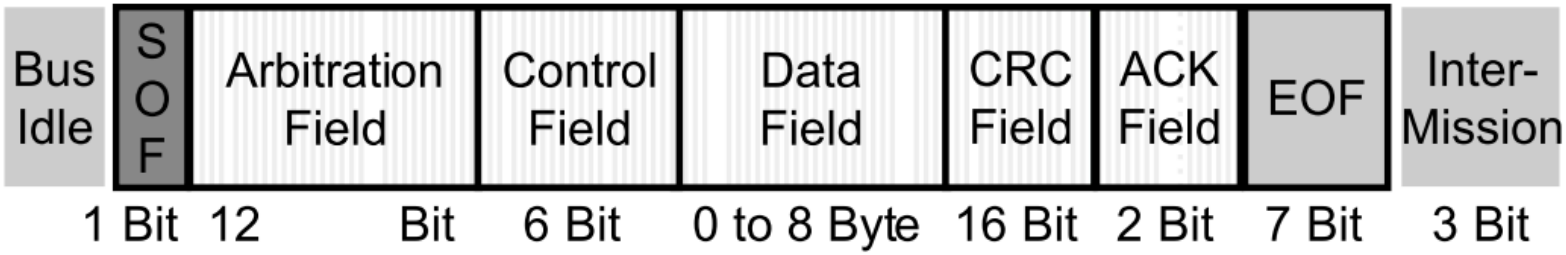
- Codificação Non-Return-to-Zero - bit recessivo ('1')/dominante ('0')

- "**Bit-stuffing**"
  - Por cada 5 bits iguais é inserido 1 bit de polaridade oposta

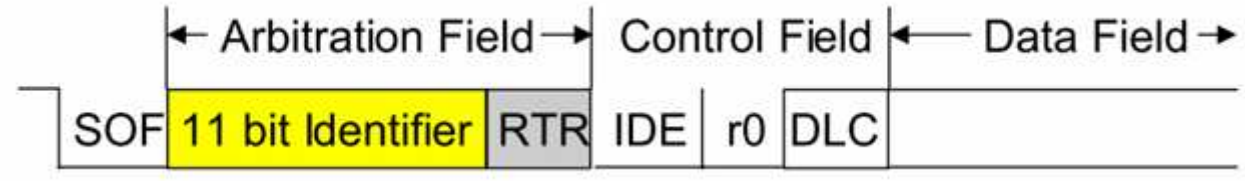


  - Garante um tempo máximo entre transições da linha de dados, assegurando que há transições suficientes para manter sincronizados os instantes de amostragem de dados em cada um dos nós

Formato da trama de dados (CAN 2.0A)

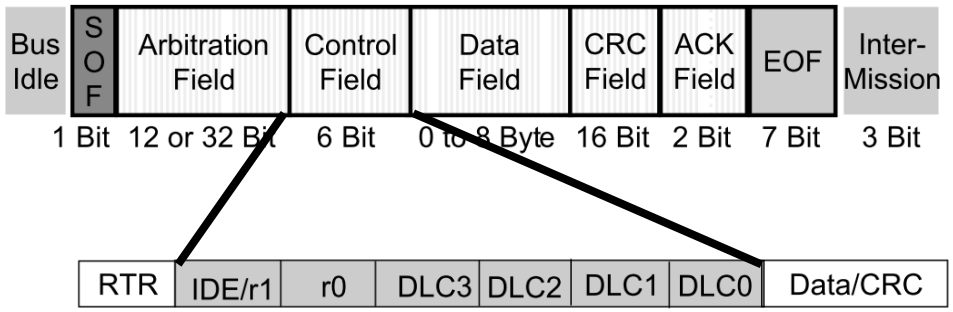


- **SOF** (Start of Frame)
  - Bit dominante ('0') indica o início da trama
  - Usado para sincronização dos instantes de amostragem dos nós recetores
- **Arbitration**
  - **Identifier** (11 bits) – identificador da mensagem que também serve para arbitragem entre diferentes *masters* que podem iniciar a transmissão das suas tramas em simultâneo (id mais baixo, maior prioridade)
  - **RTR** (1 bit) Remote Transmission Request - dominante numa trama de dados
- Standard Frame Format (CAN 2.0A):



Formato da trama de dados (CAN 2.0A)

- **IDE** (identifier extension)
  - Bit dominante ('0') significa trama standard (CAN 2.0A, 11-bit identifier)
  - Bit recessivo ('1') significa trama CAN 2.0B (com identificador estendido de 29 bits)
- **r0** – reservado
- **DLC3 – DLC0**
  - Número de bytes de dados (0 a 8)
- **Data** (Campo de dados)
  - 0 a 8 bytes (0 a 64 bits)
  - MSBit first (/byte)



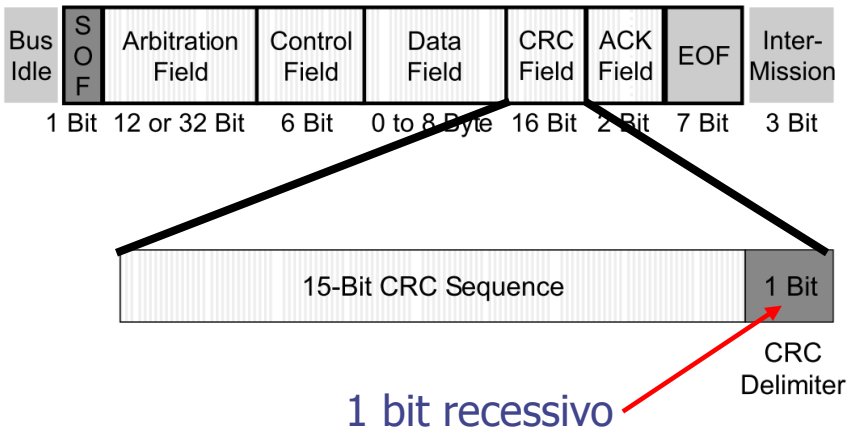
No. of Data Bytes	Data Length Code (DLC)			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d/r	d/r	d/r

min. length of Data Field = 0 Byte

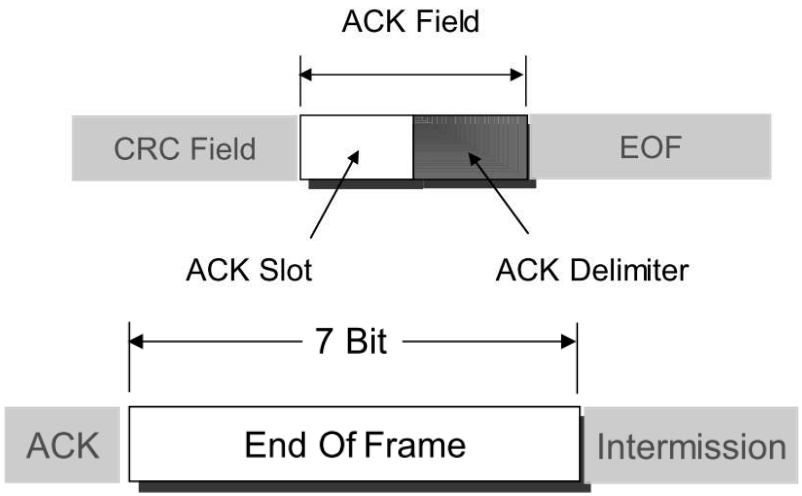
max. length of Data Field = 8 Byte

Formato da trama de dados (CAN 2.0A)

- **CRC** (Cyclic Redundancy Check)
  - Deteção de erros
  - Produtor e consumidor calculam a sequência de CRC com base nos bits transmitidos/recebidos
  - Produtor transmite a sequência CRC calculada
  - Consumidor compara a sequência CRC calculada localmente com a recebida do produtor
- **ACK** (Acknowledge)
  - Validação da trama (ACK Slot)
  - Recessivo (produtor)
  - Dominante (1+ consumidores)
- **EOF** (End of Frame)
  - Terminação da trama (7 bits recessivos)
- **IFS** (interframe/intermission)
  - Mínimo de 3 bits recessivos

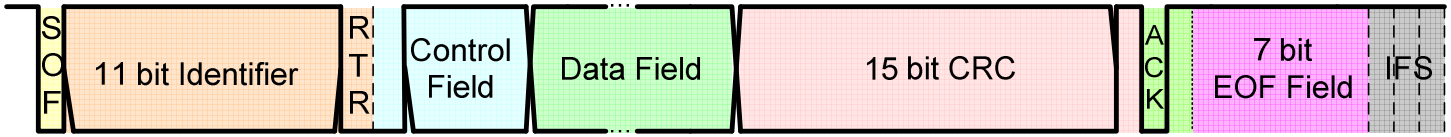


Remark: *The CRC Delimiter is a fixed formatted recessive bit.*



Tipos de tramas

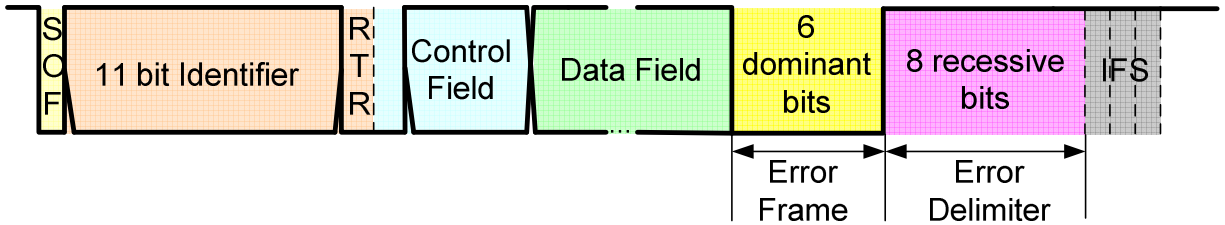
- **Data Frame**
  - Usada no envio de dados de um nó produtor para o(s) consumidor(es); numa trama de dados o bit RTR está a '0' (dominante)



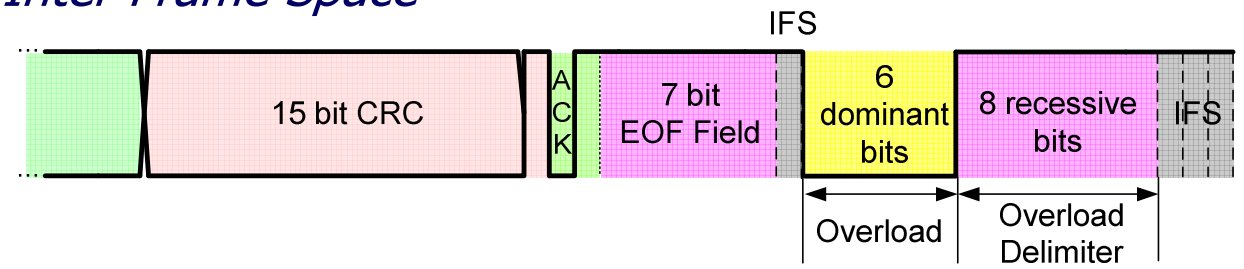
- **Remote Transmission Request Frame**
  - Enviada por um nó consumidor a solicitar (ao produtor) a transmissão de uma trama de dados específica (trama tem o campo RTR a '1' – recessivo, o que a diferencia de uma trama de dados)

Tipos de tramas

- **Error Frame**
  - Usada para reportar um erro detetado (a trama de erro sobrepõe-se a qualquer comunicação invalidando uma transmissão em curso



- **Overload Frame**
  - Usada para atrasar o envio da próxima trama (enviada por um nó em situação de sobrecarga que não teve tempo para processar a última trama enviada). Deve iniciar-se durante os dois primeiros bits do *Inter Frame Space*

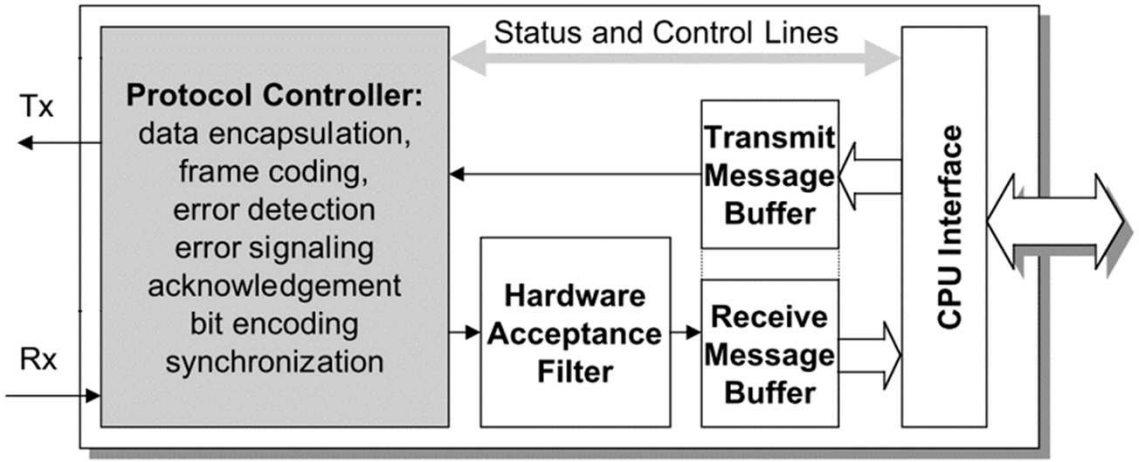


Deteção de erros de comunicação

S

- São usados vários métodos de deteção de erros. Se a receção de uma trama falha em qualquer um deles essa trama não é aceite e é gerada uma trama de erro que força o produtor a reenviar
- **CRC Error** – o CRC calculado não coincide com o CRC recebido
- **Acknowledge Error** – o produtor não recebe um bit dominante ('0') no campo ACK, o que significa que a mensagem não foi recebida por nenhum nó da rede (todos os nós fazem o "acknowledge" da receção da trama)
- **Form Error** – esta verificação analisa campos da mensagem que devem ter sempre o valor ´lógico '1' (recessivo): EOF, delimitador do ACK e delimitador do CRC; se for detetado um bit dominante em qualquer destes campos é gerado um erro
- **Bit Error** – cada bit transmitido é analisado pelo produtor da mensagem: se o produtor lê um valor que é o oposto do que escreveu gera um erro (exceções: identificador, ACK)
- **Stuffing Error** – se, após 5 bits consecutivos com o mesmo nível lógico não for recebido um de polaridade oposta, é gerado um erro

Arquitetura típica de um controlador CAN



- O controlador CAN implementa o protocolo em hardware
- O "CPU interface" assegura, tipicamente, a comunicação com o CPU de um microcontrolador (registos de controlo, estado e dados – buffers)
- O "**hardware acceptance filter**" filtra as mensagens recebidas com base no seu ID. Por programação é possível especificar quais os IDs das mensagens que serão copiadas para o "Receive Message Buffer" (i.e., que serão disponibilizadas ao microcontrolador)
- Este mecanismo de filtragem ao descartar mensagens não desejadas, reduz a carga computacional no microcontrolador

Filtros de aceitação de mensagens e máscaras

- O CAN é um barramento de tipo "**broadcast**", ou seja, uma mensagem transmitida por um nó é recebida por todos os nós da rede
- O controlador CAN de cada nó lê todas as mensagens que circulam no barramento e coloca-as num registo temporário designado por "Message Assembly Buffer" (MAB)
- Logo que uma mensagem válida é recebida no MAB, é aplicado um **mecanismo de filtragem** que permite que apenas as mensagens de interesse para o nó sejam copiadas para o buffer de receção (as restantes são descartadas)
- A filtragem é feita por verificação dos bits do identificador da mensagem

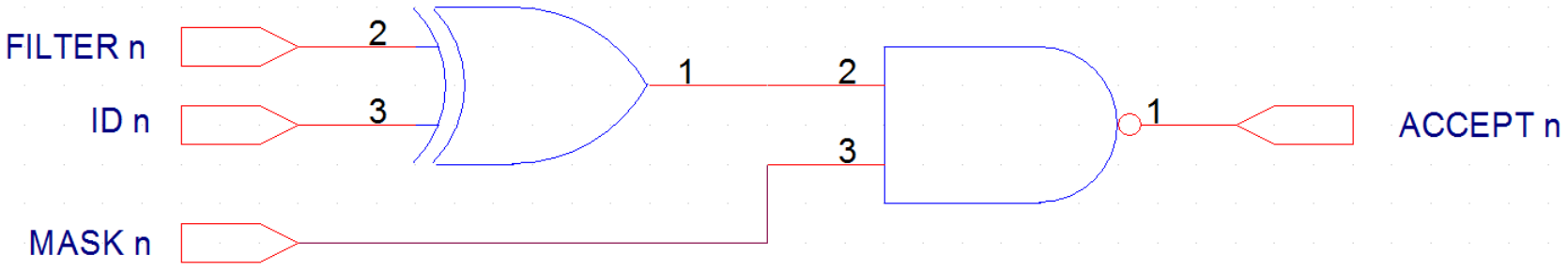
Filtros de aceitação de mensagens e máscaras

- O mecanismo de filtragem é constituído por um conjunto de **filtros** e **máscaras**: na sua forma mais simples, a mensagem só é copiada para o buffer de receção se o identificador da mensagem igualar um dos filtros de aceitação (previamente configurados por software)
- As máscaras fornecem flexibilidade adicional ao permitir definir quais os bits do identificador que têm que ser iguais aos definidos nos filtros e quais os que são aceites incondicionalmente

Mask bit n	Filter bit n	Message Identifier bit n	Accept/Reject bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

- $ACCEPT = ACCEPT_{10} . ACCEPT_{9} . \ldots . ACCEPT_{0}$
- Se ACCEPT=1, a mensagem é copiada para o buffer de receção

Filtros de aceitação de mensagens e máscaras

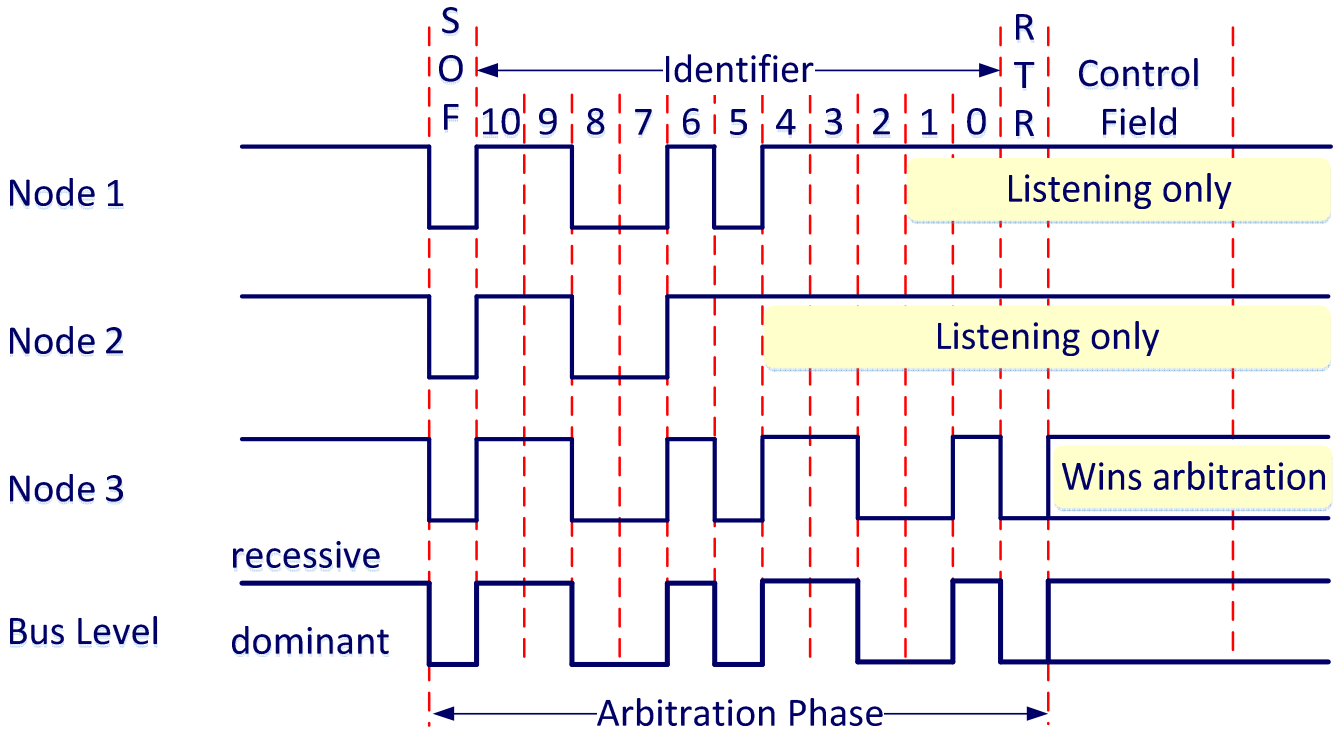


- ACCEPT = $ACCEPT_{10}$. $ACCEPT_9$. … . $ACCEPT_0$
- Se ACCEPT=1, a mensagem é copiada para o buffer de receção

- Exemplos (ID de 11 bits):
  - Máscara com o valor 0x000: todas as mensagens são aceites
  - Máscara com o valor 0x7FF, filtro com o valor 0x1F4: apenas a mensagem com o ID 0x1F4 é aceite
  - Máscara com o valor 0x7FC, filtro com o valor 0x230: são aceites as mensagens com os Ids 0x230, 0x231, 0x232 e 0x233

Controlo de acesso ao meio – Arbitragem

- Realizada durante os campos ID e RTR das tramas (*arbitration field*)
- Baseada em bit recessivo / bit dominante



- O nó produtor da mensagem com o identificador de menor valor binário ganha o processo de arbitragem e transmite os seus dados (um identificador com todos os bits a '0' tem a mais alta prioridade)

Aula 16

- A interface RS-232C
- Estrutura das tramas
- Codificação dos sinais
- Sincronização de relógio
- Tolerância na frequência dos relógios do emissor e do recetor

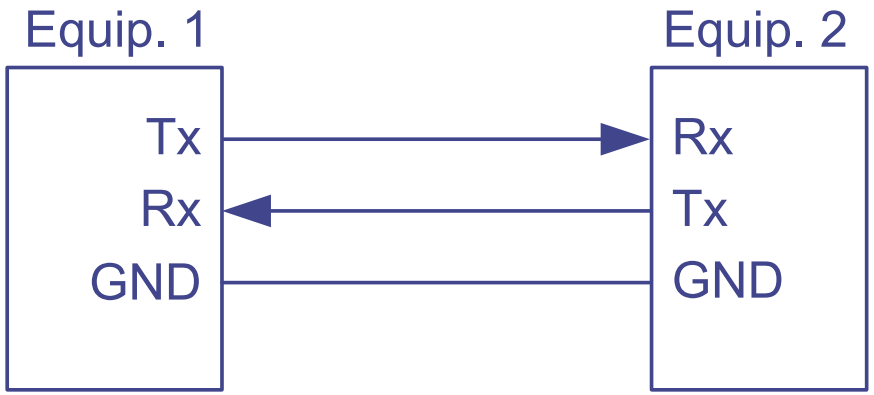
José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Introdução

- RS-232C – Standard (1969) para comunicação série assíncrona entre um Equipamento Terminal de Dados (DTE, e.g. computador) e um Equipamento de Comunicação de Dados (DCE, e.g. Modem)
- Permite comunicação bidirecional, *full-duplex*
- Transmissão orientada ao byte
- Conheceu uma grande utilização, que se estendeu muito para além do seu objetivo inicial (ligar DTEs a modems)
- Com o aparecimento do USB os computadores deixaram de disponibilizar comunicação RS-232C
- Por ser um modo de comunicação série muito fácil de implementar e de programar continua a ser muito usado em microcontroladores
- Apareceram no mercado conversores USB/RS-232C que permitem a ligação a PCs de equipamentos que implementam RS-232C

Sinalização

- Na sua forma mais simples, a implementação da norma RS-232C requer apenas a utilização de 2 linhas de sinalização e uma linha de massa



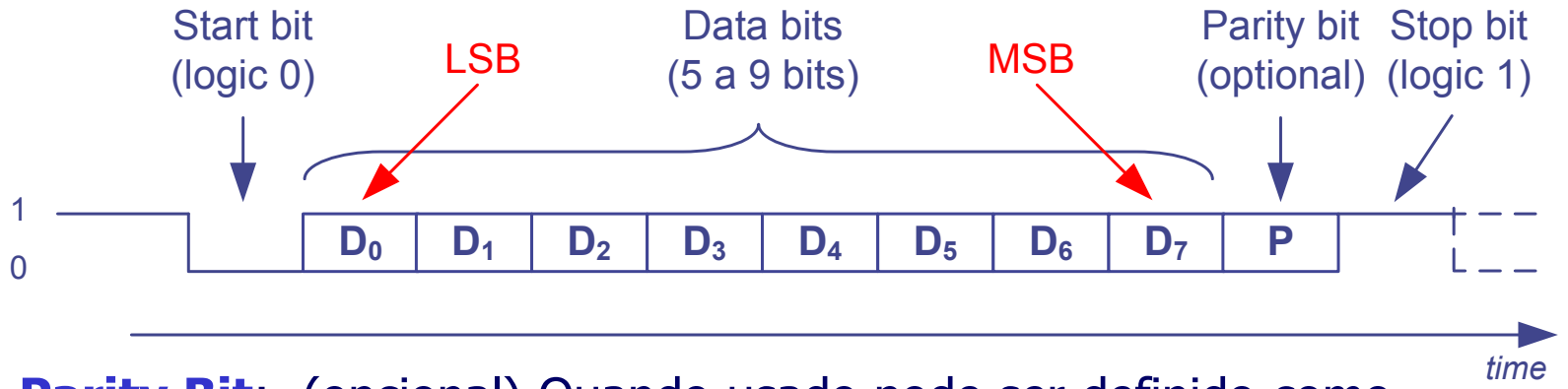
- Podem ser usadas linhas adicionais para protocolar a troca de informação entre os dois equipamentos (*handshake*)
  - RTS (*Request to send*)
  - CTS (*Clear to send*)
  - DTR (*Data terminal ready*)
  - DSR (*Data set ready*)

A norma original definia um total de 12 sinais (sendo 9 apenas para *handshaking*!)

Alguns problemas da norma RS-232C

- A nível físico (na linha de comunicação) os níveis lógicos são codificados com tensões simétricas (por exemplo +10V e -10V)
- Consumo de energia elevado
- Sinalização *single-ended*
  - Sinal é diferença entre tensão num fio e *common ground* (0V)
  - Baixa imunidade ao ruído
  - Impõe limitação na velocidade / distância
- Apenas suporta ligações ponto-a-ponto (implementações multi-ponto não standard)
- A norma era suficientemente vaga para permitir implementações proprietárias que, na prática, dificultavam ou mesmo impossibilitavam a interligação entre equipamentos de fabricantes diferentes

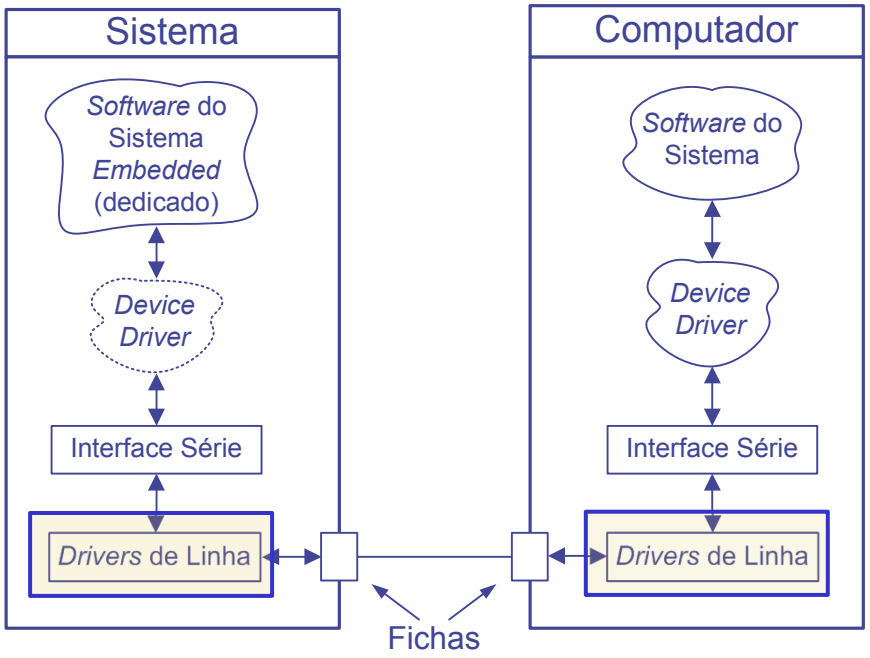
Estrutura de uma trama RS-232C

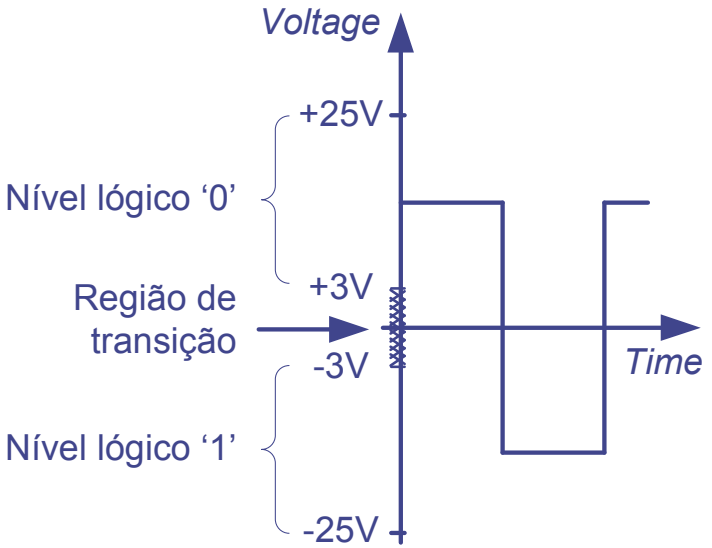


- **Parity Bit**: (opcional) Quando usado pode ser definido como **paridade PAR** ("*even*") ou **paridade ÍMPAR** ("*odd*"). É calculado pelo controlador de comunicação em cada trama:
  - Paridade PAR: o bit é obtido através do XOR de todos os bits do campo de dados. **Exemplo**: data bits – 10011000 -> *parity* bit: 1
  - Paridade ÍMPAR: o bit é obtido através do XNOR de todos os bits do campo de dados. **Exemplo**: data bits – 10011000 -> *parity* bit: 0
  - Permite detetar erros de comunicação sempre que houver um número ímpar de bits errados
- **Stop Bit(s)**: Podem ser usados 1 ou 2 bits.
  - Coincidem com o estado de linha inativa (*idle*)
  - Proporcionam um intervalo de tempo de guarda mínimo entre o envio consecutivo de dois valores

Camada física – codificação dos níveis lógicos

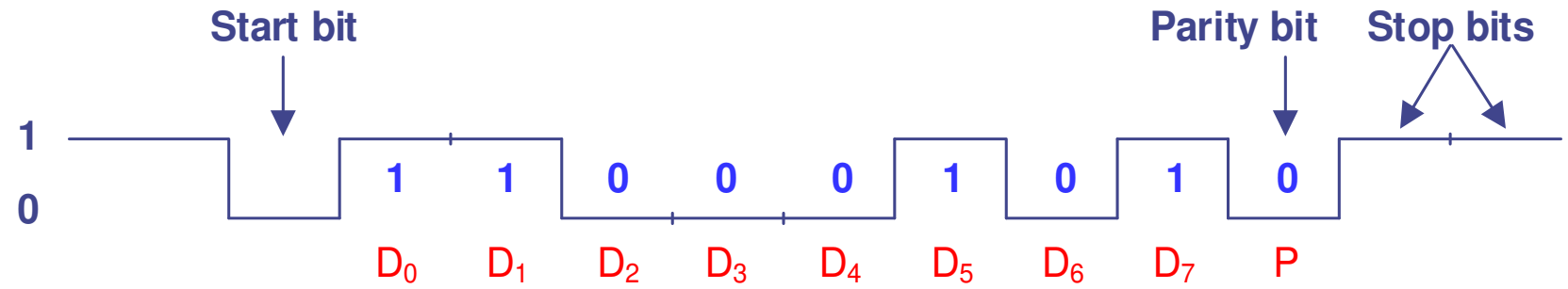
- Numa ligação física RS-232C os bits da trama são codificados em NRZ-L (*Non Return to Zero - Level*)
  - Nível lógico 1: codificado com uma tensão negativa (na gama -3V a -25V)
  - Nível lógico 0: codificado com uma tensão positiva (na gama +3V a +25V)
- A codificação e descodificação da trama com estes níveis de tensão é assegurada por circuitos eletrónicos designados por **drivers de linha**



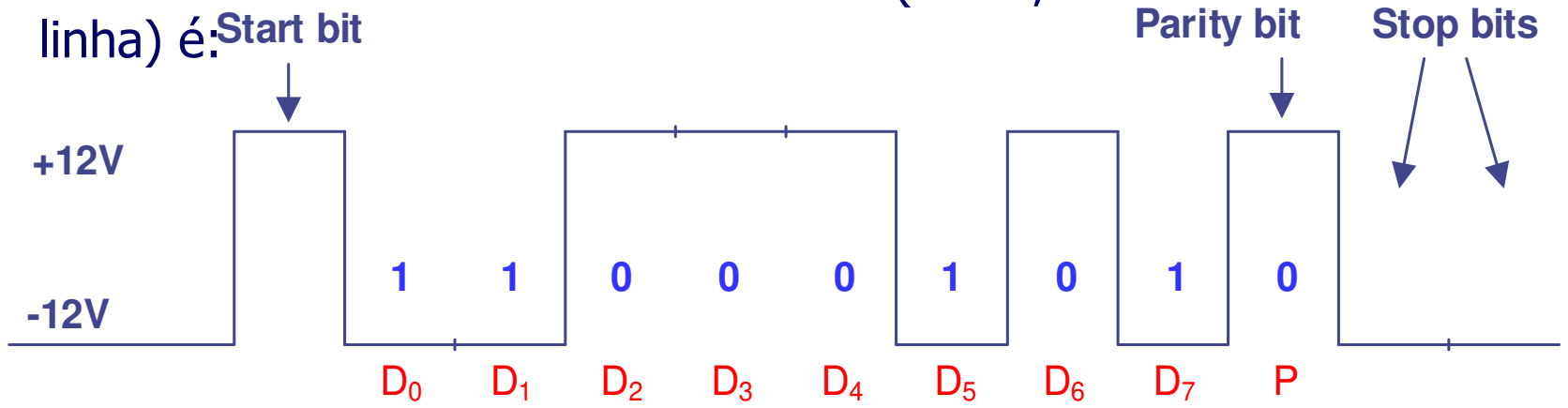


Exemplo: 8 bits de dados, 2 stop bits, paridade par

- A trama gerada pelo controlador de comunicação série RS-232C (e.g. UART-Universal asynchronous receiver/transmitter) para transmitir o valor 0xA3 é:



- A trama codificada em níveis RS-232C (isto é, à saída do driver de linha) é:

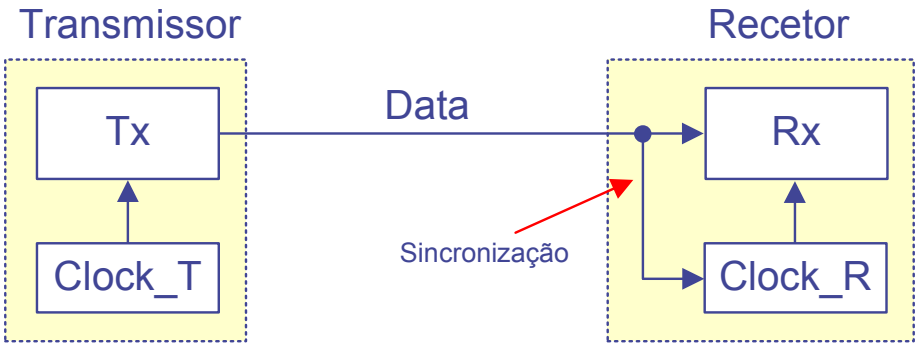


Baudrate (taxa de transmissão)

- O ***baudrate*** é, genericamente, o número de símbolos transmitidos por segundo. A cada símbolo pode corresponder um ou mais bits de dados
- A taxa de transmissão de dados bruta (*gross bit rate*) corresponde ao número de bits transmitidos por segundo (bps) (o *baudrate* não deve ser confundido com *gross bit rate*)
- No caso do RS-232C a cada símbolo está associado um único bit, logo o *baudrate* e o *gross bit rate* coincidem
- Exemplos comuns de *baudrates* em RS-232C [bps]: 600, 1200, 4800, 9600, 19200, 38400, 57600, 115200, 230400
- No exemplo anterior o número total de bits a serem transmitidos é 12
  - 1 start bit, 8 bits de dados, 1 bit de paridade, 2 stop bits
  - considerando um *baudrate* de 57600 bps a transmissão completa de uma trama demora ~208 μs (12 / 57600)
  - o bit rate líquido é (8 * 57600) / 12, i.e., 38400 bps
  - o byte rate é: 38400 / 8 = 4800 bytes/s

Receção de dados

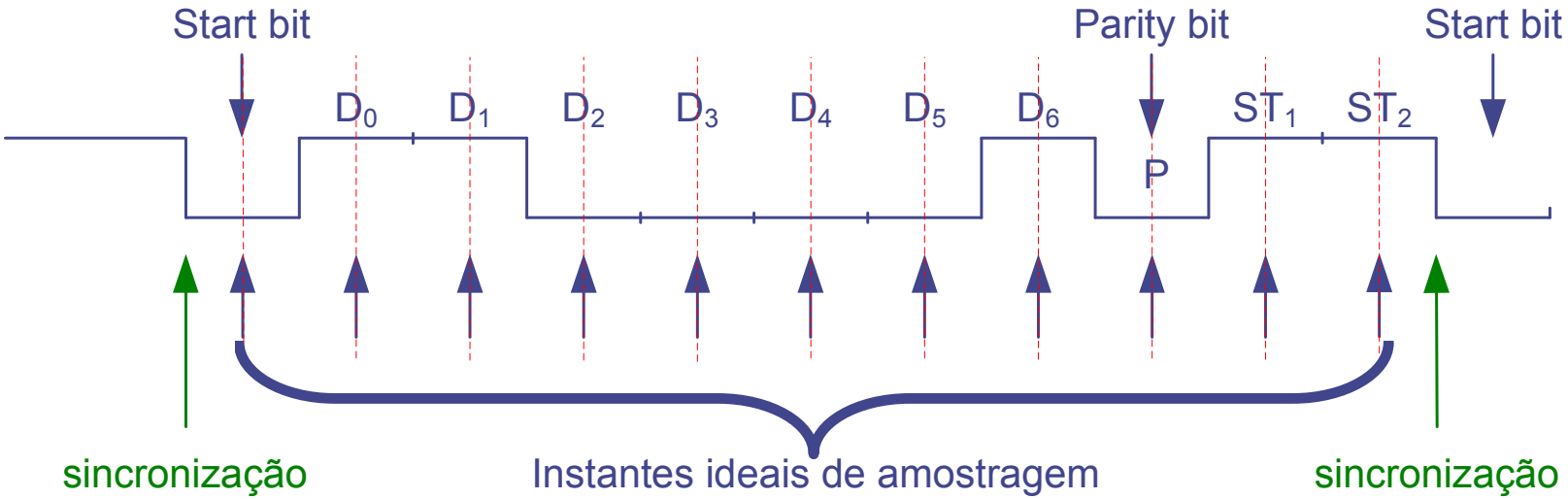
- Sincronização de relógio: **relógio implícito**
  - Comunicação assíncrona (i.e. não há transmissão do relógio)
  - O transmissor e o recetor têm relógios locais (independentes)



- A sincronização do processo de receção de dados é assegurada **no início da receção de cada nova trama** (sinalizado pelo **start bit**: transição de "1" para "0" na linha após um período de inatividade, por exemplo depois da receção completa de uma trama)
- Este método deve ser robusto, dentro de certos limites, a diferenças de frequência entre os relógios do transmissor e do recetor
  - Imprecisão na geração do relógio
  - Constante de divisão dos timers (que geram o relógio) não inteiras

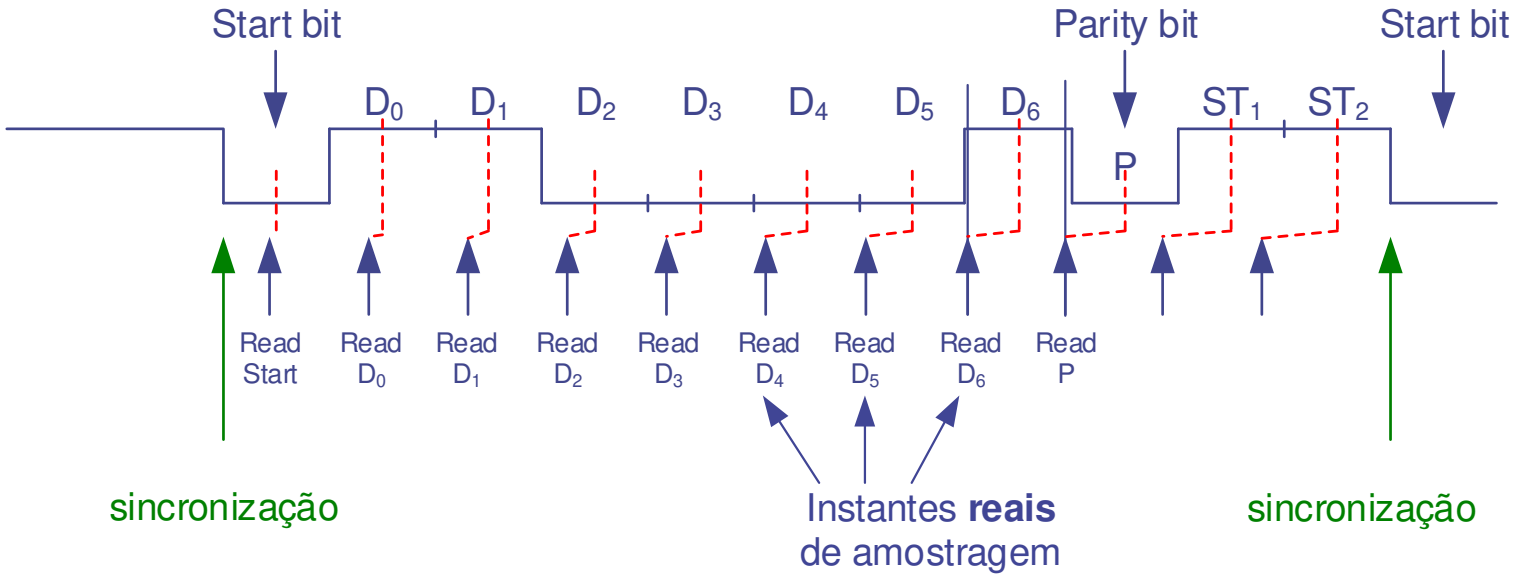
Receção de dados

- Para que a comunicação se processe corretamente, o transmissor e o recetor têm que estar **configurados com os mesmos parâmetros**:
  - Estrutura da trama: **nº de bits de dados**, **tipo de paridade**, **número de stop bits**
  - **Baudrate** (relógios com a mesma frequência)
- O recetor deve sincronizar-se pelo flanco negativo (transição do nível lógico "1" para o nível lógico "0") da linha (Start bit) e, idealmente, **fazer as leituras a meio do intervalo reservado a cada bit**
- Exemplo da receção do valor 0x43: estrutura da trama 7, O, 2 (7 data bits, odd parity, 2 stop bits)



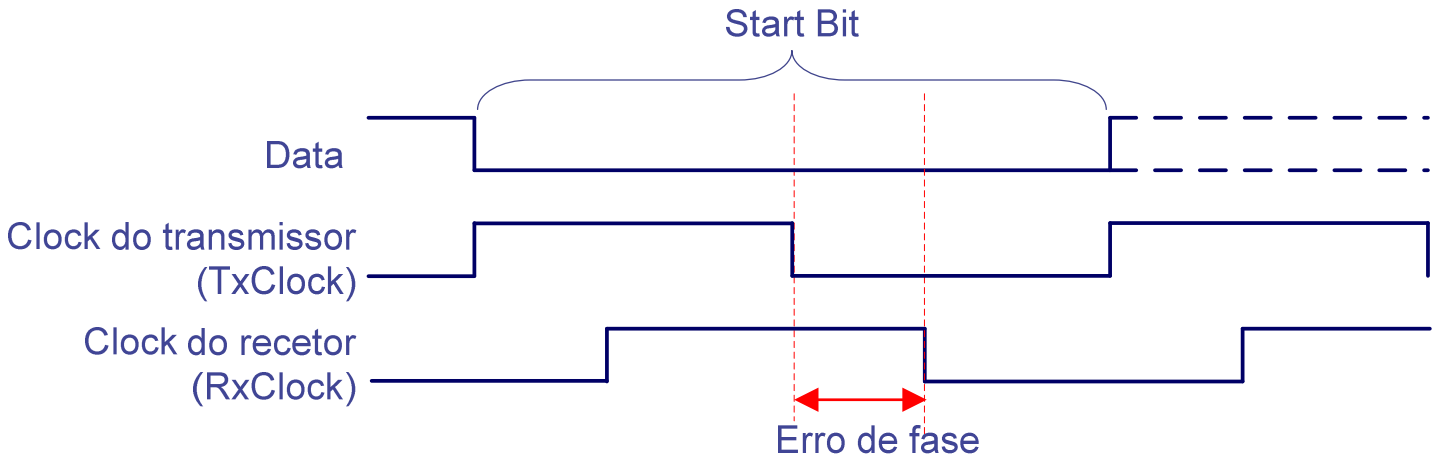
Sincronização

- Entre instantes de sincronização o desvio dos relógios depende da estabilidade/precisão dos relógios do transmissor e do recetor
- Exemplo em que a receção não é corretamente efetuada devido a um desvio da frequência dos relógios do transmissor e do recetor



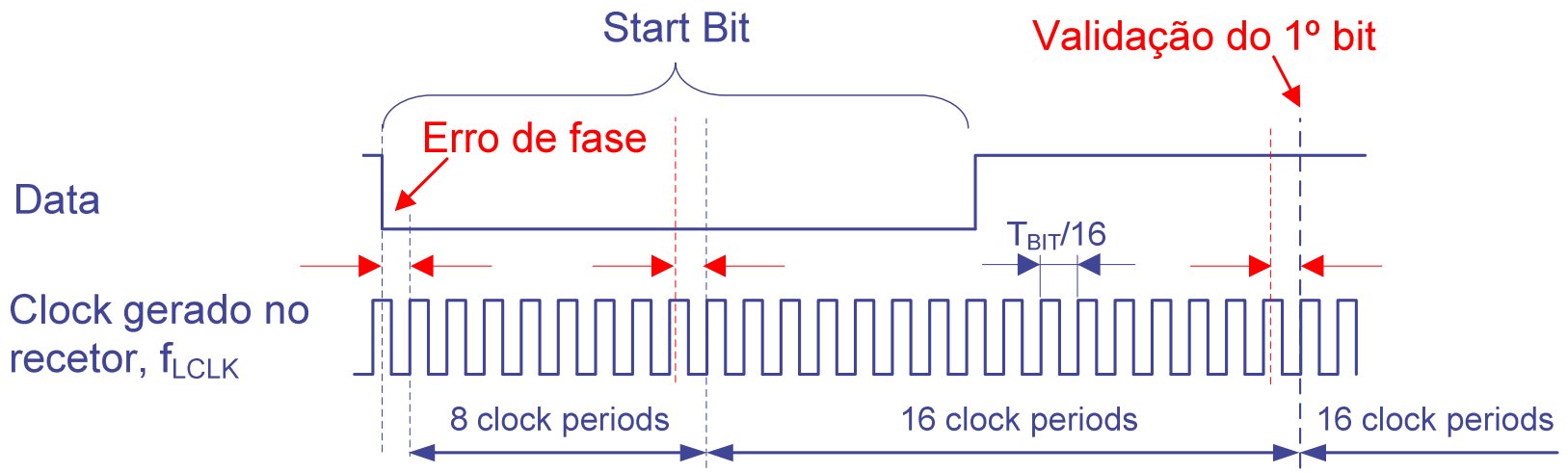
- Neste exemplo o recetor detetaria dois erros:
  - **Erro de paridade** (se o bit D6 for detetado como 1): o bit de paridade devia ser 0 e é lido como 1
  - **Erro de *framing***: é detetado o nível lógico 0 no instante em que era esperado um stop bit (nível lógico 1)

Sincronização



- Mesmo que a frequência dos relógios do transmissor e do recetor seja a mesma, subsiste o erro de fase que pode impedir a correta validação da informação (idealmente a meio do "tempo de bit")
- Sincronizar a fase do relógio do recetor com a do transmissor é tecnicamente complicado
- Em vez disso, é mais simples gerar no recetor um relógio com uma frequência N vezes superior ao relógio do transmissor e sincronizar a receção a partir desse relógio (designado a seguir por f_{LCLK})

Sincronização



- Por exemplo, se N = 16, o erro de fase máximo desse relógio, relativamente ao aparecimento do start, é T/16, em que T é o período do relógio do transmissor (ou seja, o "tempo de bit", T_{BIT})
- O erro de fase mantém-se até ao fim da receção da trama corrente, mas os instantes de validação estão bem definidos:
  - "Start bit", validado ao fim de 8 ciclos de relógio
  - Restantes bits validados a cada 16 ciclos de relógio

Sincronização

- O relógio local (**LCLK**) deverá então ter, idealmente, uma frequência igual a:

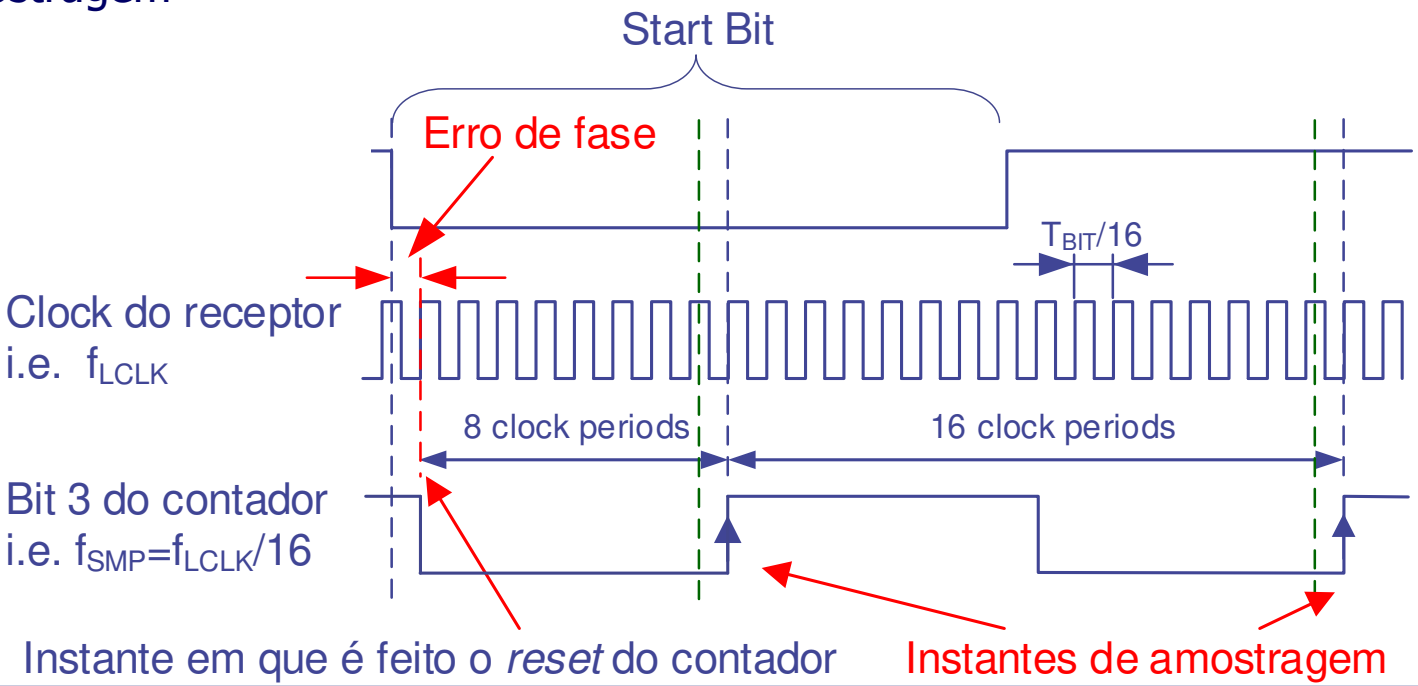
$$f_{LCLK} = N * f_{TCLK}$$

em que $f_{TCLK} = 1/T_{BIT}$ é a frequência do relógio de transmissão

- N é normalmente designado por **fator de sobreamostragem**
- Valores típicos de N: 4, 16, 64
- Esse relógio não é sincronizado com o sinal da linha, logo impõe um erro de fase (que é sempre inferior a um período, $\Delta_1 < T_{LCLK}$)
- Utilizando um relógio com N=16 o erro de fase máximo é $T_{BIT}/16$. Para N=64, o erro de fase máximo será $T_{BIT}/64$
- A utilização de fatores de sobreamostragem elevados nem sempre é possível (frequência da fonte de relógio disponível, constante de divisão do timer)

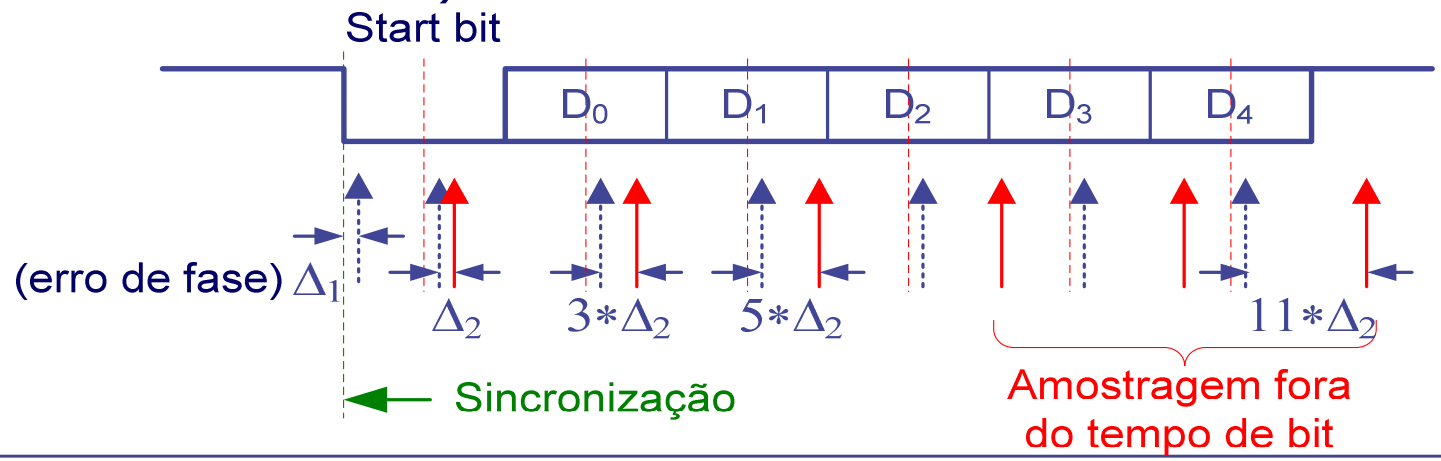
Sincronização – exemplo de implementação

- Considerando um fator de sobreamostragem (N) de 16, o tempo de bit equivale a 16 períodos do relógio local ($16T_{LCLK}$). O erro de fase máximo é $T_{BIT}/16$
- f_{SMP} é a frequência usada pelo recetor para a amostragem de cada bit da trama
- Usando um contador de 4 bits (com o relógio f_{LCLK}) como divisor de frequência por 16 (i.e. $f_O=f_{SMP}=f_{LCLK}/16$) a sincronização é trivial: basta fazer o reset (síncrono) desse contador quando é detetado o start bit
- As transições ascendentes do bit 3 do contador (MSBit) definem os instantes de amostragem



Sincronização – erro do instante de amostragem

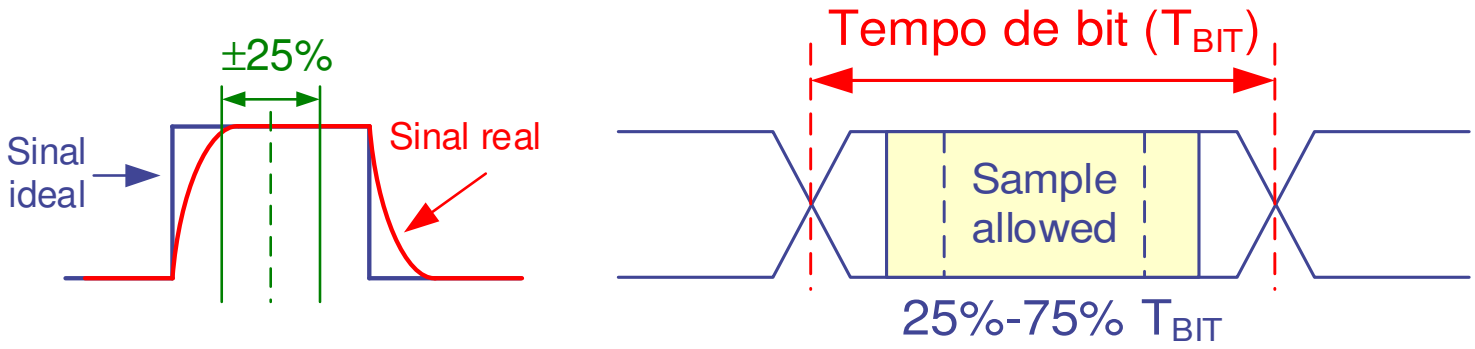
- Os erros nos instantes de amostragem podem ter duas causas distintas:
  - **Erro de fase** (Δ_1): erro cometido ao determinar o instante inicial de sincronização
  - **Erro provocado por desvio de frequência** (Δ_2): a frequência dos relógios do transmissor e do recetor não são exatamente iguais (e.g. tolerância dos cristais de quartzo dos osciladores, constantes de divisão dos timers). Este erro é cumulativo e proporcional ao comprimento da trama (no caso da figura abaixo, no bit D_4 o erro é $11*\Delta_2$)
- O efeito cumulativo destas fontes de erro pode originar erros na receção (como vimos no slide 11)



Máximo desvio de frequência entre emissor e recetor

S

- É comum considerar-se como zona segura de amostragem do bit:
  - **Pior caso** (cabos longos com efeito capacitivo pronunciado, velocidades de transmissão elevadas, ...): **±25%** do tempo de bit, em torno do instante ideal de amostragem, i.e., a meio
  - **Caso ideal** (cabos curtos e de acordo com as especificações, velocidades moderadas, ...): **±37,5%** do tempo de bit, em torno do instante ideal de amostragem



Máximo desvio de frequência entre emissor e recetor

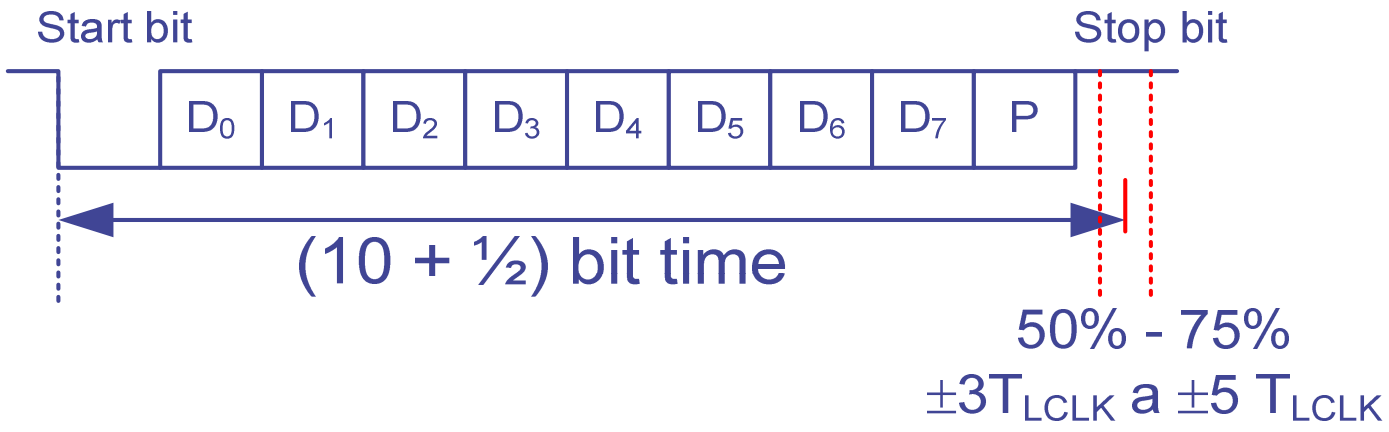
- Considerando um fator de sobreamostragem (N) de 16, o tempo de bit equivale a 16 períodos do relógio local ($\mathbf{T_{BIT} = 16T_{LCLK}}$)
- Assim, o desvio máximo admitido no instante de amostragem de um bit é de $\mathbf{\pm 4T_{LCLK}}$ (pior caso, ±0.25*16 ciclos) a $\mathbf{\pm 6T_{LCLK}}$ (caso ideal, ±0.375*16 ciclos)

- Como há um erro intrínseco máximo de $\mathbf{1T_{LCLK}}$ devido ao erro de fase, então o desvio máximo aceitável, resultante da diferença de frequência dos relógios, é de $\mathbf{\pm 3T_{LCLK}}$ a $\mathbf{\pm 5T_{LCLK}}$ (em cada instante de amostragem)

Esta correção do erro de fase para a obtenção do desvio máximo aceitável é conservativa. Porquê?

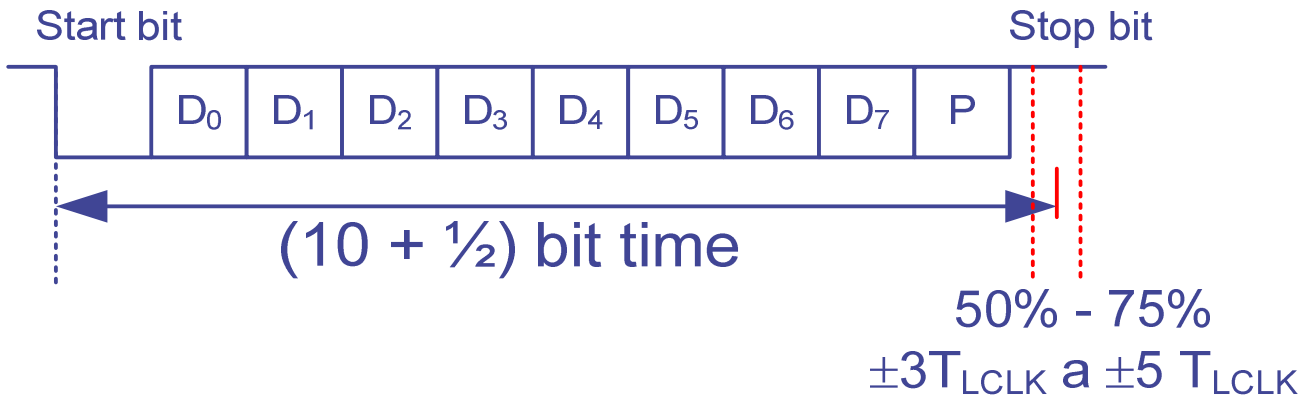
Máximo desvio de frequência entre emissor e recetor

- Ao contrário do erro de fase, o erro provocado por desvio de frequência é cumulativo e diretamente proporcional ao comprimento da trama
- Por outro lado é necessário garantir que o último bit da trama (independentemente do comprimento da trama) é sempre amostrado dentro da zona segura (**$\pm 3T_{LCLK}$** a **$\pm 5T_{LCLK}$** , relativamente ao instante ideal de amostragem)



Máximo desvio de frequência entre emissor e recetor

- Consideremos então o caso em que pretendemos amostrar uma das tramas mais longas (start bit, 8 data bits, bit de paridade e 1 stop bit).



- Continuando a admitir que o relógio local tem um fator de sobreamostragem de N=16, para amostrar esta trama serão então necessários 10.5*16=168 períodos do relógio local
- Assim, a máxima discrepância que poderá ser tolerada entre os relógios do transmissor e do recetor é $\Delta T = \pm 3/168 \approx \pm 1.8\%$ (pior caso) a $\Delta T = \pm 5/168 \approx \pm 3.0\%$ (caso "ideal")

Máximo desvio de frequência entre emissor e recetor

S

- Como vimos, a máxima discrepância que poderá ser tolerada entre os relógios do transmissor e do recetor é **$\Delta T = \pm 3/168 \approx \pm 1.8\%$** (pior caso) a **$\Delta T = \pm 5/168 \approx \pm 3.0\%$** (caso "ideal") para as condições indicadas.
- **Exemplo**: vamos assumir que a taxa de transmissão é de 115200 bps, 8 data bits, parity bit, 1 stop bit e N=16. Para os dois casos-limite, para que a comunicação se processe sem erros, qual deve ser a gama de frequência do relógio do recetor?
- $T_{LCLK} = \frac{1}{(16*115200)} \gg f_{LCLK} = 1.843.200\ Hz$ (valor ideal)

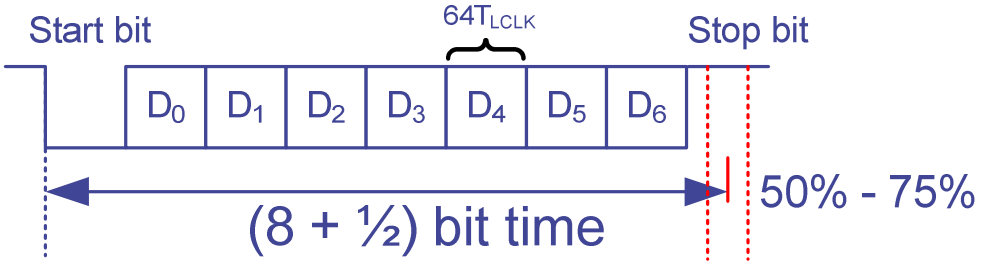
  $T_{LCLK} = \frac{(1 \pm 0.018)}{(16*115200)} \gg f_{LCLK} \in [1.810.609, 1.876.986]\ Hz$ (pior caso)

  $T_{LCLK} = \frac{(1 \pm 0.03)}{(16*115200)} \gg f_{LCLK} \in [1.798.515, 1.900.206]\ Hz$ (melhor caso)

- **Exercício:** Admita que a configuração da comunicação é 38400 bps, 7 bits sem paridade, 1 stop bit e N = 64. Calcule o valor de frequência ideal no recetor e os intervalos admissíveis dessa frequência para os casos limite (tente resolver sozinho. A resolução está na versão em PDF)

Exercício: resolução

S



- Intervalo de validação (melhor caso): $\pm\left(\left(\frac{75\%}{2} * 64\right) - 1\right) = \pm 23 T_{LCLK}$
- Intervalo de validação (pior caso): $\pm\left(\left(\frac{50\%}{2} * 64\right) - 1\right) = \pm 15 T_{LCLK}$
- Nº de períodos de relógio para amostrar a trama $= 8.5 * 64 = 544$
- Máxima discrepância temporal (melhor caso): $\frac{\pm 23}{544} \approx \pm 4.3\%$
- Máxima discrepância temporal (pior caso): $\frac{\pm 15}{544} \approx \pm 2.76\%$
- $T_{LCLK} = \frac{1}{(64*38400)} \gg f_{LCLK} = 2.457.600\ Hz$ (valor ideal)
- $T_{LCLK} = \frac{(1 \pm 0.0276)}{(64*38400)} \gg f_{LCLK} \in [2.391.592, 2.527.355]\ Hz$ (pior caso)
- $T_{LCLK} = \frac{(1 \pm 0.043)}{(64*38400)} \gg f_{LCLK} \in [2.356.280, 2.568.025]\ Hz$ (melhor caso)

Aula 17

- *Device drivers*
- Princípios gerais
- Caso de estudo: *device driver* para uma UART
- Princípio de operação e estruturas de dados
- Funções de interface com a aplicação
- Funções de serviço de interrupções e interface com o hardware

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Introdução

- O **número** de **periféricos** existentes é muito **vasto**:
  - Teclado, rato, placas (gráfica, rede, som, etc.), disco duro, *pen drive*, scanner, câmara de vídeo, etc.

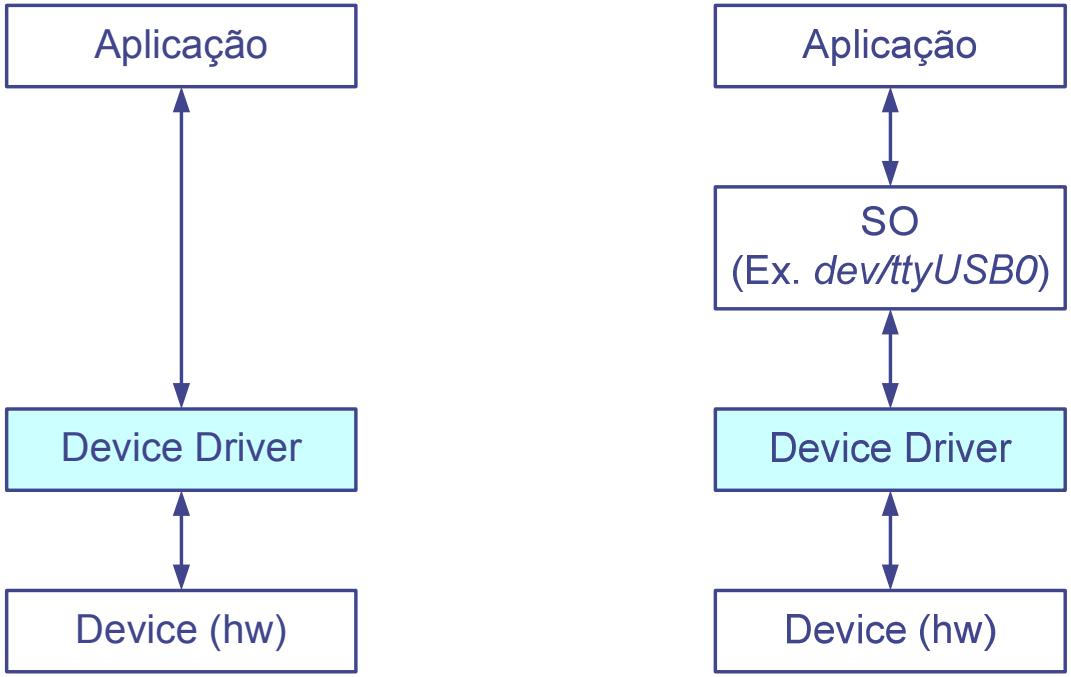
- Estes periféricos apresentam características distintas:
  - **Operações suportadas**: leitura, escrita, leitura e escrita
  - **Modo de acesso**: carater, bloco, etc.
  - **Representação da informação**: ASCII, UNICODE, *Little/Big Endian*, etc.
  - **Largura de banda**: alguns bytes/s a MB/s
  - **Recursos utilizados**: portos (I/O, *memory mapped*), interrupções, DMA, etc.
  - **Implementação**: diferentes dispositivos de uma dada classe podem ser baseados em implementações distintas (e.g. diferentes fabricantes, diferentes modelos) com reflexos profundos na sua operação interna

Introdução

- As aplicações/Sistemas Operativos (SO) **não podem conhecer todos os tipos de dispositivos** passados, atuais e futuros com um nível de detalhe suficiente para realizar o seu controlo a baixo nível!
- **Solução**: Criar uma camada de abstração que permita o acesso ao dispositivo de forma independente da sua implementação
- ***Device driver***
  - Um programa que permite a outro programa (aplicação, SO) interagir com um dado dispositivo de hardware
  - Implementa a camada de abstração e lida com as particularidades do dispositivo controlado
  - Como o *Device Driver* tem de lidar com os aspetos específicos da implementação física, o seu fornecimento é assegurado pelo fabricante
- **Aspetos-chave:**
  - Abstração, uniformização de acesso, independência entre aplicações/SO e o hardware

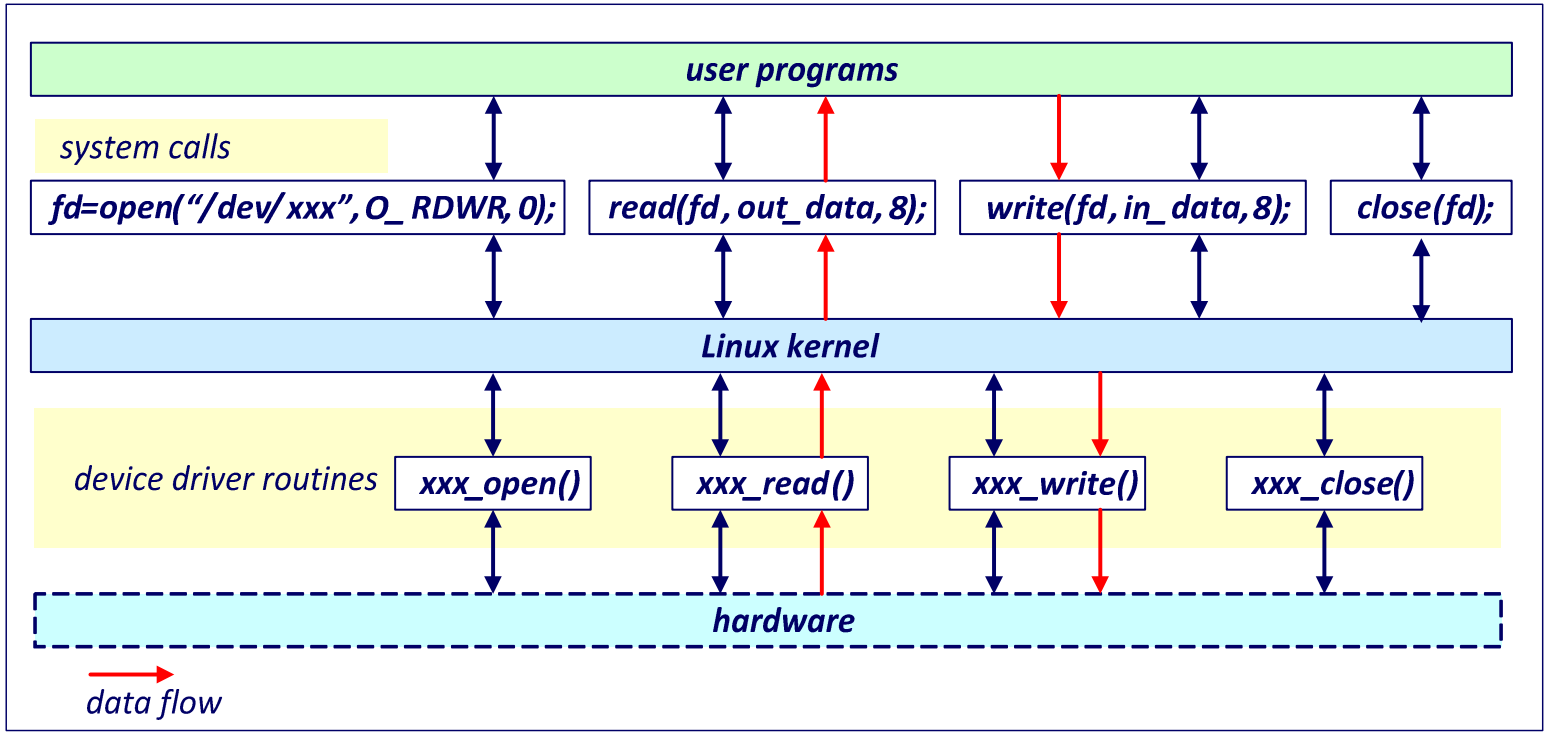
Princípios gerais

- O acesso, por parte das aplicações, a um *device driver* é diferente num sistema embutido e num sistema computacional de uso geral (com um Sistema Operativo típico, e.g. Linux, Windows, Mac OS):
  - Aplicações em sistemas embutidos acedem, tipicamente, de forma direta aos *device drivers*
  - Aplicações que correm sobre SO acedem a funções do SO (*system calls*); o kernel do SO, por sua vez, acede aos *device drivers*



Princípios gerais

- O Sistema Operativo especifica classes de dispositivos e, para cada classe, uma interface que estabelece como é realizado o acesso a esses dispositivos
  - A função do *device driver* é traduzir as chamadas realizadas pela aplicação/SO em ações específicas do dispositivo
  - Exemplos de classes de dispositivos: interface com o utilizador, armazenamento de massa, comunicação, ...



Exemplo de um *device driver*: comunicação série

- Admitindo que é fornecida uma biblioteca que apresenta a seguinte interface:
  - `void comDrv_init(int baudrate, char dataBits, char parity, char stopBits);`
  - `char comDrv_getc(void);  // read a character`
  - `void comDrv_putc(char ch);  // write a character`
  - `void comDrv_close(void);`

- **Do ponto de vista da aplicação**:
  - Do ponto de vista funcional é relevante qual o modelo/fabricante do dispositivo de comunicação série?
  - Se o dispositivo de comunicação for substituído por outro com arquitetura interna distinta, sendo fornecida uma biblioteca com interface compatível, é necessário alterar a aplicação?

Caso de estudo

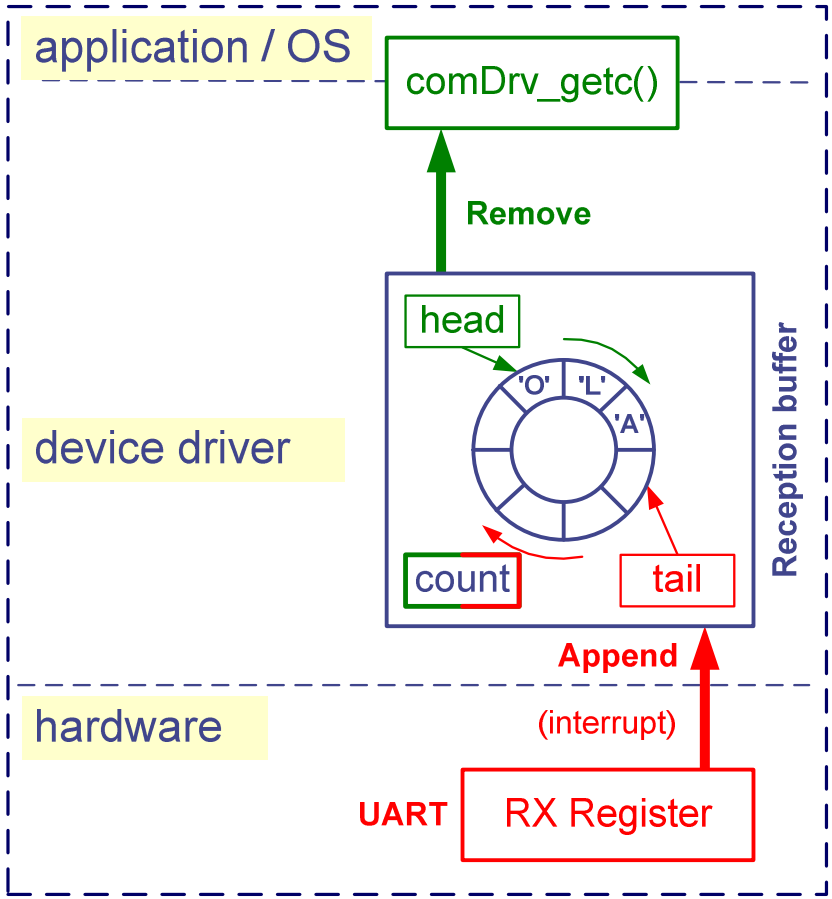
- Aspetos-chave da implementação de um *device driver* para uma UART RS232 (*Universal Asynchronous Receiver Transmitter*) para executar num sistema com microcontrolador (i.e., sem sistema operativo)

- Princípio de operação
  - Desacoplamento da transferência de dados entre a UART e a aplicação, realizada por meio de FIFOs (um FIFO de transmissão e um de receção). Do ponto de vista da aplicação:
    - A **transmissão** consiste em copiar os dados a enviar para o FIFO de transmissão do *device driver*
    - A **receção** consiste em ler os dados recebidos que residem no FIFO de receção do *device driver*
  - A transferência de dados entre os FIFOS e a UART é realizada por interrupção, i.e., sem intervenção explícita da aplicação

  - Um FIFO pode ser implementado através de um *buffer* circular

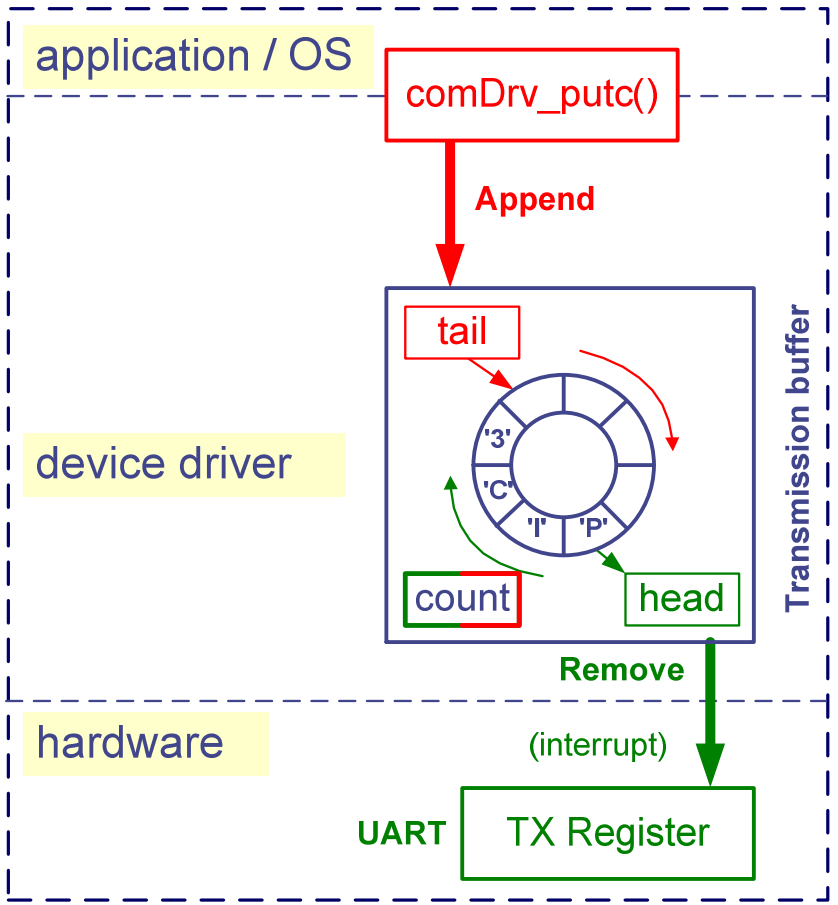
Princípio de operação – receção

- "**tail**" – posição do buffer circular onde a **rotina de serviço à interrupção escreve** o próximo caracter lido da UART
- "**head**" – posição do buffer circular de onde a função **comDrv_getc()** lê o próximo caracter
- "**count**" – número de caracteres residentes no buffer circular (ainda não lidos pela aplicação)
- **O acesso à variável "count" tem que ser feito numa secção crítica do código. Porquê?**



Princípio de operação – transmissão

- "**tail**" – posição do buffer circular onde a função **comDrv_putc()** escreve o próximo caracter
- "**head**" – posição do buffer circular de onde a **rotina de serviço à interrupção lê** o próximo caracter a enviar para a UART
- "**count**" – número de caracteres residentes no buffer circular (ainda não enviados para a UART)
- **O acesso à variável "count" tem que ser feito numa secção crítica do código. Porquê?**



Implementação – FIFO

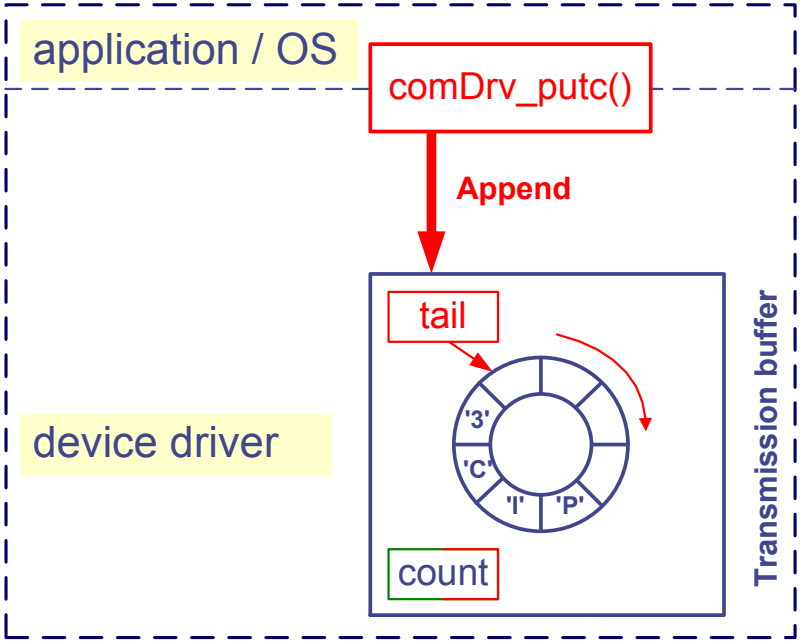
- FIFO - Buffer circular implementado através de um *array* linear:

```
#define BUF_SIZE 32

typedef struct
{
  unsigned char data[BUF_SIZE];
  unsigned int head;
  unsigned int tail;
  unsigned int count;
} circularBuffer;

circularBuffer txb; // Transmission buffer
circularBuffer rxb; // Reception buffer
```

- A constante "BUF_SIZE" deve ser definida em função das necessidades previsíveis de pico de tráfego.
- Se "BUF_SIZE" for uma potência de 2 simplifica a atualização dos índices do buffer circular (podem ser encarados como contadores módulo 2^N e podem ser geridos com uma simples máscara)

Implementação – Função de transmissão

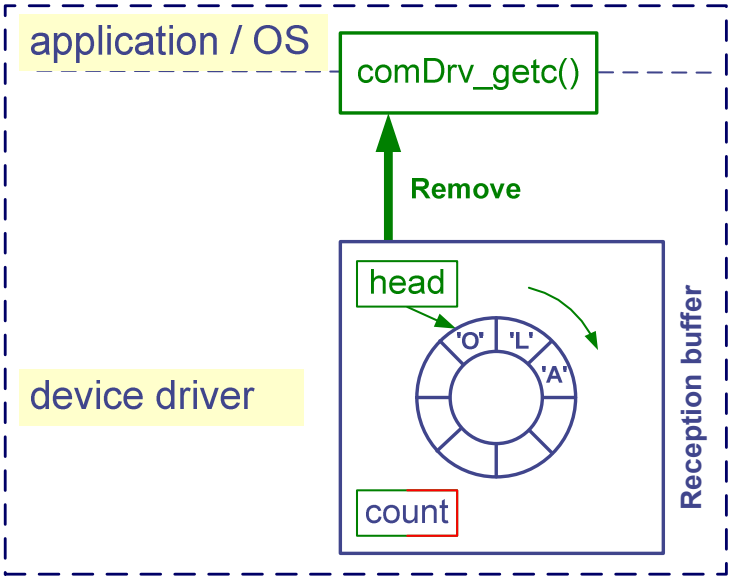
- A função de transmissão, evocada pela aplicação, copia o caracter para o **buffer de transmissão** (posição "**`tail`**"), incrementa o índice "**`tail`**" e o contador




```
void comDrv_putc(char ch)
{
  Wait while buffer is full (txb.count==BUF_SIZE)
  Copy "ch" to the buffer ("tail" position)
  Increment "tail" index (mod BUF_SIZE)
  Increment "count" variable
}
```

Implementação – Função de receção

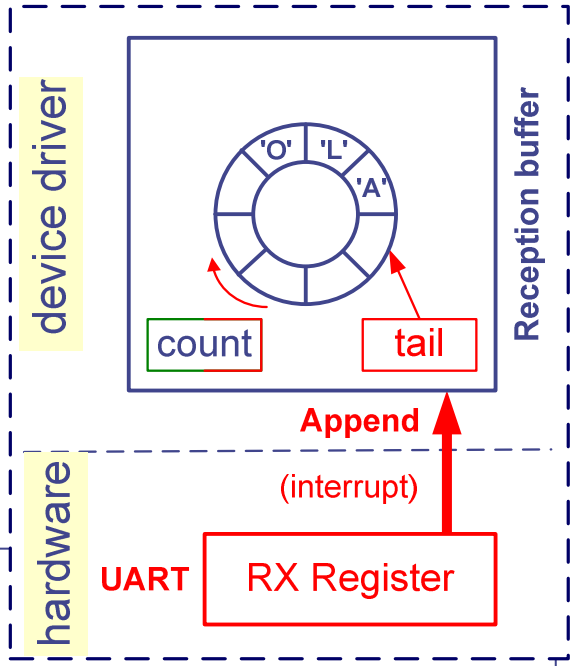
- A função de receção, evocada pela aplicação, verifica se há caracteres no **buffer de receção** para serem lidos e, caso haja, retorna o caracter presente na posição "**head**", incrementa o índice "**head**" e decrementa o contador




```
int comDrv_getc(char *pchar)
{
  If "count" variable is 0 then return false
  Copy character at position "head" to *pchar
  Increment "head" index (mod BUF_SIZE)
  Decrement "count" variable
  return true;
}
```

Implementação – RSI de receção

- A rotina de serviço à interrupção da receção é executada sempre que a UART recebe um novo caracter
- O caracter recebido pela UART deve então ser copiado para o **buffer de receção**, na posição "**`tail`**"; a variável "**`count`**" deve ser incrementada e o índice "**`tail`**" deve ser igualmente incrementado

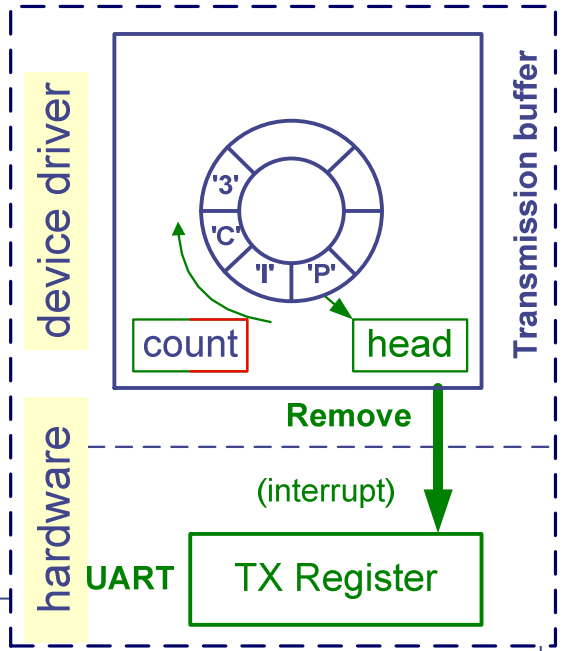



```
void interrupt isr_rx(void)
{
  Copy received character from UART to RX
     buffer ("tail" position)
  Increment "tail" index (mod BUF_SIZE)
  Increment "count" variable
}
```

O que acontece no caso em que o *buffer* circular está cheio e a UART recebe um novo caracter? Como resolver esse problema?

Implementação – RSI de transmissão

- A UART gera, normalmente, uma interrupção de transmissão quando tiver disponibilidade para transmitir um novo caracter
- As tarefas a implementar na respetiva rotina de serviço à interrupção são:
  - Se o número de caracteres no **buffer de transmissão** for maior que 0 ("`count`" > 0), copiar o conteúdo do buffer na posição "`head`" para a UART
  - Decrementar a variável "`count`" e incrementar o índice "`head`"




```
void interrupt isr_tx(void)
{
  If "count" > 0 then {
    Copy character from TX buffer ("head") to UART
    Increment "head" index (mod BUF_SIZE)
    Decrement "count" variable
  }
  If "count" == 0 then disable UART TX interrupts
}
```

Atualização do TX "count" - Secção crítica

```
void comDrv_putc(char ch)
{
  Wait while buffer is full (count==BUF_SIZE)
  Copy "ch" to the transmission buffer ("tail")
  Increment "tail" index (mod BUF_SIZE)
  Disable UART TX interrupts
  Increment "count" variable
  Enable UART TX interrupts
}
```

Secção crítica ("count" é um recurso partilhado)

Se a UART estiver disponível para transmitir, desencadeia a imediata geração da interrupção de transmissão

```
void interrupt isr_tx(void)
{
  if "count" > 0 then
  {
    Copy character from TX buffer ("head") to UART
    Increment "head" index (mod BUF_SIZE)
    Decrement "count" variable
  }
  if "count" == 0 then disable UART TX interrupts
}
```

Atualização do RX "count" - Secção crítica

```
int comDrv_getc(char *pchar)
{
  If "count" variable is 0 then return false
  Copy character at position "head" to *pchar
  Increment "head" index (mod BUF_SIZE)
  Disable UART RX interrupts
  Decrement "count" variable
  Enable UART RX interrupts
  return true;
}
```

Secção crítica ("count" é um recurso partilhado)

```
void interrupt isr_rx(void)
{
  Copy received character from UART to RX buffer
     ("tail" position)
  Increment "tail" index (mod BUF_SIZE)
  Increment "count" variable
}
```

Aulas 18 e 19

- Tecnologias de memória
- Organização genérica de um circuito de memória a partir de uma célula básica
- Memória SRAM *(Static Random Access Memory)*:
  - organização de células básicas num array
  - ciclos de acesso para leitura e escrita: diagramas temporais
  - construção de módulos de memória SRAM
- Memória DRAM *(Dynamic Random Access Memory)* :
  - célula básica; organização interna
  - ciclos de acesso para leitura e escrita: diagramas temporais
  - refrescamento: modo "RAS only"
  - construção de módulos de memória DRAM

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Introdução – conceitos básicos

- RAM – Random Access Memory
  - Designação para memória volátil que pode ser lida e escrita
  - Acesso "random"

- ROM – Read Only Memory
  - Memória não volátil que apenas pode ser lida
  - Acesso "random"

(Acesso "random" - tempo de acesso é o mesmo para qualquer posição de memória)

Introdução – conceitos básicos

- Tecnologias:
  - Semicondutor
  - Magnética
  - Ótica
  - Magneto-ótica
- Memória volátil:
  - Informação armazenada perde-se quando o circuito é desligado da alimentação: RAM (SRAM e DRAM)
- Memória não volátil:
  - A informação armazenada mantém-se até ser deliberadamente alterada: EEPROM, Flash EEPROM, tecnologias magnéticas

Memória não volátil – evolução histórica

- **ROM** – programada durante o processo de fabrico (1965)
- **PROM** – Programmable Read Only Memory: programável uma única vez (1970)
- **EPROM** – Erasable PROM: escrita em segundos, apagamento em minutos (ambas efectuadas em dispositivos especiais) (1971)
- **EEPROM** – Electrically Erasable PROM (1976)
  - O apagamento e a escrita podem ser efetuados no próprio circuito em que a memória está integrada
  - O apagamento é feito byte a byte
  - Escrita muito mais lenta que leitura
- **Flash EEPROM** (tecnologia semelhante à EEPROM) (1985)
  - A escrita pressupõe o prévio apagamento das zonas de memória a escrever
  - O apagamento é feito por blocos (por exemplo, blocos de 4 kB) o que torna esta tecnologia mais rápida que a EEPROM
  - O apagamento e a escrita podem ser efetuados no próprio circuito em que a memória está integrada
  - Escrita muito mais lenta que leitura

Tecnologias de memória

Tecnologia	Tempo Acesso	$ / GB
SRAM	0,5 – 2,5 ns	$500 - $1000
DRAM	35 - 70 ns	$10 - $20
Flash	5 – 50 us	$0,75 - $1
Magnetic Disk	5 - 20 ms	$0,005 - $0,1

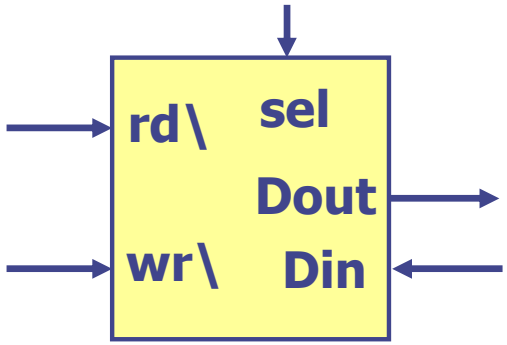
(Dados de 2012)

- SRAM - Static Random Access Memory
- DRAM - Dynamic Random Access Memory

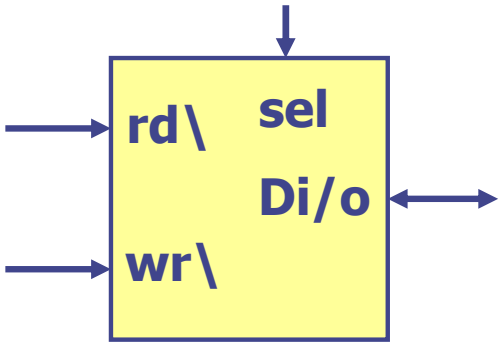
- Dadas estas diferenças de custo e de tempo de acesso, é vantajoso construir o sistema de memória como uma hierarquia onde se utilizem todas estas tecnologias

Organização básica de memória

- Uma memória pode ser encarada como uma coleção de N registos de dimensão K (N x K)
- Cada registo é formado por K células, cada uma delas capaz de armazenar 1 bit
- Uma célula de memória (de 1 bit) pode ser representada por:



Din – Data In (1 bit)
Dout – Data Out (1 bit)
sel – Select
rd\ – Read\
rw\ – Write\

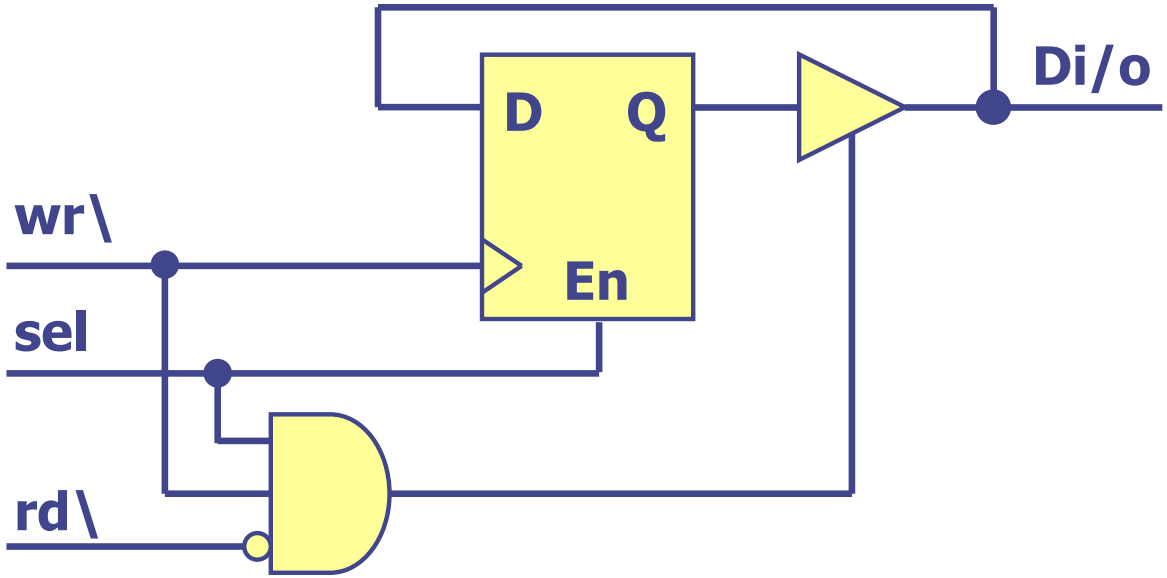


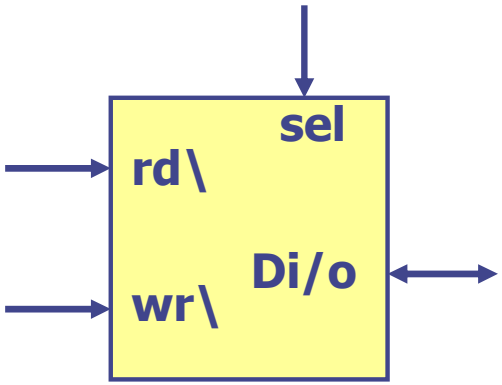
Di/o – Data In/Out (1 bit)
sel – Select
rd\ – Read\
rw\ – Write\

Organização básica de memória

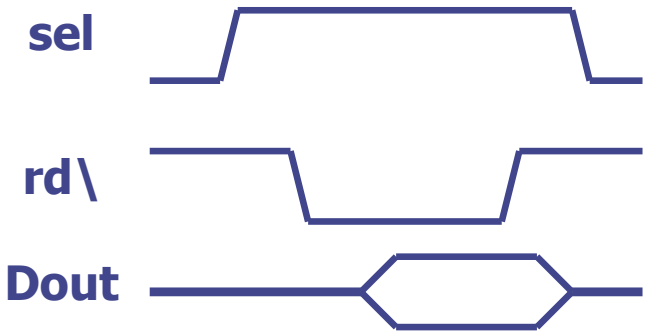
S

- Uma possível implementação de uma célula de memória é:

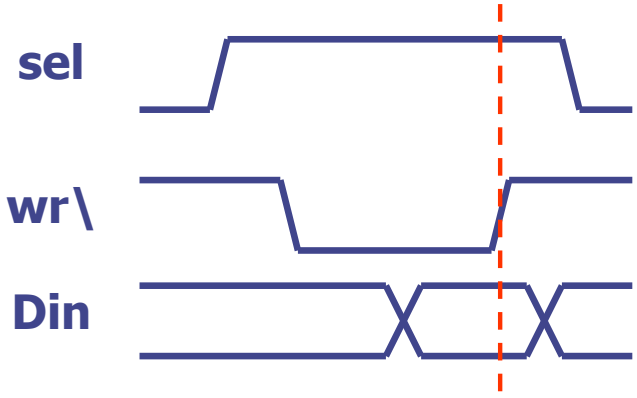




Operação de leitura



Operação de escrita



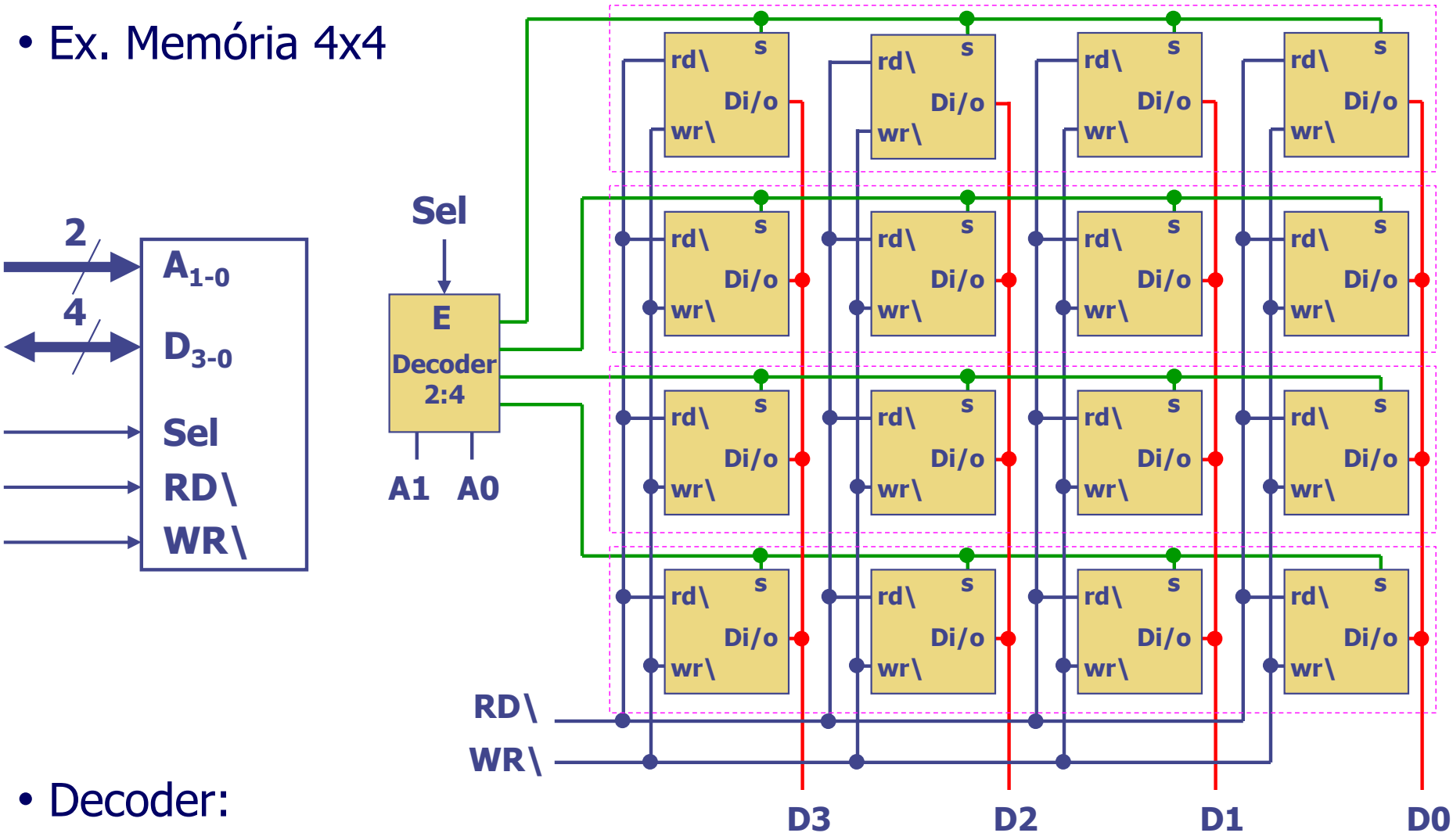
Agrupamento de células de memória

- Através do agrupamento de células-base pode formar-se uma memória de maior dimensão
- O que é necessário especificar:
  - O **número total de Words (N)** que a memória pode armazenar
  - O **Word size** (**K** = 1, 4, 8, 16, 32, ...)
    (Número total de bits = word size * nº words)

- Exemplos:
- 1k x 8
  - 8 bits / word
  - 1k = 2^{10} $\rightarrow$ 10 linhas de endereço $\rightarrow$ 1.024 endereços
- 1M x 4
  - 4 bits / word
  - 1M = 2^{20} $\rightarrow$ 20 linhas de endereço $\rightarrow$ 1.048.576 endereços

Organização 2D

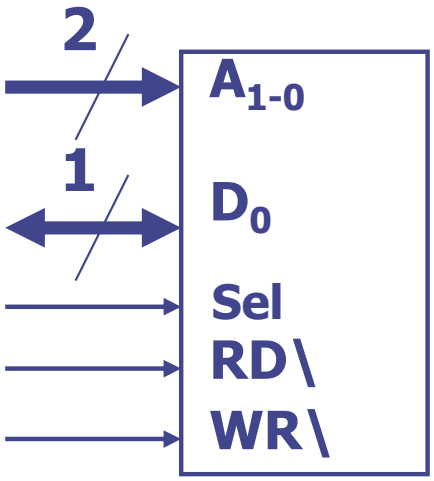
- Ex. Memória 4x4



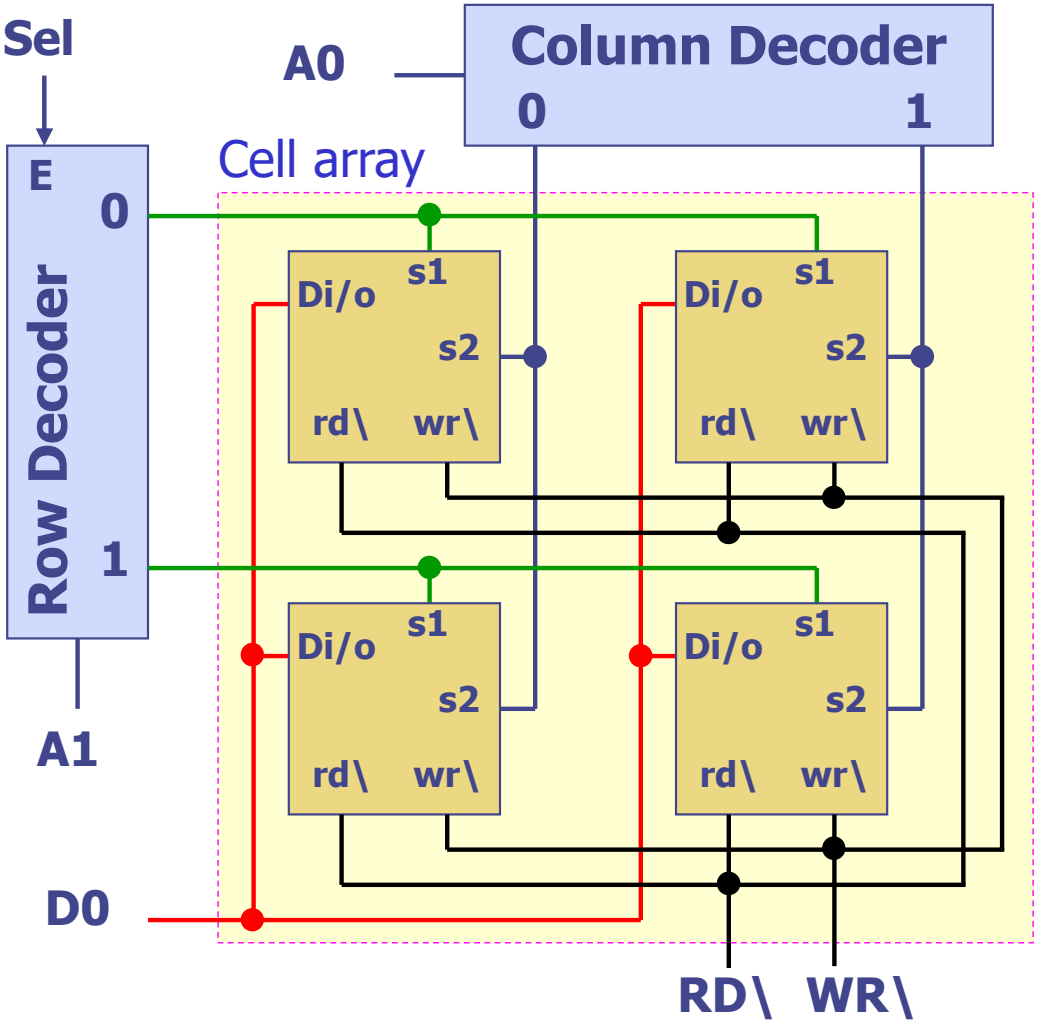
- Decoder:
  - 2^N saídas
  - Ex. 1Mx4 = 2^{20} x 4 -> 1.048.576 saídas, Nº de gates >> 2^{20}

Organização em matriz (conceito)

- Ex. Memória 4x1



(Célula seleccionada: S1.S2=1)



Q1. E se a memória fosse de 16x1? e se fosse 8x1? e 1Mx1?

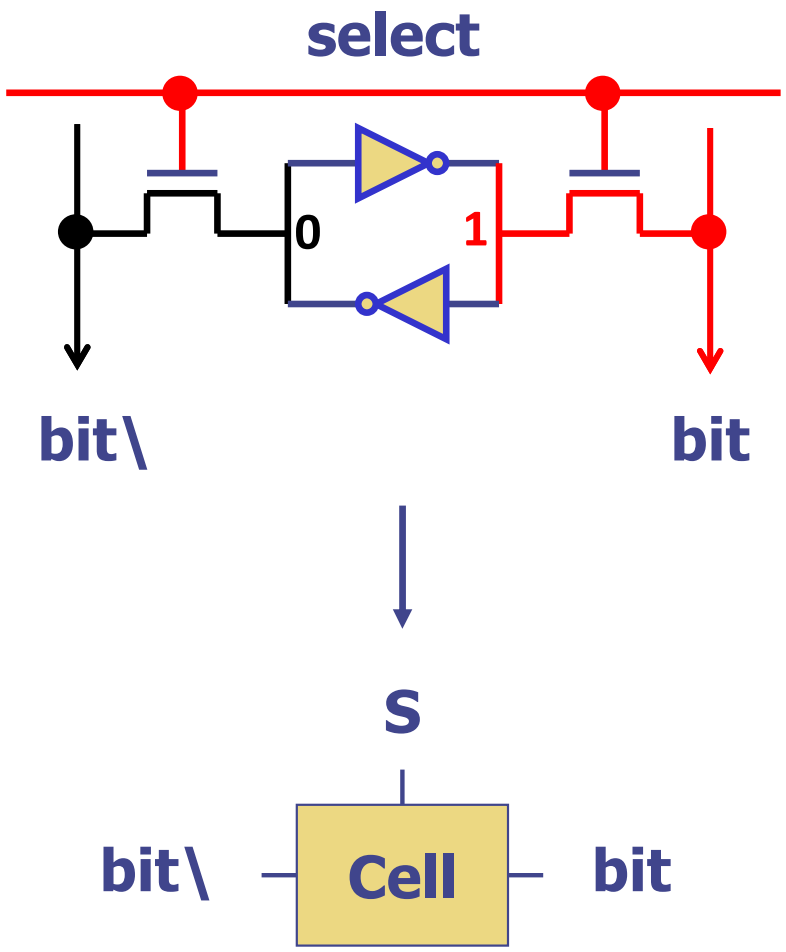
Q2. E se a memória fosse de 4x2? E se fosse 4x4?

Memória do tipo RAM (volátil)

- **SRAM – Static RAM**
  - Vantagens:
    - Rápida
    - Informação permanece até que a alimentação seja cortada
  - Inconvenientes:
    - Implementações típicas: 6 transistores / célula
    - Baixa densidade, elevada dissipação de potência
    - Custo/bit elevado
- **DRAM – Dynamic RAM**
  - Vantagens:
    - Implementações típicas: (1 transistor + 1 condensador) / célula
    - Alta densidade, baixa dissipação de potência
    - Custo/bit baixo
  - Inconvenientes:
    - Informação permanece apenas durante alguns mili-segundos (necessita de *refresh* regular – daí a designação "dynamic")
    - Mais lenta (pelo menos 1 ordem de grandeza) que a SRAM

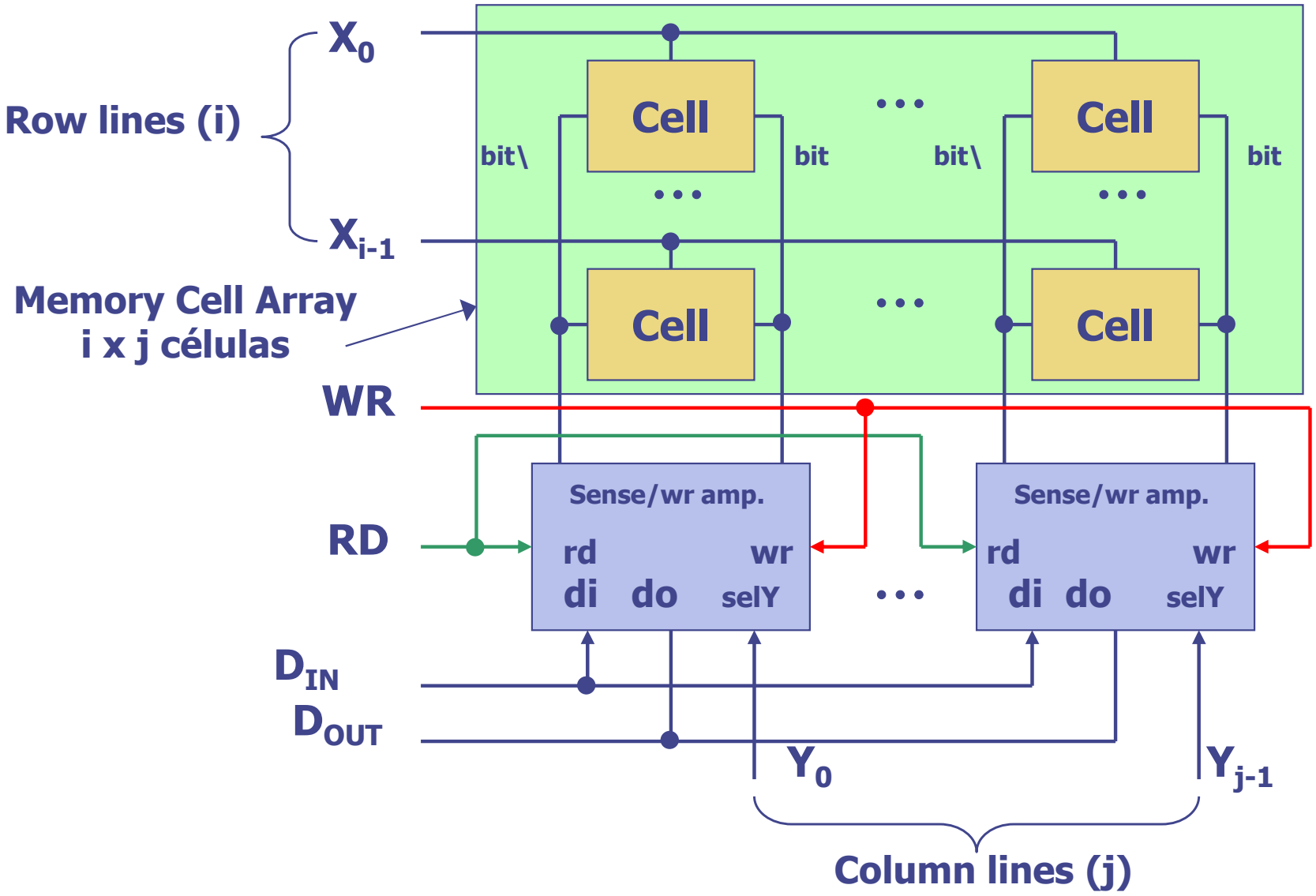
RAM estática (SRAM)

- 6 transístores / célula



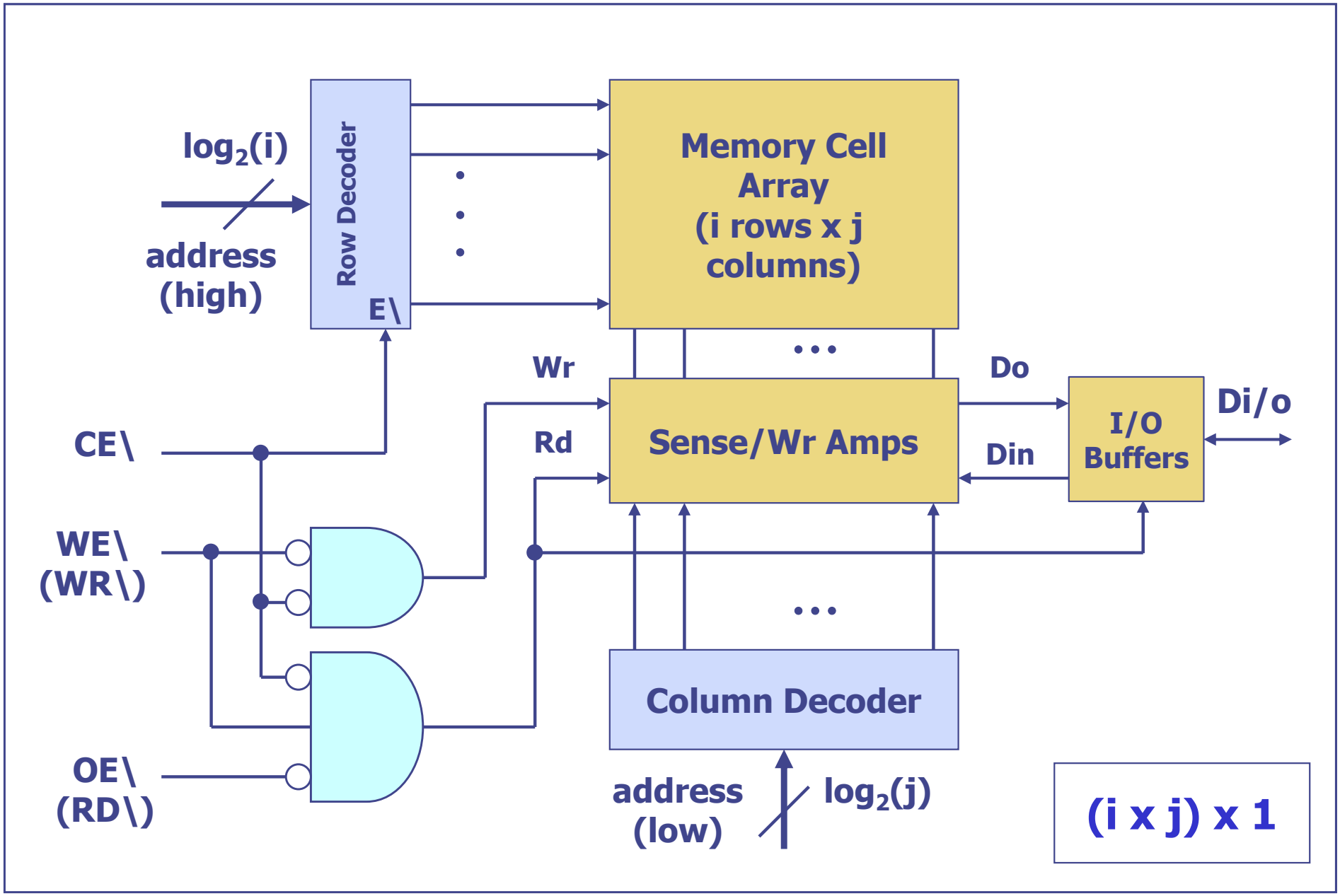
- **Write**
  - Colocar a informação em "bit" (e "bit\"). Exemplo: para a escrita do valor lógico "1" – "bit"=1, "bit\"=0
  - Ativar a linha "select"
- **Read**
  - Ativar a linha "select"
  - O valor lógico armazenado na célula é detetado pela diferença de tensão entre as linhas "bit" e "bit\"

SRAM - Organização interna

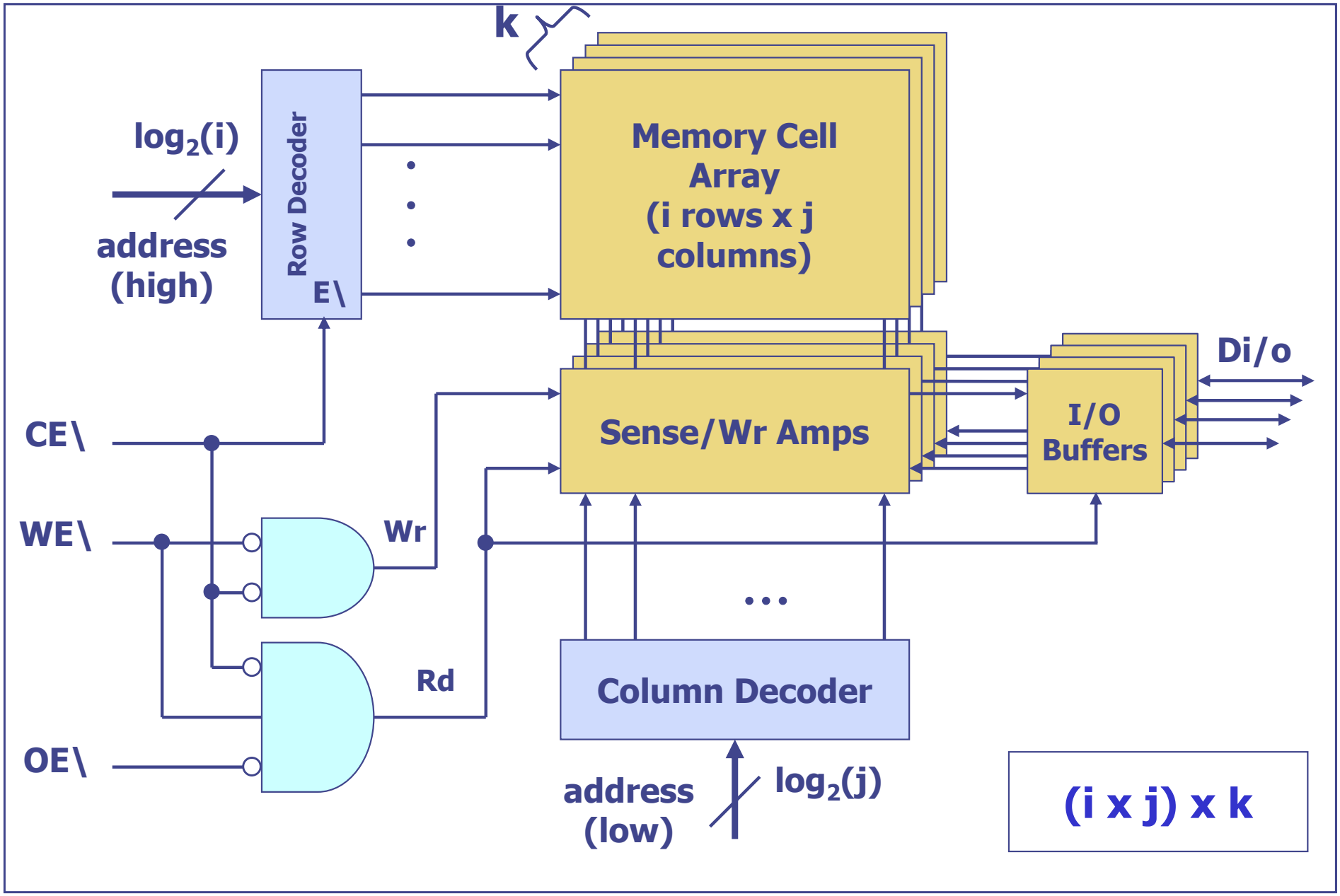


- Se selY=0, as linhas "bit" e "bit\" ficam em alta impedância

SRAM - Organização interna

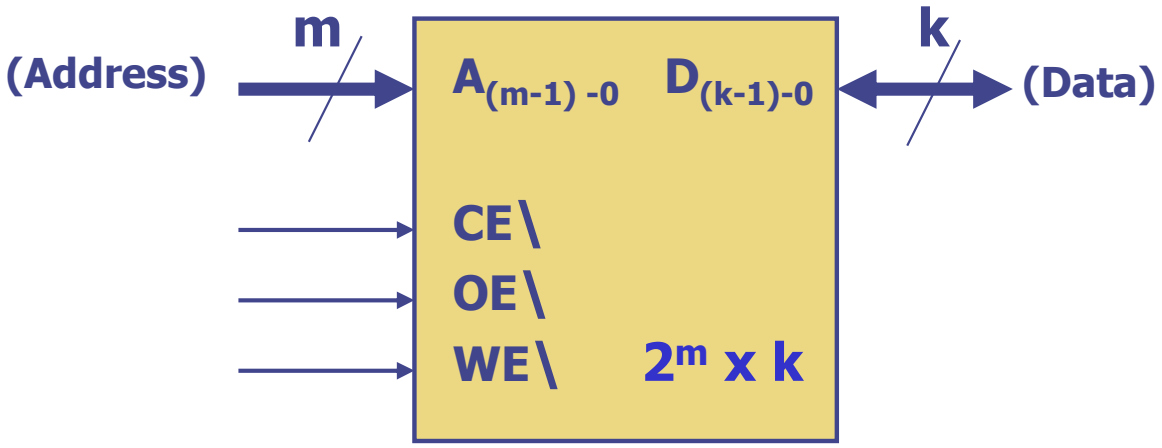


SRAM - Organização interna



SRAM - Bloco funcional

- Diagrama lógico (interface assíncrona)

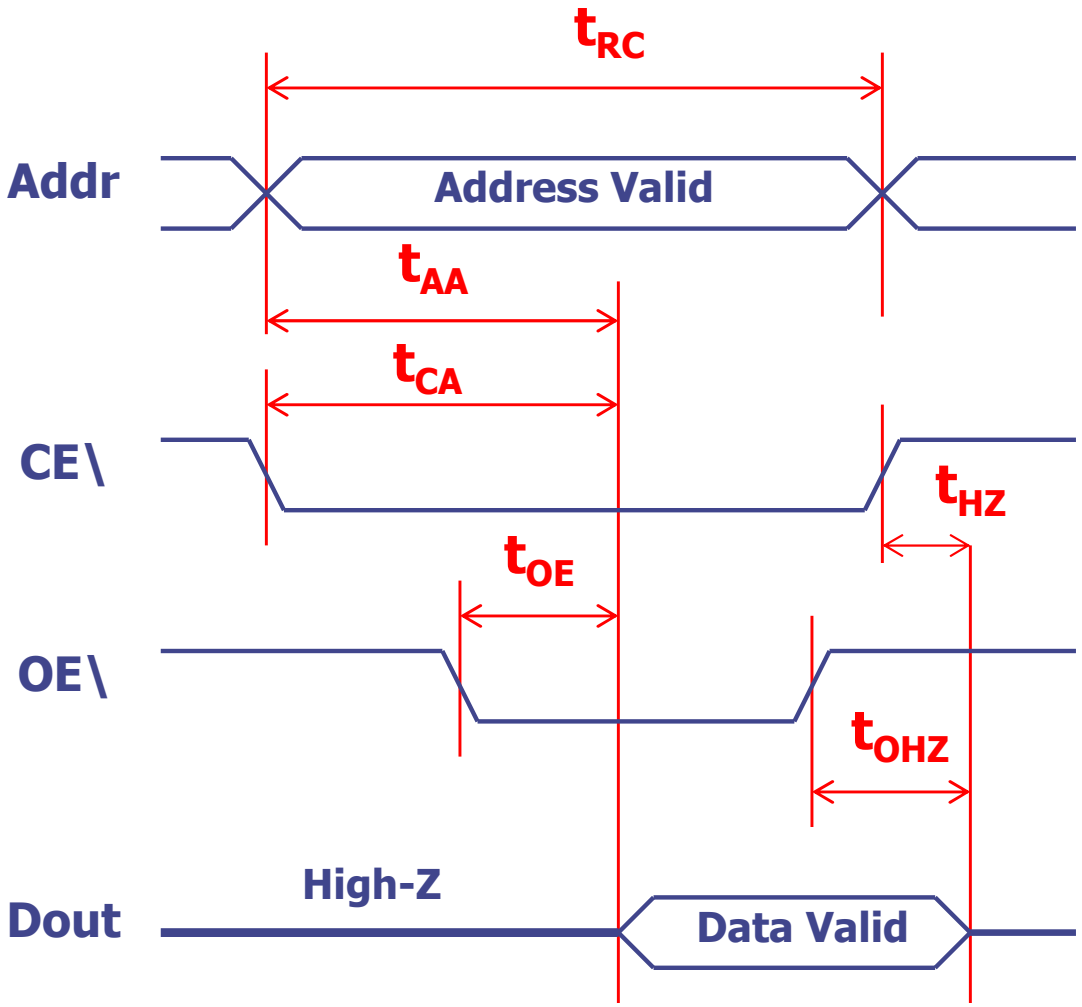


- Tabela de verdade

CE\	OE\	WE\	Operação
1	X	X	High-Z
0	1	1	High-Z
0	X	0	Escrita
0	0	1	Leitura

SRAM – Ciclo de Leitura

- Diagrama temporal típico de um ciclo de leitura de uma memória SRAM (interface assíncrona)



t_{RC} - Read Cycle Time

t_{AA} - Address Access Time

t_{CA} - CE\ Access Time

t_{OE} - Output Enable to Output Valid

t_{HZ} - CE\ to Output in High-Z

t_{OHZ} - OE\ to Output in High-Z

SRAM – Ciclo de leitura

S

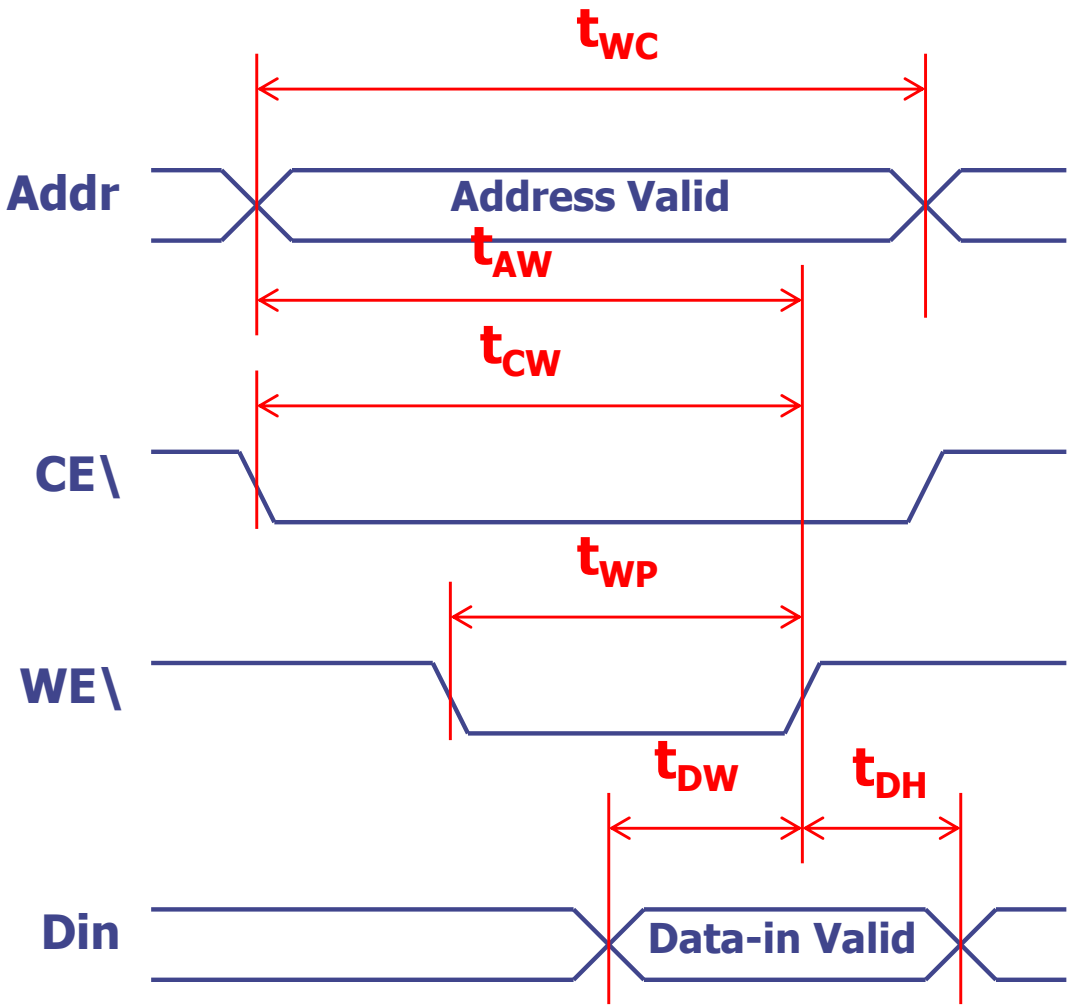
- Valores indicativos (em ns) dos parâmetros associados a um ciclo de leitura de uma memória SRAM:

Parameter	Symbol	Min.	Max.
Read Cycle Time	t_{RC}	1.5	
Address Access Time	t_{AA}		1.5
CE\ Access Time	t_{CA}		1.5
Output Enable to Output Valid	t_{OE}		0.7
CE\ to Output in High-Z	t_{HZ}		0.6
OE\ to Output in High-Z	t_{OHZ}		0.6

- **Cycle Time**: tempo de acesso mais qualquer tempo adicional necessário antes que um segundo acesso possa ter início
- **Access Time**: tempo necessário para os dados ficarem disponíveis no barramento de saída da memória
- **Taxa de transferência**: taxa a que os dados podem ser transferidos de/para uma memória (1 / cycle_time)

SRAM – Ciclo de Escrita

- Diagrama temporal típico de um ciclo de escrita de uma memória SRAM



t_{wC} - Write Cycle Time

t_{AW} - Address Valid to End of Write

t_{CW} - CE\ to End of Write

t_{WP} - Write Pulse Width

t_{DW} - Data Valid to End of Write (data setup time)

t_{DH} - Data Hold Time

SRAM – Ciclo de Escrita

S

- Valores indicativos (em ns) dos parâmetros associados a um ciclo de escrita de uma memória SRAM:

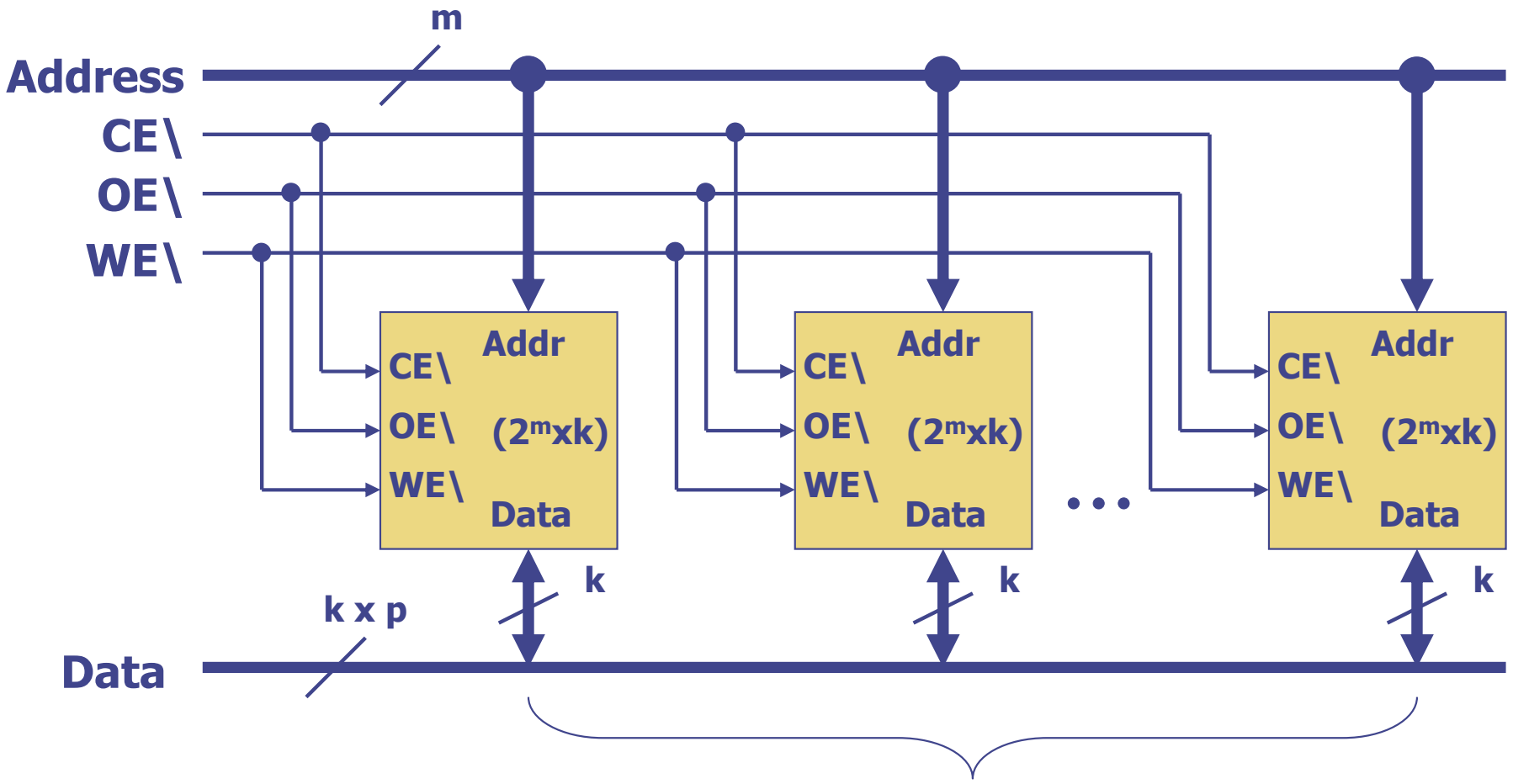
Parameter	Symbol	Min.	Max.
Write Cycle Time	t_{WC}	1.5	
Address Valid to End of Write	t_{AW}	1.0	
CE\ to End of Write	t_{CW}	1.0	
Write Pulse Width	t_{WP}	1.0	
Data Valid to End of Write	t_{DW}	0.7	
Data Hold Time	t_{DH}	0	

Aumento da capacidade de armazenamento

- É frequente ter-se necessidade de memórias com uma capacidade de armazenamento superior à capacidade individual dos circuitos disponíveis comercialmente
- Nessa situação recorre-se à construção de módulos de memória que resultam do agrupamento de circuitos de acordo com o aumento pretendido
- Assim, a construção de um módulo de memória pode envolver as duas fases seguintes, ou apenas uma delas, em função dos circuitos disponíveis e dos requisitos finais de armazenamento:
  - **Aumento da dimensão da palavra**. Exemplo: a partir de C.I.s de 32K x 1, construir uma memória de 32K x 8
  - **Aumento do número total de posições de memória**. Exemplo: a partir de C.I.s de 32K x 8, construir uma memória de 128K x 8

Módulo de memória SRAM

- Aumento da dimensão da palavra

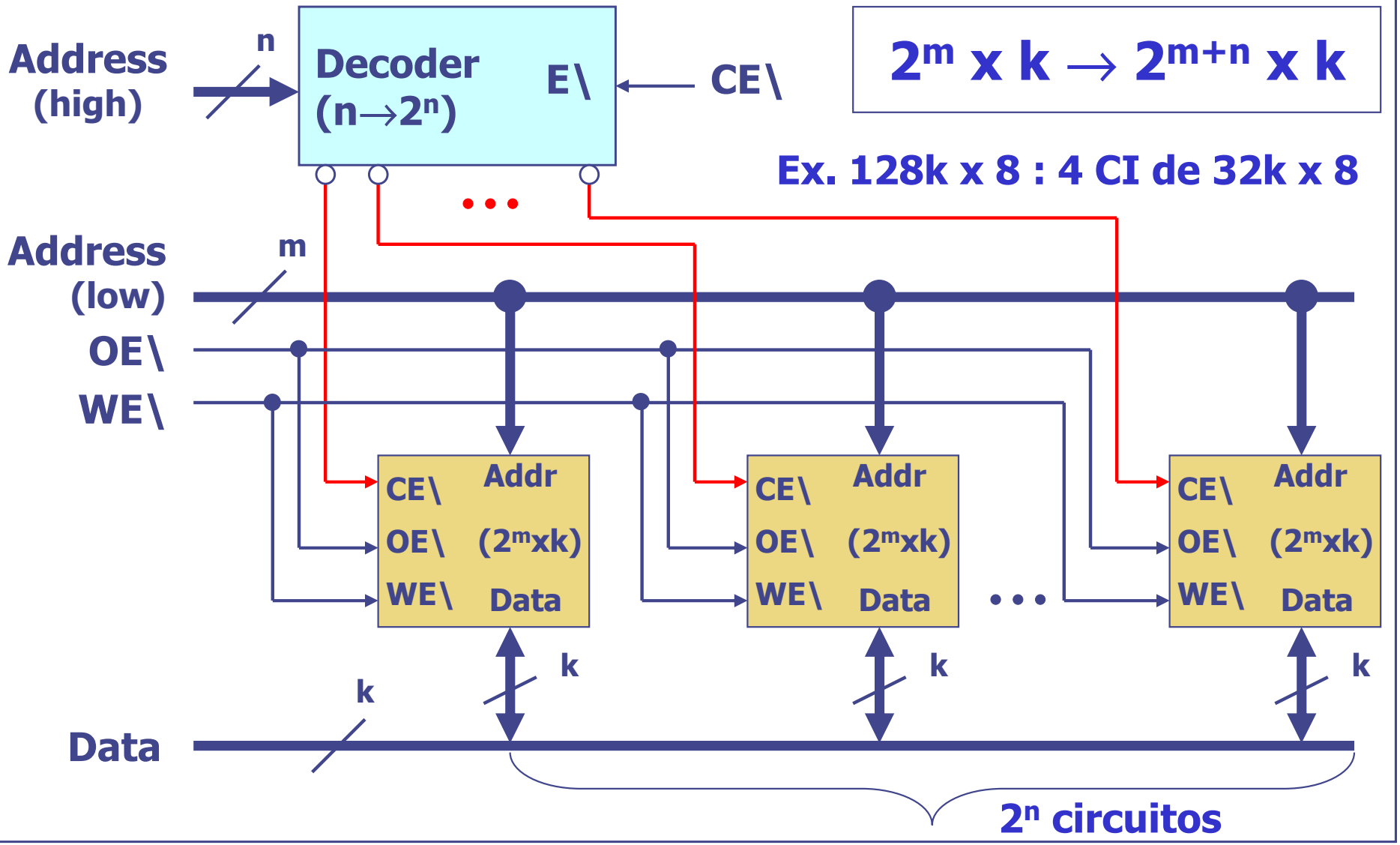


$2^m \times k \rightarrow 2^m \times (k \times p)$

Ex. 32k x 8 : 8 CI de 32k x 1

Módulo de memória SRAM

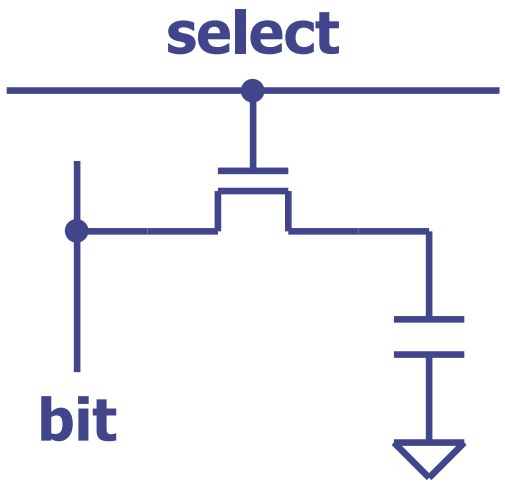
- Aumento do número total de posições de memória



Memória do tipo RAM (volátil)

- **SRAM – Static RAM**
  - Vantagens:
    - Rápida
    - Informação permanece até que a alimentação seja cortada
  - Inconvenientes:
    - Implementações típicas: 6 transistores / célula
    - Baixa densidade, elevada dissipação de potência
    - Custo/bit elevado
- **DRAM – Dynamic RAM**
  - Vantagens:
    - Implementações típicas: (1 transistor + 1 condensador) / célula
    - Alta densidade, baixa dissipação de potência
    - Custo/bit baixo
  - Inconvenientes:
    - Informação permanece apenas durante alguns mili-segundos (necessita de *refresh* regular – daí a designação "dynamic")
    - Mais lenta (pelo menos 1 ordem de grandeza) que a SRAM

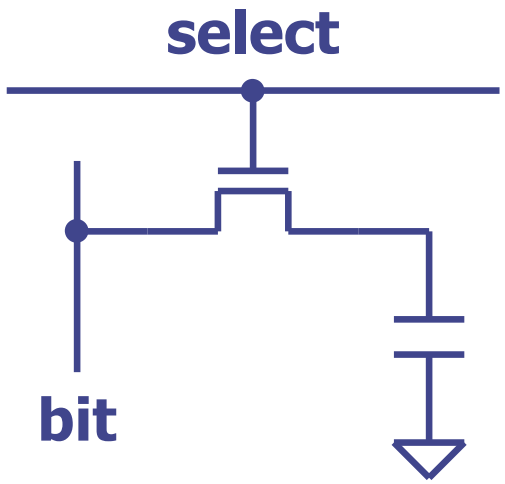
RAM Dinâmica (DRAM)



- Condensador com uma capacidade muito pequena (dezenas de fF (1 fF = 10^{-15} F)

- Na ausência de leitura, o condensador descarrega "lentamente"
- Informação permanece na célula apenas durante alguns mili-segundos
- É obrigatório fazer o refrescamento ("refresh") periódico da carga do condensador
- A operação de leitura é destrutiva (descarrega o condensador)
- Após uma operação de leitura é necessário repor a carga no condensador

RAM Dinâmica (DRAM)



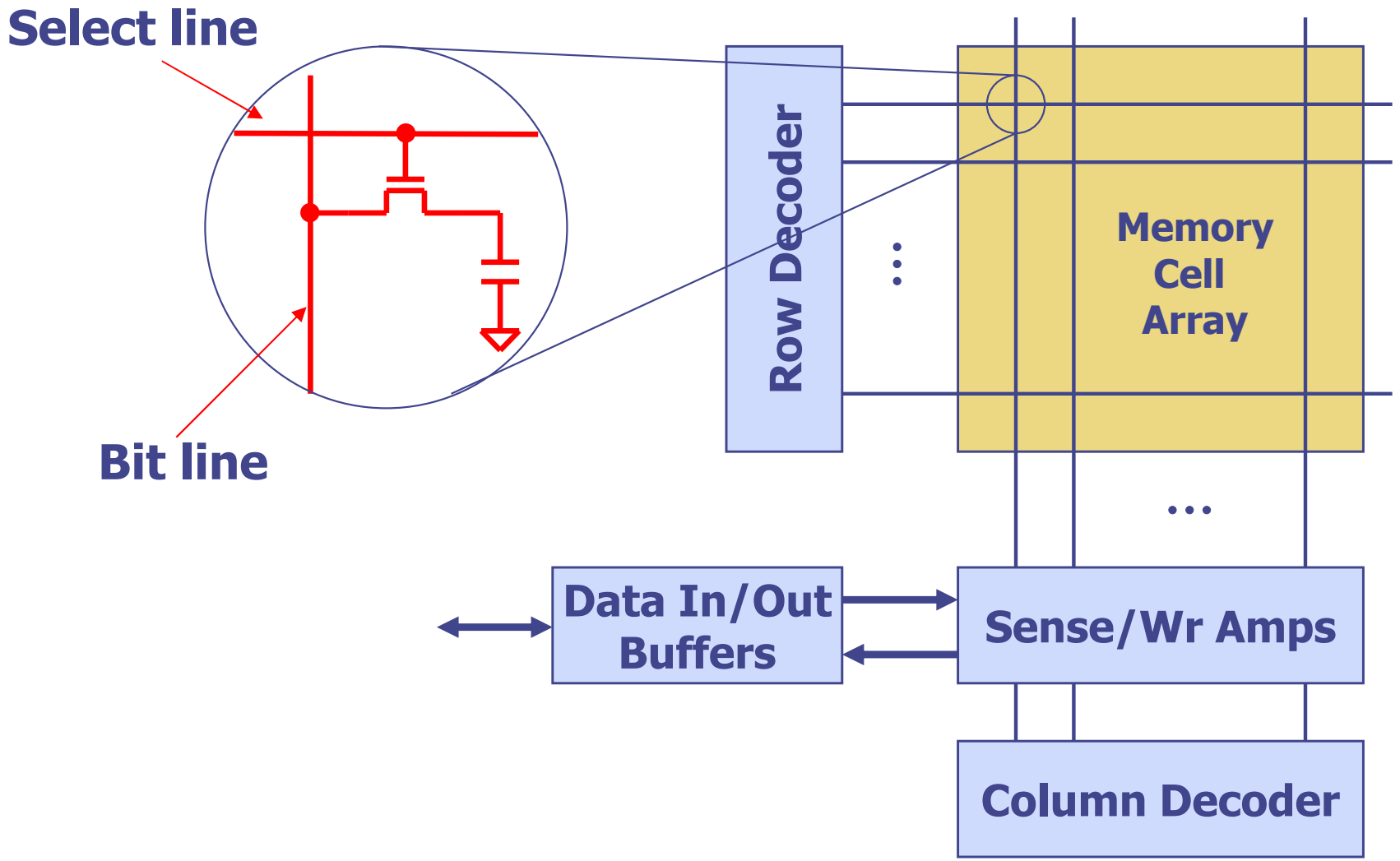
- **Write**
  - Colocar dado na linha "bit"
  - Ativar a linha "select"

- **Read**
  - Pre-carregar a linha "bit" a VDD/2
  - Ativar a linha "select"
  - Valor lógico detetado pela diferença de tensão na linha bit, relativamente a VDD/2
  - Restauro do valor da tensão no condensador (write)

- **Refresh da célula**
  - Operação interna idêntica a uma operação de "Read"

RAM Dinâmica (DRAM)

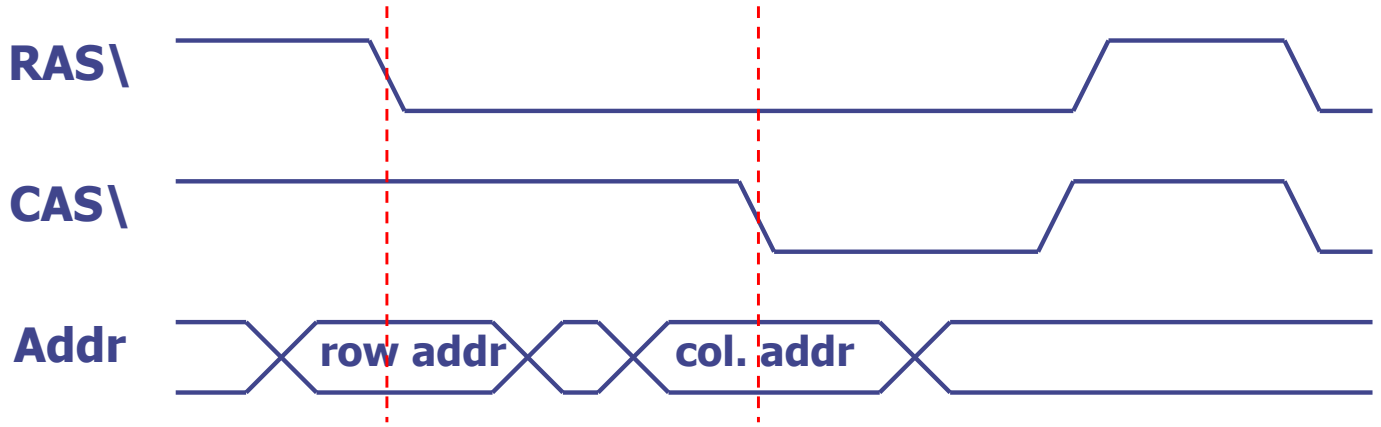
- Organização em matriz



RAM Dinâmica (DRAM)

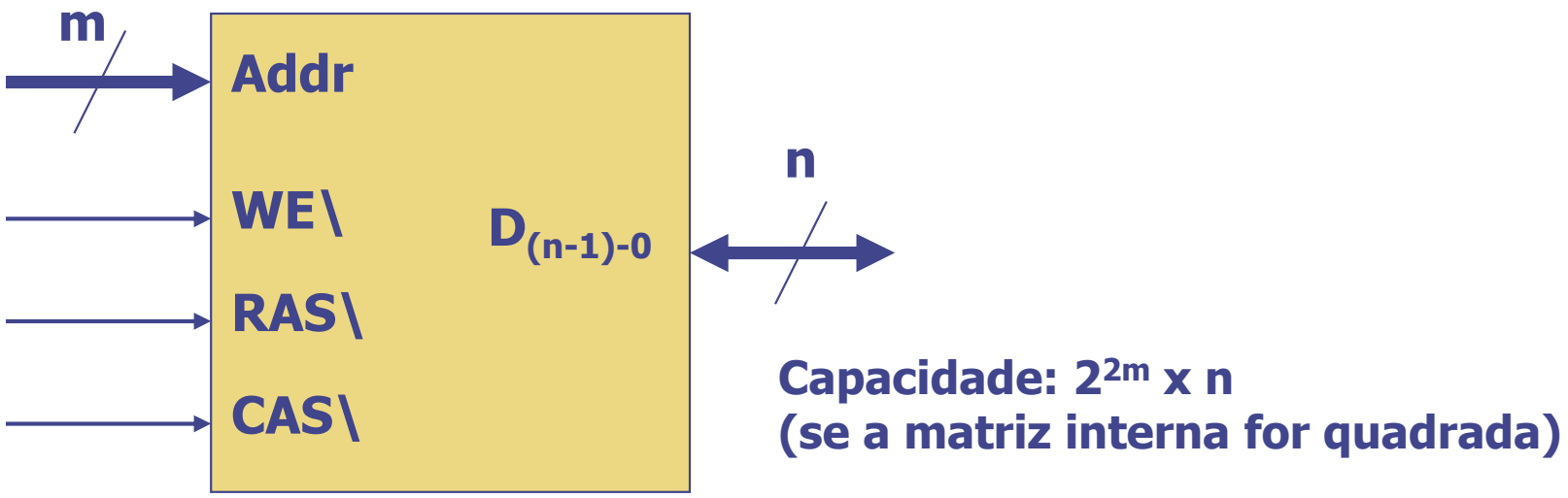
- O endereço de acesso à memória é dividido em 2 partes:

- O barramento de endereços é multiplexado: primeiro é enviado o **endereços de linha** e depois é enviado o **endereço de coluna**
- A multiplexagem no tempo é feita com 2 strobes independentes
  - **RAS** – Row Address Strobe
  - **CAS** – Column Address Strobe



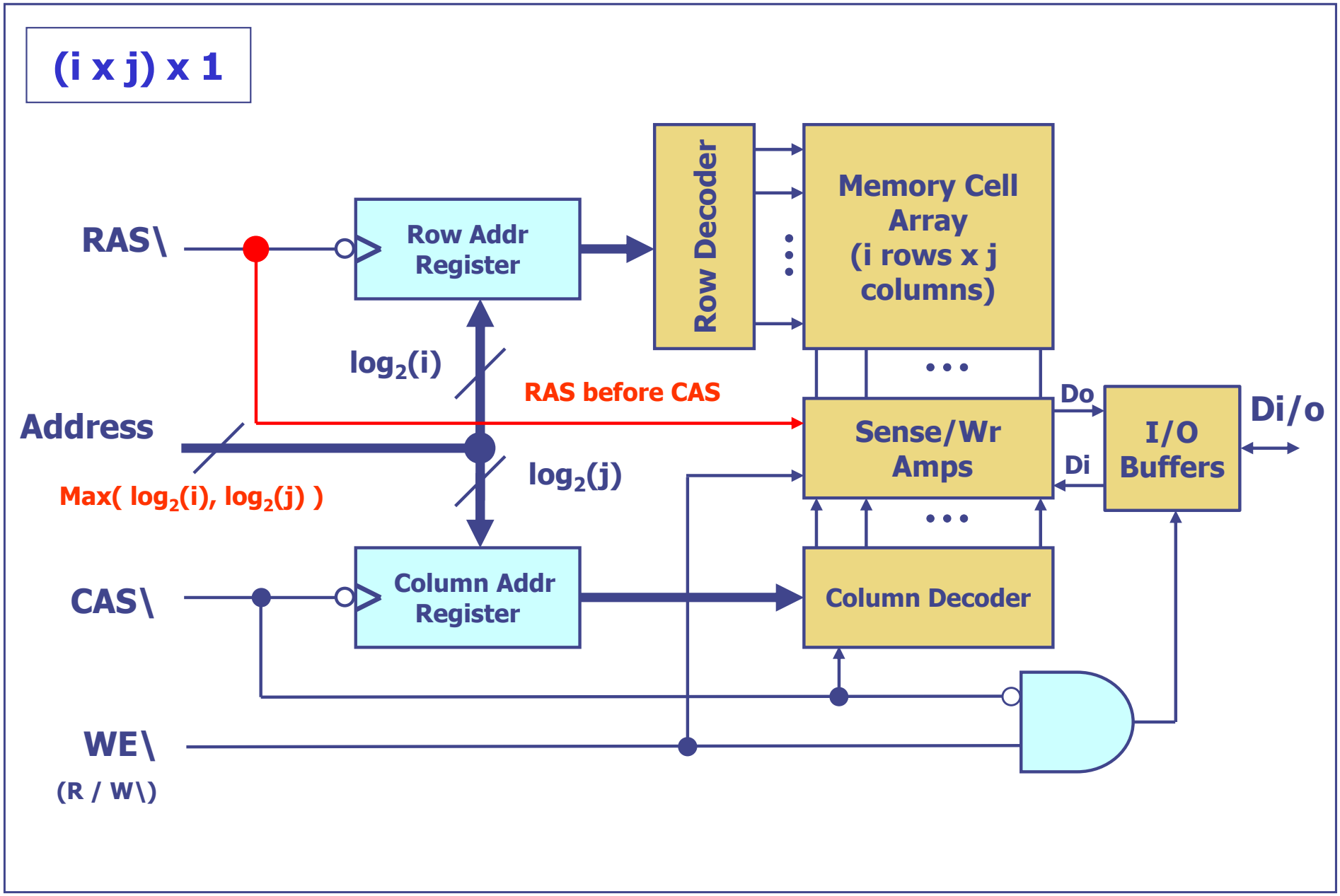
- As transições do RAS e do CAS são usadas para armazenar internamente os endereços de linha e de coluna, respetivamente
- Linha CAS funciona também como "chip-enable"

DRAM - Diagrama lógico



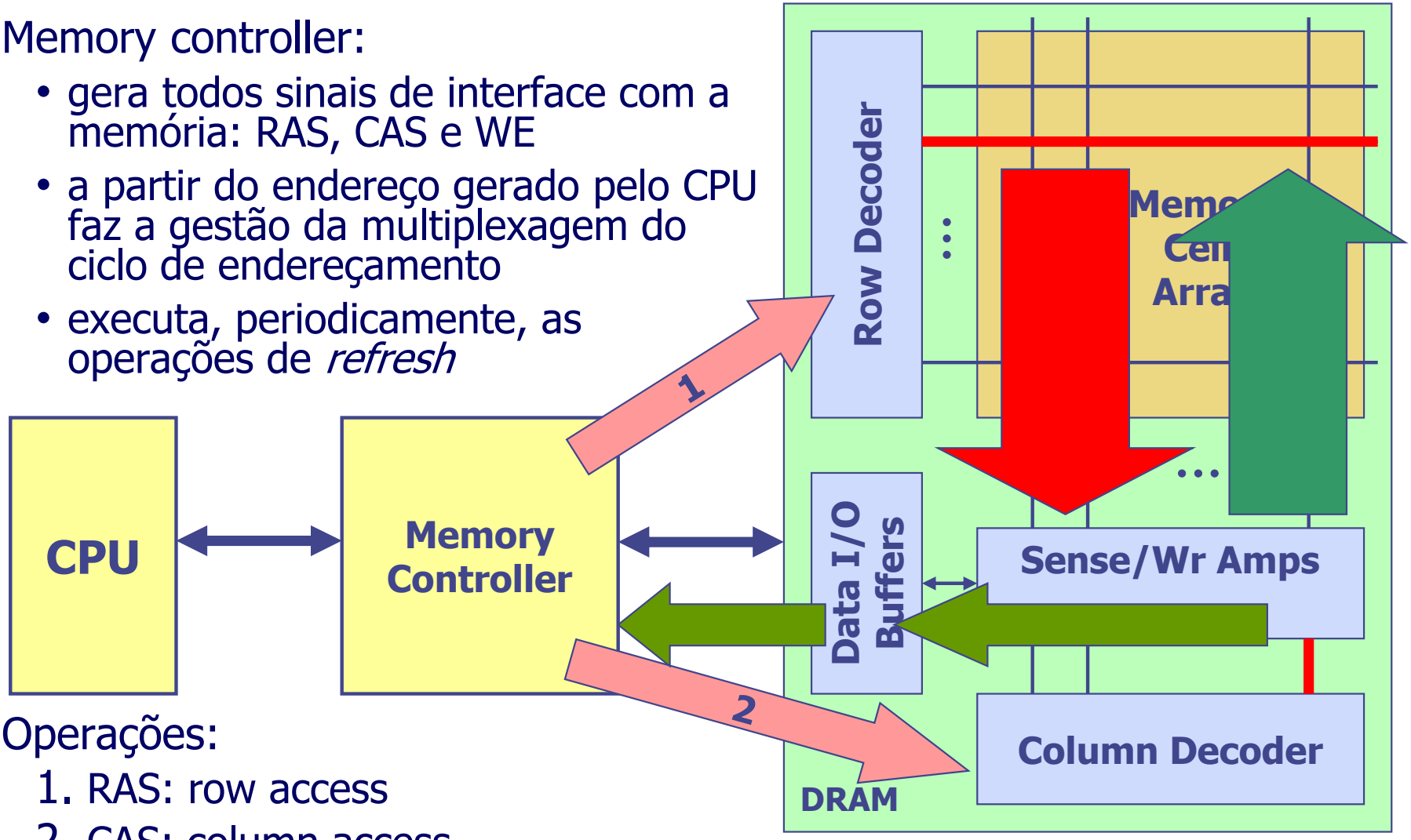
- WE\= 0 → escrita; WE\=1 → leitura (≡ R/W\)
- RAS\: valida endereço da linha na transição descendente
- CAS\: valida endereço da coluna na transição descendente

DRAM – Diagrama de blocos conceptual



DRAM – Leitura

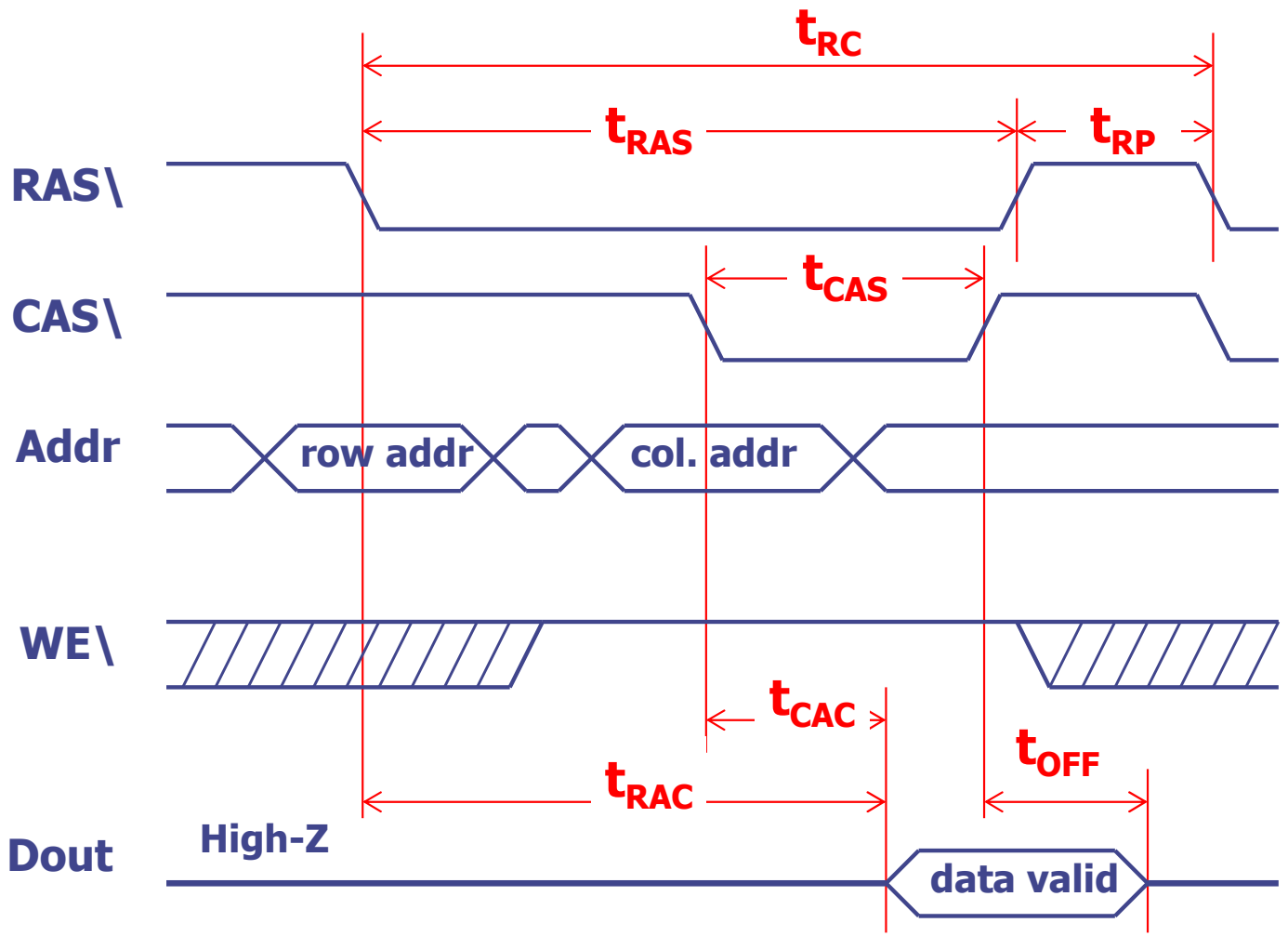
- Memory controller:
  - gera todos sinais de interface com a memória: RAS, CAS e WE
  - a partir do endereço gerado pelo CPU faz a gestão da multiplexagem do ciclo de endereçamento
  - executa, periodicamente, as operações de *refresh*



- Operações:
  1. RAS: row access
  2. CAS: column access
- Buffer de linha (*row buffer*) armazena temporariamente todos os bits de uma linha de células da matriz

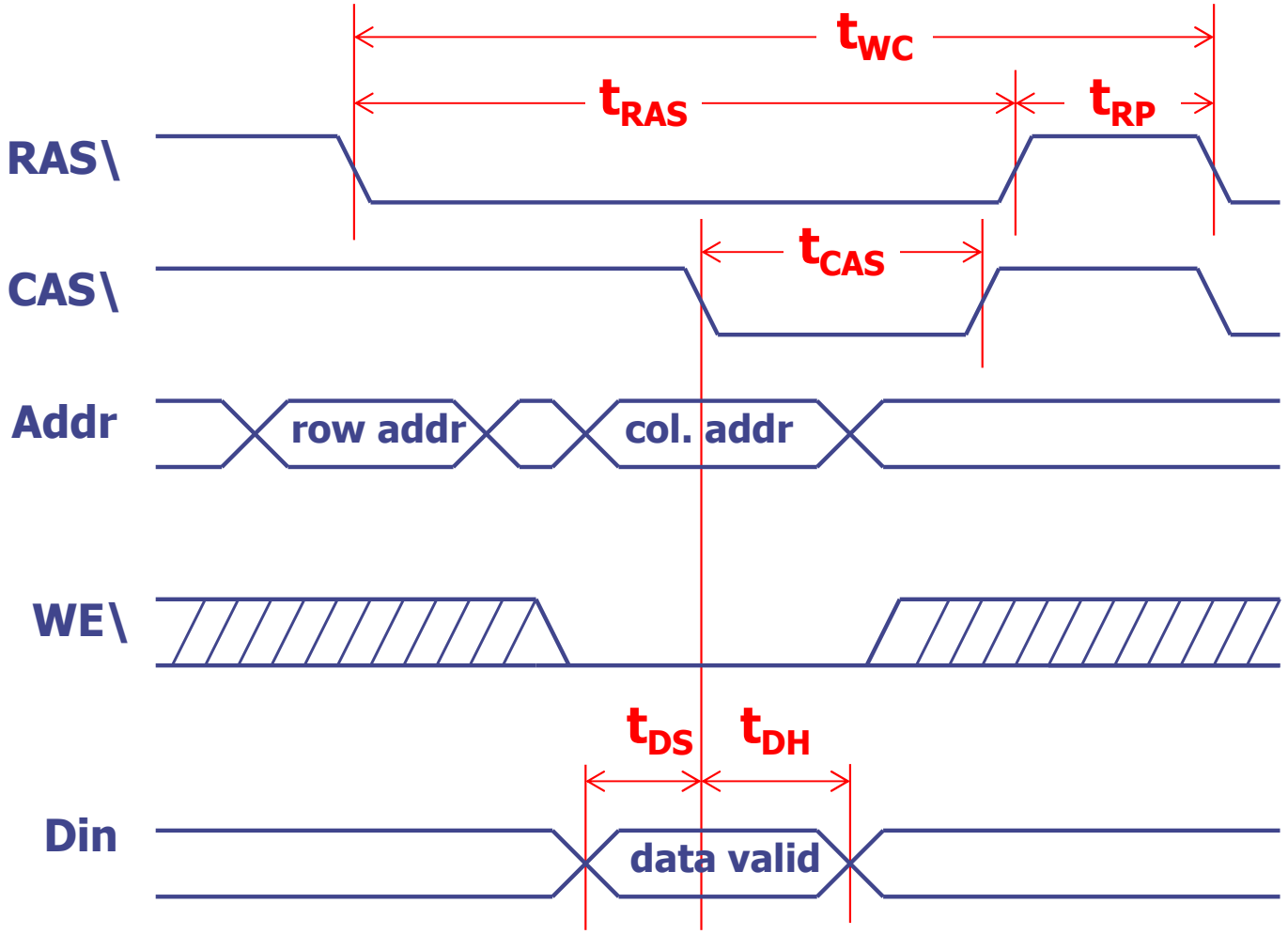
DRAM – Ciclo de Leitura

- Diagrama temporal típico de um ciclo de leitura de uma memória DRAM



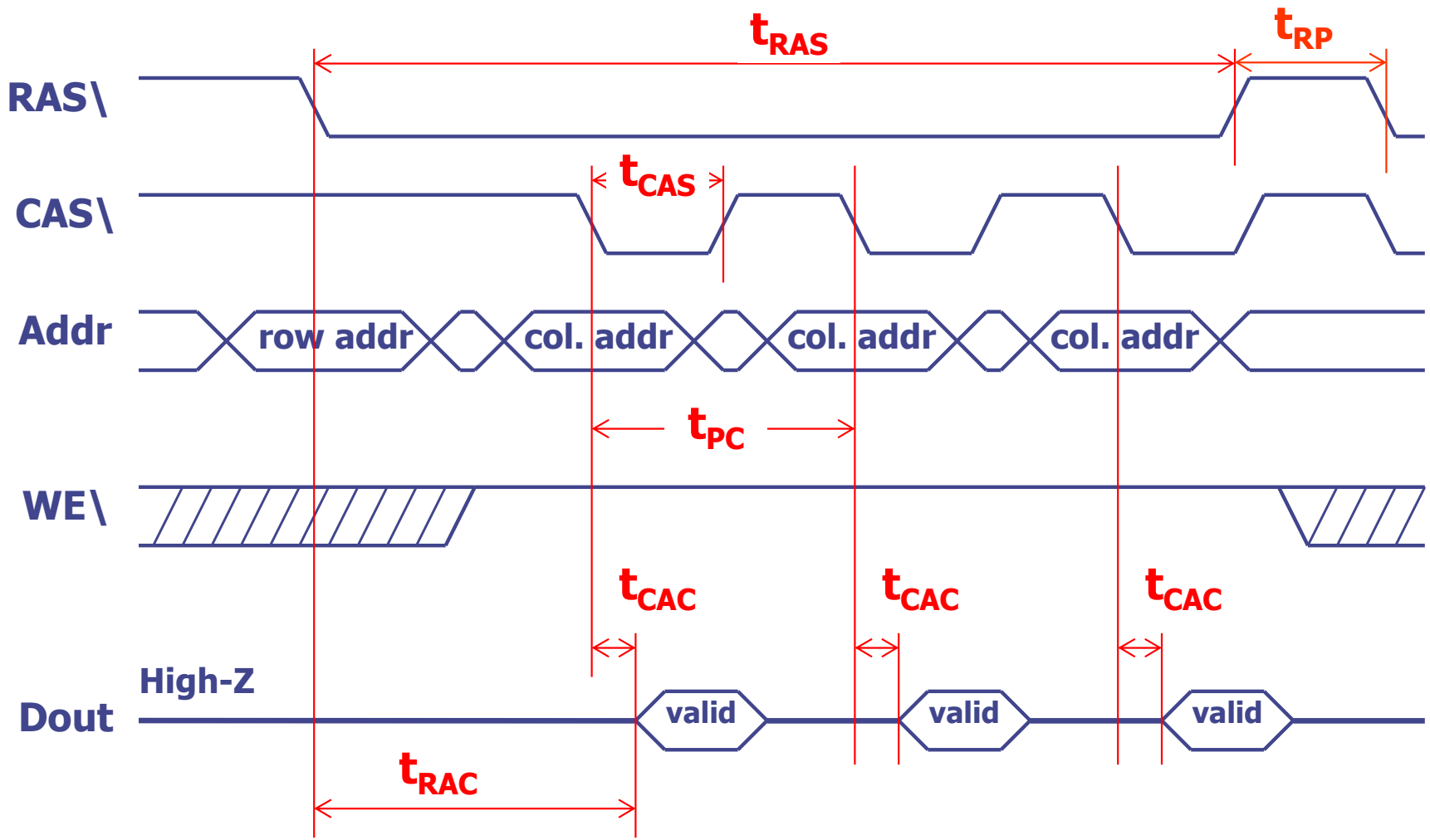
DRAM – Ciclo de Escrita

- Diagrama temporal típico de um ciclo de escrita (*early write*) de uma memória DRAM



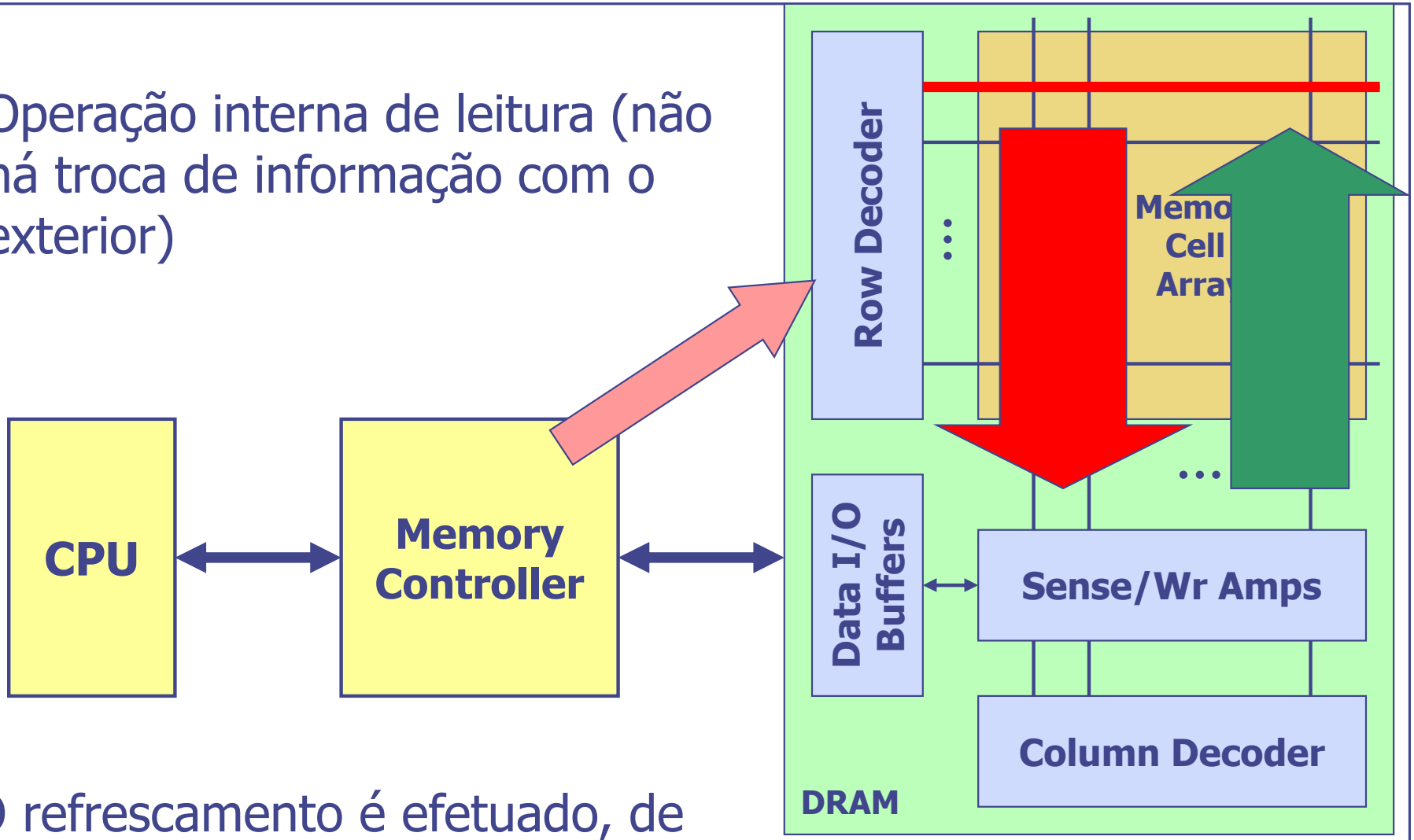
DRAM – Ciclo de Leitura em *page mode*

- Diagrama temporal típico de um ciclo de leitura de uma memória DRAM, em modo paginado (*page mode*)



DRAM – Refrescamento

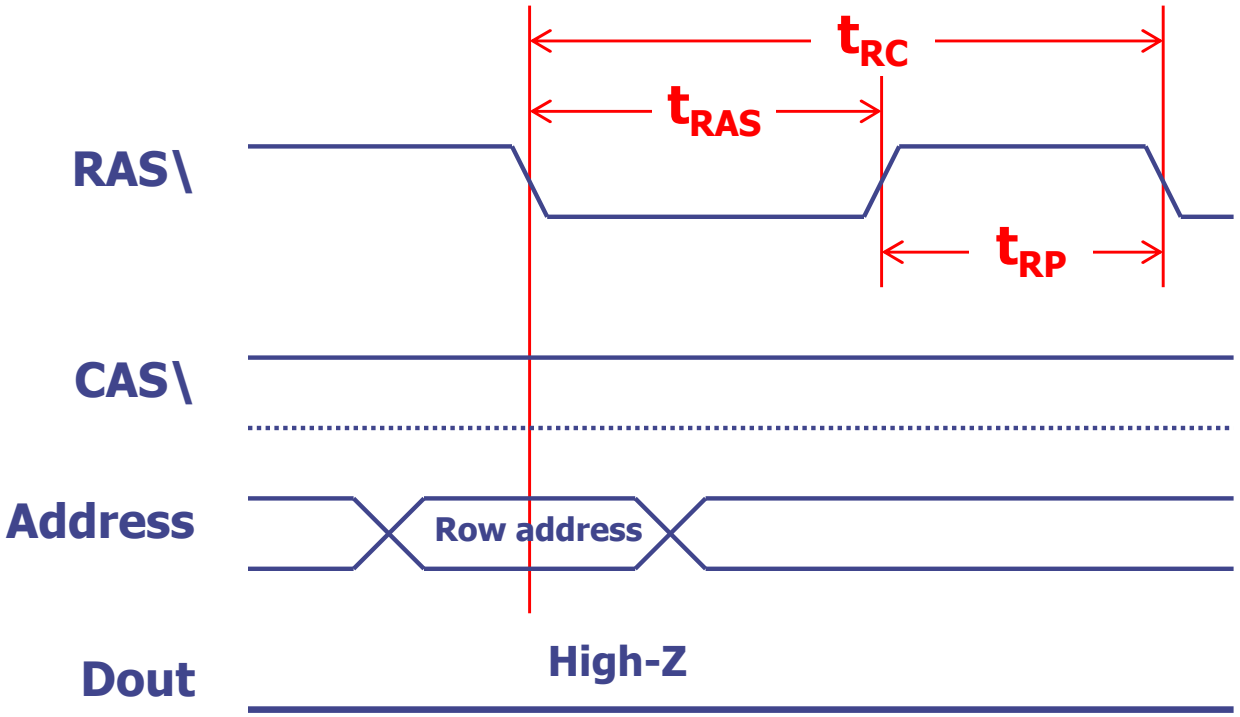
Operação interna de leitura (não há troca de informação com o exterior)



O refrescamento é efetuado, de forma sequencial, para cada uma das linhas da matriz

DRAM Refresh – RAS Only

- O *refresh* é feito simultaneamente em **todas as células da mesma linha da matriz** (especificada no address bus, no momento da ativação do sinal RAS\)
- O sinal CAS\ mantém-se inativo durante o processo



DRAM - Parâmetros principais

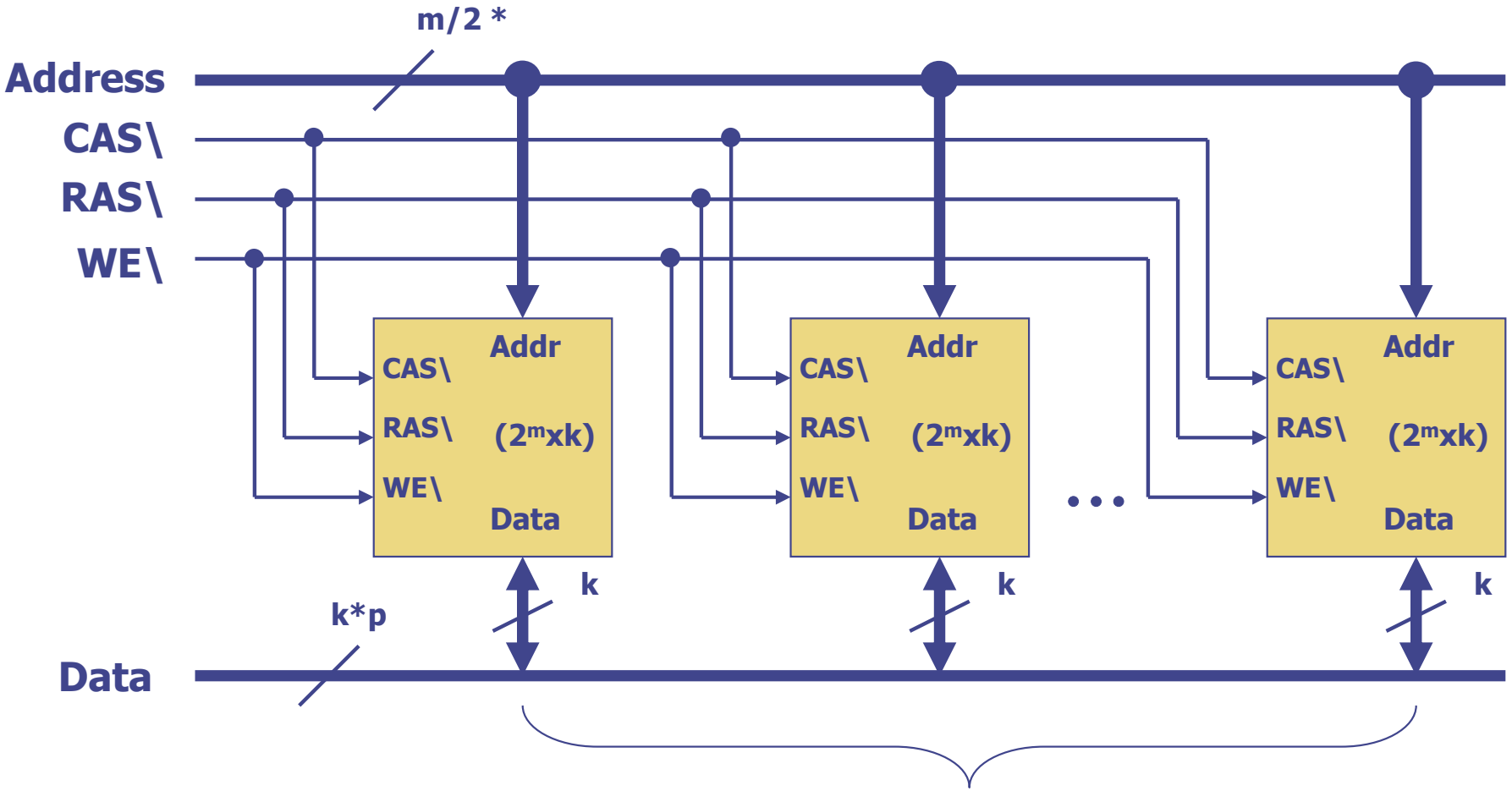
S

- Valores indicativos (em ns) dos tempos indicados nos diagramas temporais de leitura e escrita de uma memória DRAM com um tempo de acesso de 55 ns:

Parameter	Symbol	Min.	Max.
Read or Write Cycle Time	t_{RC}	100	
RAS\ precharge time	t_{RP}	45	
Page mode cycle time	t_{PC}	35	
RAS\ pulse width	t_{RAS}	55	10000
CAS\ pulse width	t_{CAS}	28	10000
Data-in setup time	t_{DS}	5	
Data-in hold time	t_{DH}	14	
Output buffer turn-off delay	t_{OFF}		15
Access time from RAS\	t_{RAC}		55
Access time from CAS\	t_{CAC}		28

Módulo de memória DRAM

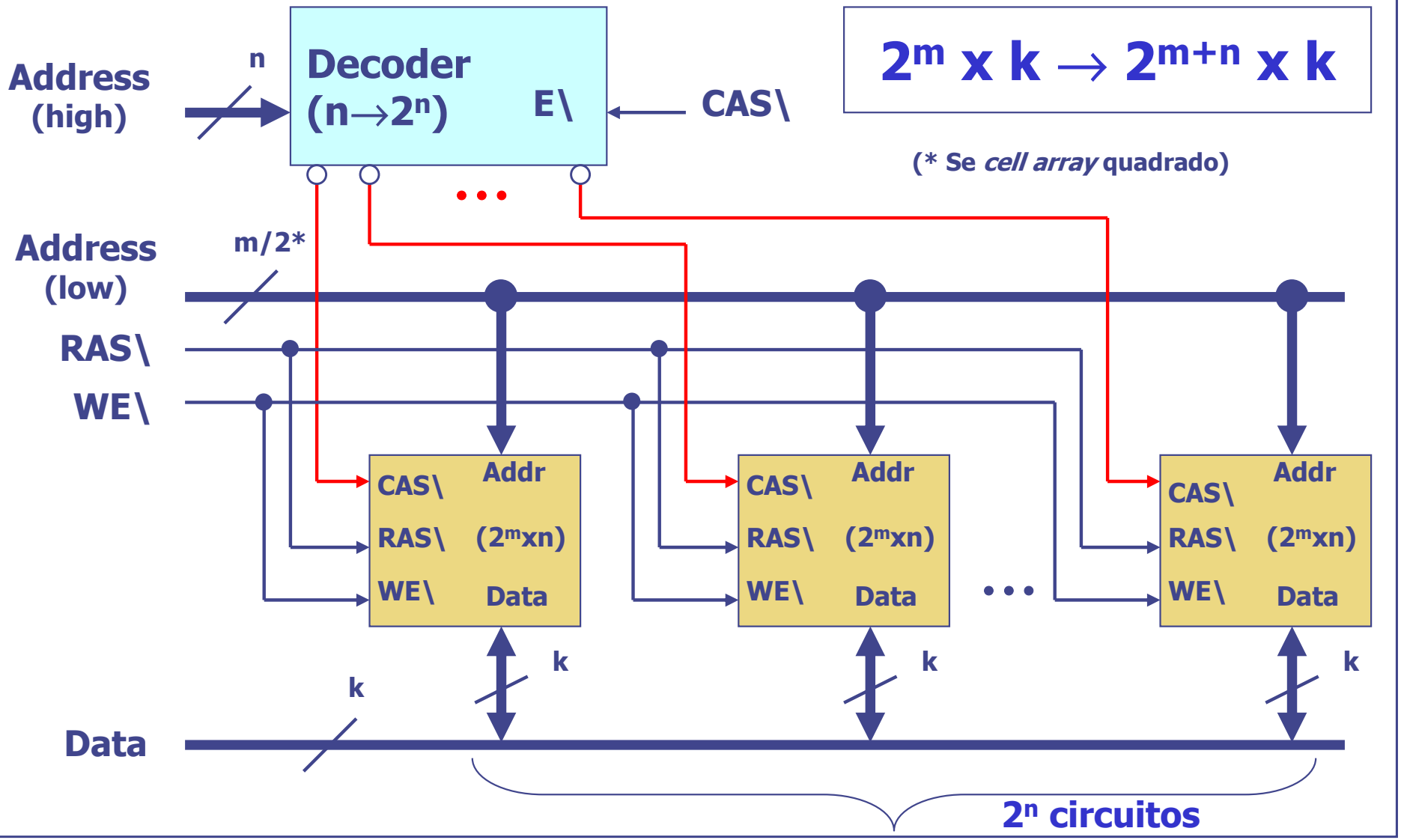
- Aumento da dimensão da palavra



$2^m \times k \rightarrow 2^m \times (k*p)$

Módulo de memória DRAM

- Aumento do número total de posições de memória



Melhorias de desempenho da DRAM

- **Fast Page Mode**
  - Adiciona sinais de temporização que permitem acessos repetidos ao buffer de linha (sem outro tempo de acesso à linha)
- **Synchronous DRAM (SDRAM)**
  - Adiciona um sinal de relógio à interface DRAM, para facilitar a sincronização de transferências múltiplas
  - Múltiplos bancos, cada um com o seu buffer de linha
- **Double Data Rate (DDR SDRAM)**
  - Transferência de dados tanto no flanco ascendente como no flanco descendente do sinal de relógio (duplica a taxa de transferência de pico)
  - Versão atual: DDR5 (2021->). Exemplo: DDR5-6400, 6400 Milhões de transferências por segundo, relógio de 3.2 GHz

- Estas técnicas melhoram a largura de banda, mas não a latência

Copied from Wikipedia

Name		Release year	Chip			Bus			Voltage (V)
Gen	Standard		Clock rate (MHz)	Cycle time (ns)	Pre-fetch	Clock rate (MHz)	Transfer rate (MT/s)	Bandwidth (MB/s)	
DDR	DDR-200	1998	100	10	2n	100	200	1600	2.5
	DDR-266		133	7.5		133	266	$2133\frac{1}{3}$	
	DDR-400		200	5		200	400	3200	2.6
DDR2	DDR2-400	2003	100	10	4n	200	400	3200	1.8
	DDR2-533		$133\frac{1}{3}$	7.5		$266\frac{2}{3}$	$533\frac{1}{3}$	$4266\frac{2}{3}$	
	DDR2-800		200	5		400	800	6400	
	DDR2-1066		$266\frac{2}{3}$	3.75		$533\frac{1}{3}$	$1066\frac{2}{3}$	$8533\frac{1}{3}$	
DDR3	DDR3-800	2007	100	10	8n	400	800	6400	1.5/1.35
	DDR3-1600		200	5		800	1600	12800	
	DDR3-1866		$233\frac{1}{3}$	4.29		$933\frac{1}{3}$	$1866\frac{2}{3}$	$14933\frac{1}{3}$	
DDR4	DDR4-1600	2014	200	5	8n	800	1600	12800	1.2/1.05
	DDR4-2400		300	$3\frac{1}{3}$		1200	2400	19200	
	DDR4-2666		$333\frac{1}{3}$	3		$1333\frac{1}{3}$	$2666\frac{2}{3}$	$21333\frac{1}{3}$	
	DDR4-3200		400	2.5		1600	3200	25600	
DDR5	DDR5-3200	2020	200	5	16n	1600	3200	25600	1.1
	DDR5-4000		250	4		2000	4000	32000	
	DDR5-5600		350	2.86		2800	5600	44800	
	DDR5-6400		400	2.5		3200	6400	51200	
	DDR5-7200		450	2.22		3600	7200	57600	

Aulas 20 e 21

- Organização da memória de um sistema computacional
- Hierarquia do sistema de memória
- Localidade temporal e espacial
- A memória cache
  - Princípio de funcionamento
  - Cache com mapeamento associativo
  - Cache com mapeamento direto
  - Cache com mapeamento parcialmente associativo

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Introdução

- Pretensão do utilizador:
  - Uma memória rápida e com grande capacidade de armazenamento
  - Que custe o preço de uma memória lenta... ☺
- Solução perfeita para este dilema não existe

Tecnologia	Tempo Acesso	$ / GB
SRAM	0,5 – 2,5 ns	$500 - $1000
DRAM	35 - 70 ns	$10 - $20

(Dados de 2012)

- A organização da memória de um sistema computacional resulta de um compromisso entre:
  - Velocidade, Capacidade, Custo, Consumo energético
- Menor tempo de acesso: maior custo por bit
- Maior capacidade: maior tempo de acesso

Introdução

- Memória DRAM (Dynamic RAM)

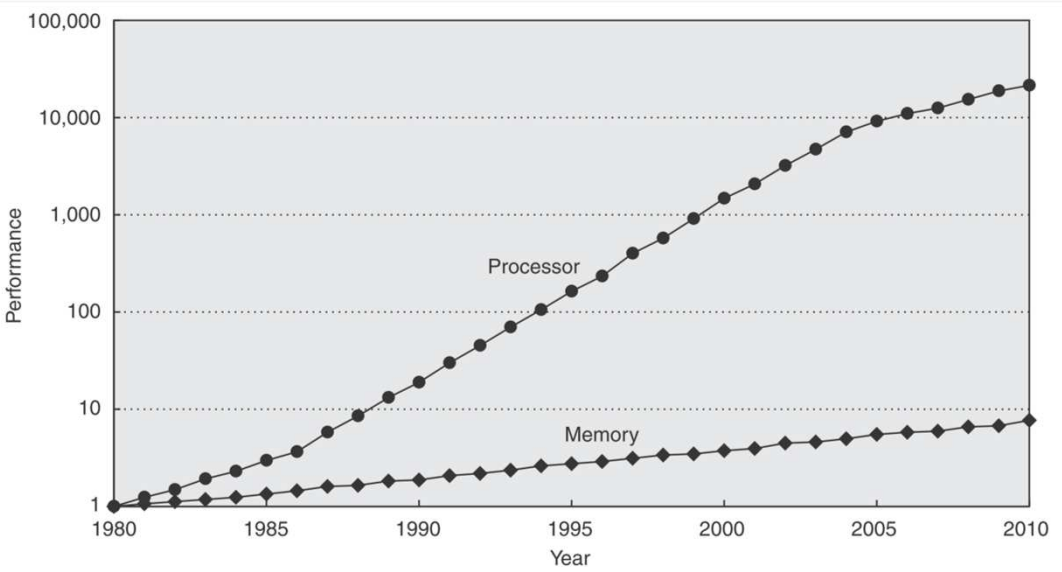
Ano	Capacidade (max. por chip)	Access Time	$ / Mb
1980	64 kbit	250 ns	$1500
1983	256 kbit	185 ns	$500
1985	1 Mbit	135 ns	$200
1989	4 Mbit	110 ns	$50
1992	16 Mbit	90 ns	$15
1996	64 Mbit	60 ns	$10
1998	128 Mbit	60 ns	$4
2000	256 Mbit	55 ns	$1
2004	512 Mbit	50 ns	$0.25
2007	1024 Mbit	45 ns	$0.05
2010	2 Gbit	40 ns	$0.03
2012	4 Gbit	35 ns	$0.001

x 60000

÷ 7

Introdução

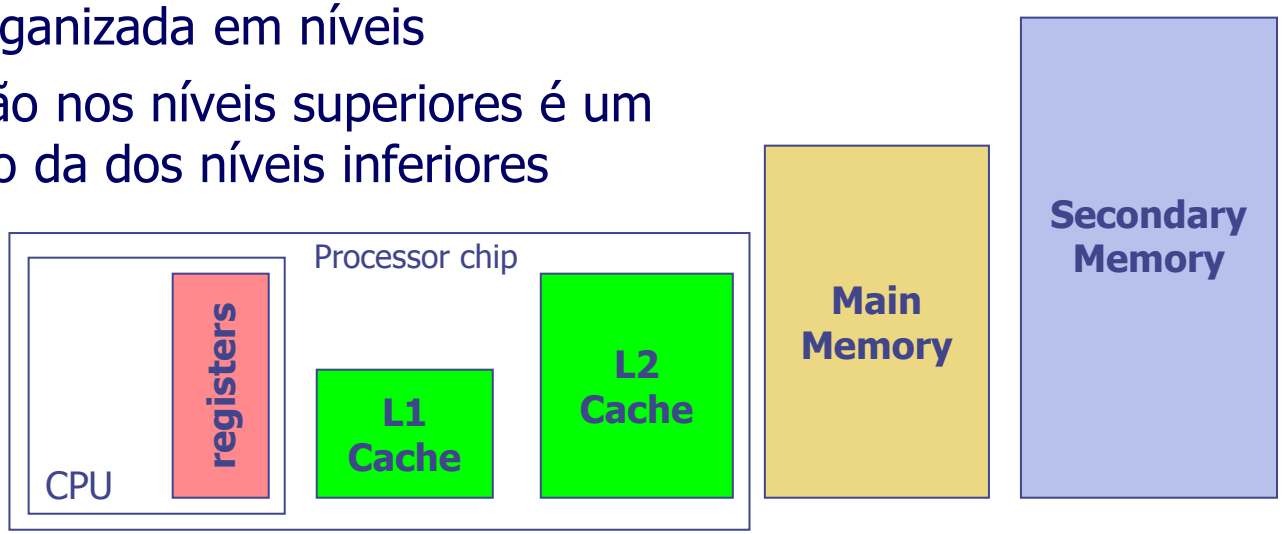
- O processador deve ser alimentado de instruções e dados a uma taxa que não comprometa o desempenho do sistema
- A diferença entre o desempenho do processador e da memória (DRAM) tem vindo a aumentar



- Solução:
  - Guardar a informação mais vezes utilizada pelo CPU numa memória rápida (*static* RAM) de pequena capacidade
  - Aceder raramente à memória principal (mais lenta) para obter a informação em falta (apenas quando necessário)
  - Transferir blocos de informação da memória principal para a memória rápida
- Conceito: **cache**

Hierarquia de memória

- Memória organizada em níveis
- A informação nos níveis superiores é um subconjunto da dos níveis inferiores



Speed:	**0.1's nS**	**1's nS**	**1's nS**	**10's nS**	**10's mS [10's μs]**
Size (bytes):	**100's**	**10's K**	**100's K**	**1's G**	**1's T [0.1s T]**
$$/byte:	**High**	→			**Low**

- A informação circula apenas entre níveis adjacentes da hierarquia
- **Bloco** – quantidade de informação que circula entre níveis adjacentes (n bytes)

Hierarquia de memória

- Solução 1 – expor a hierarquia
  - Alternativas de armazenamento: registos internos do CPU, memória rápida, memória principal, disco
  - Cabe ao programador utilizar racionalmente estas alternativas de armazenamento
  - Exemplo de processador que usa esta técnica: Cell microprocessor (Cell Broadband Engine Architecture; usado por ex. na PlayStation 3 e algumas televisões)
- Solução 2 – esconder a hierarquia
  - Modelo de programação:
    - Tipo de memória único
    - Espaço de endereçamento único
  - A máquina gere automaticamente o acesso ao sistema de memória
  - Solução usada na maioria dos processadores contemporâneos

Hierarquia de memória

- A hierarquia de memória combina uma memória adaptada à velocidade do processador (de pequena dimensão) com uma (ou mais) menos rápida, mas de maior dimensão
- A memória rápida armazena um sub-conjunto da informação residente na memória principal.
- Uma vez que a memória com que o processador interage diretamente é de pequena dimensão, a eficiência da hierarquia resulta do facto de se copiar informação para a memória rápida poucas vezes e de se aceder a essa informação muitas vezes (antes de surgir a necessidade de a substituir)
- Para se tirar partido deste esquema, a probabilidade de a informação (que o processador necessita) estar nos níveis mais elevados da hierarquia tem que ser elevada
- O que torna essa probabilidade elevada ?

Localidade

- **Princípio da localidade**: os programas não acedem à memória (dados e instruções) de forma aleatória mas usam tipicamente endereços que se situam na vizinhança uns dos outros
- Ou seja, num dado intervalo de tempo um programa acede a uma zona reduzida do espaço de endereçamento
- O princípio da localidade manifesta-se de duas formas:
  - **Localidade no espaço (spatial locality)**: Se existe um acesso a um endereço de memória então é provável que os endereços contíguos sejam também acedidos
  - **Localidade no tempo (temporal locality)**: Se existe um acesso a um endereço de memória então é provável que esse mesmo endereço seja acedido novamente no futuro próximo

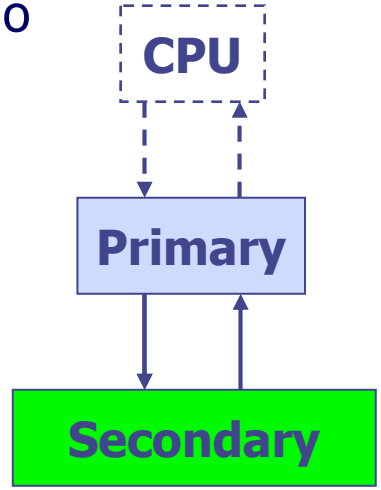
Localidade

- **Localidade espacial:**
  "a informação que o processador necessita de seguida, tem uma elevada probabilidade de estar próxima da que consome agora"
  - Exemplos: instruções de um programa; processamento de um *array*
  - Quanto maior for o bloco (zona contígua de memória) de informação existente na memória rápida, melhor

- **Localidade temporal:**
  "a informação que o processador consome agora tem uma elevada probabilidade de ser novamente necessária num curto espaço de tempo"
  - Exemplos: variável de controlo de um ciclo, instruções de um ciclo
  - Quantos mais blocos de informação estiverem na memória rápida, melhor

Hierarquia de memória com 2 níveis

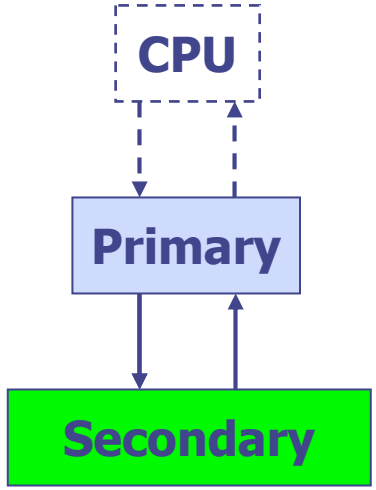
- Nível primário (superior) rápido e de pequena dimensão
- Nível secundário (inferior) mais lento mas de maior dimensão
  - O nível primário contém os blocos de memória mais recentemente utilizados pelo CPU
- Os pedidos de informação são sempre dirigidos ao nível primário, sendo o nível secundário envolvido apenas quando a informação pretendida não está nesse nível
  - Se os dados pretendidos se encontram num bloco do nível primário então existe um "hit"
  - Caso contrário ocorre um "miss"
- Na ocorrência de um "miss" acede-se ao nível secundário e transfere-se o bloco que contém a informação pretendida



Hierarquia de memória com 2 níveis

- A taxa de sucesso (**hit ratio**) é dada por:

$$hit_{ratio} = \frac{nr_{hits}}{nr_{accesses}}$$



- A taxa de insucesso (**miss ratio**) é dada por:

$$miss_{ratio} = 1 - hit_{ratio}$$

- O tempo de acesso no caso de um "hit" designa-se **hit time**
- O tempo de substituir um bloco do nível superior e enviar os dados para o processador é designado por **penalty time** (**miss penalty**)
- De um modo simplificado, o "penalty time" é dado por:

$$penalty_{time} = hit_{time} + t_{mem}$$

em que "t_{mem}" é o tempo de acesso ao nível secundário

Hierarquia de memória com 2 níveis

- O tempo médio de acesso à informação é então:

$$T_{access} = hit_{ratio} * hit_{time} + (1 - hit_{ratio}) * penalty_{time}$$

$$T_{access} = hit_{ratio} * hit_{time} + (1 - hit_{ratio}) * (hit_{time} + t_{mem})$$

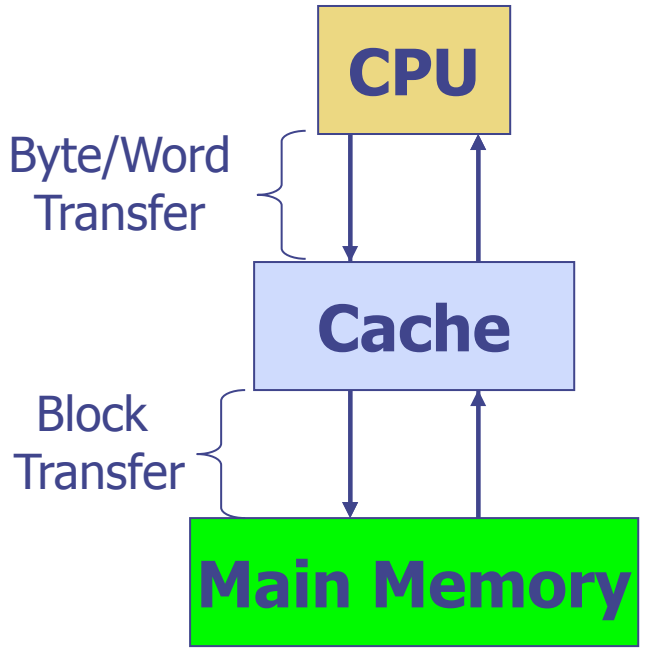
- **Exercício**: Assumindo um *hit_ratio* de 95%, um tempo de acesso ao nível superior de 5ns e um tempo de acesso ao nível inferior de 50ns, calcular o tempo médio de acesso à memória
  - Ta = 0,95 * 5 + 0,05 * (50 + 5) = 7,5ns

  Ou seja, aproximadamente 6,7 vezes mais rápido que o acesso direto ao nível secundário

- Quando o acesso a uma palavra no nível superior falha, acede-se ao nível seguinte não apenas a essa palavra, mas também às que estão em endereços adjacentes (vizinhas) - **bloco**

Memória cache

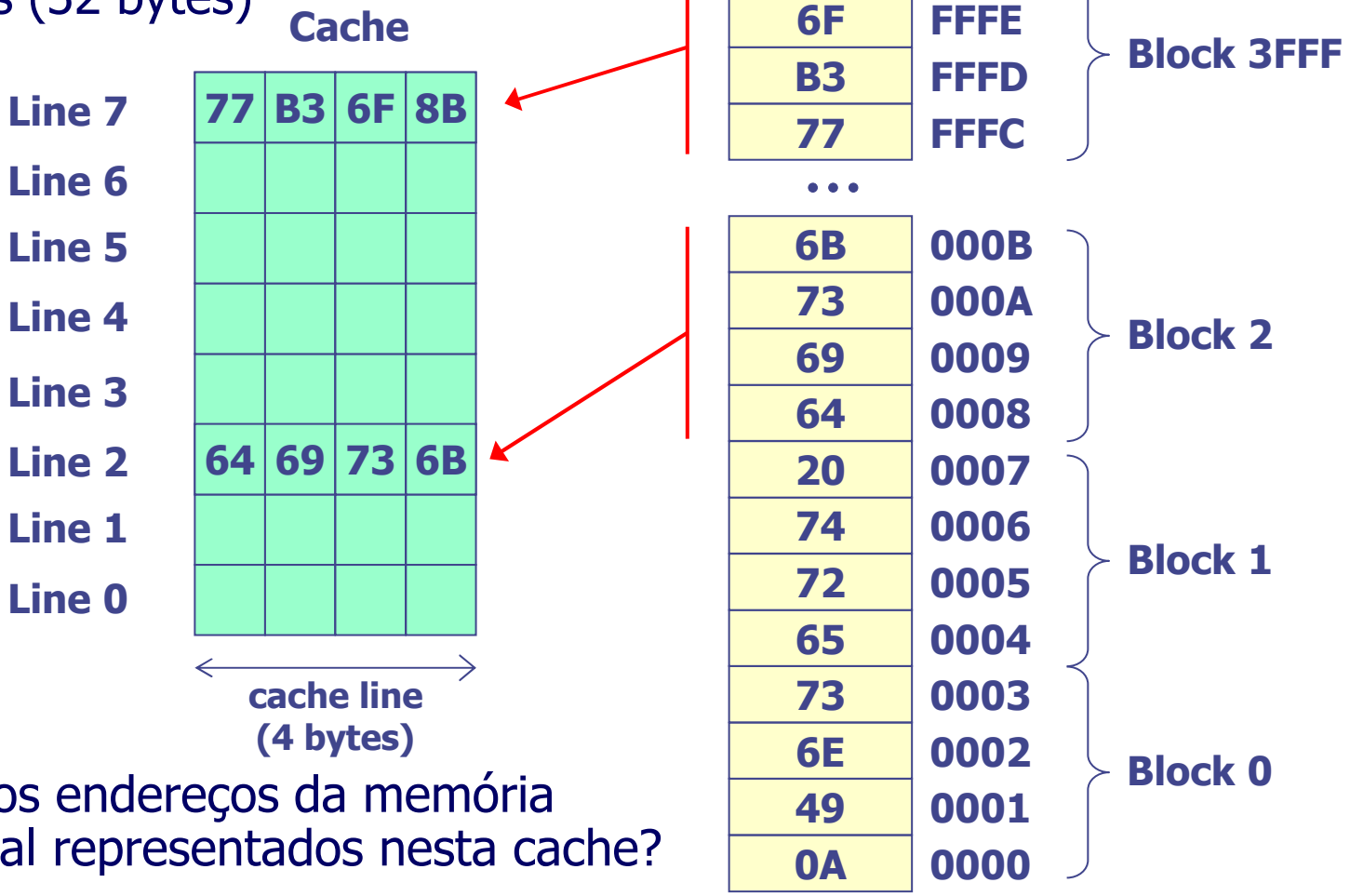
- **cache**: nível de memória que se encontra entre o CPU e a memória principal (primeiras utilizações no início da década de 60)
- Exemplo - **cache read**:
  - Cache recebe endereço (MemAddr) do CPU
  - O bloco que contém MemAddr está na cache?
  - **SIM (hit)**:
    - Lê a cache e envia para o CPU o conteúdo de MemAddr
  - **NÃO (miss)**:
    - Encontra espaço na cache para um novo bloco
    - Acede à memória principal e lê o bloco que contém o endereço MemAddr
    - Conteúdo de Memaddr é enviado para o CPU



- É comum os computadores recentes incluírem 2 ou mais níveis de cache; a cache é normalmente integrada no mesmo circuito integrado do processador

Memória cache

- Exemplo: memória de 64K (16 bits de endereço), cache de 8 linhas de 4 bytes (32 bytes)



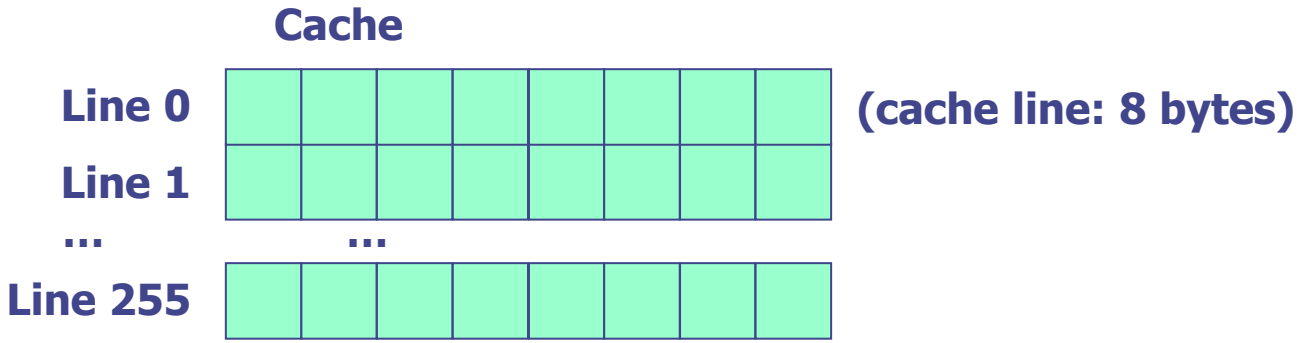
- Quais os endereços da memória principal representados nesta cache?

Memória cache

- As operações da cache são transparentes para o processador
- O processador gera um endereço e pede que seja feita uma operação de leitura ou escrita e esse pedido é satisfeito pelo sistema de memória:
  - é desconhecido para o processador se é a cache ou a memória principal a satisfazer o pedido
- Na organização da cache e na implementação da respetiva unidade de controlo é necessário tomar em consideração um conjunto de aspetos, nomeadamente:
  - Como saber se um determinado endereço está na cache?
  - Se está na cache, onde é que está?
  - Quando ocorre um miss, onde colocar um novo bloco na cache?
  - Quando ocorre um miss, qual o bloco a retirar da cache?
  - Como tratar o problema das operações de escrita de modo a manter a coerência da informação nos vários níveis?
- Implementação tem de ser eficiente! Realizável em hardware

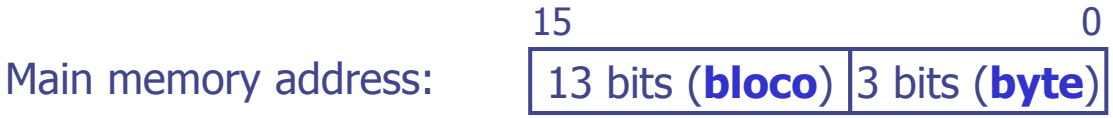
Organização dos sistemas de cache

- Há basicamente 3 formas de organizar os sistemas de cache:
  - Cache **totalmente associativa** ("fully associative")
  - Cache **com mapeamento direto** ("direct mapped")
  - Cache **parcialmente associativa** ("set associative")
- A metodologia de organização de cada uma delas é apresentada a seguir, partindo do seguinte conjunto de especificações-base:
  - Espaço de endereçamento de 16 bits (memória principal organizada em bytes, com 64 kBytes)
  - Memória cache de 2 kBytes, em que cada linha tem 8 bytes (2^3) (ou seja, cada bloco tem uma dimensão de 8 bytes)



Organização dos sistemas de cache

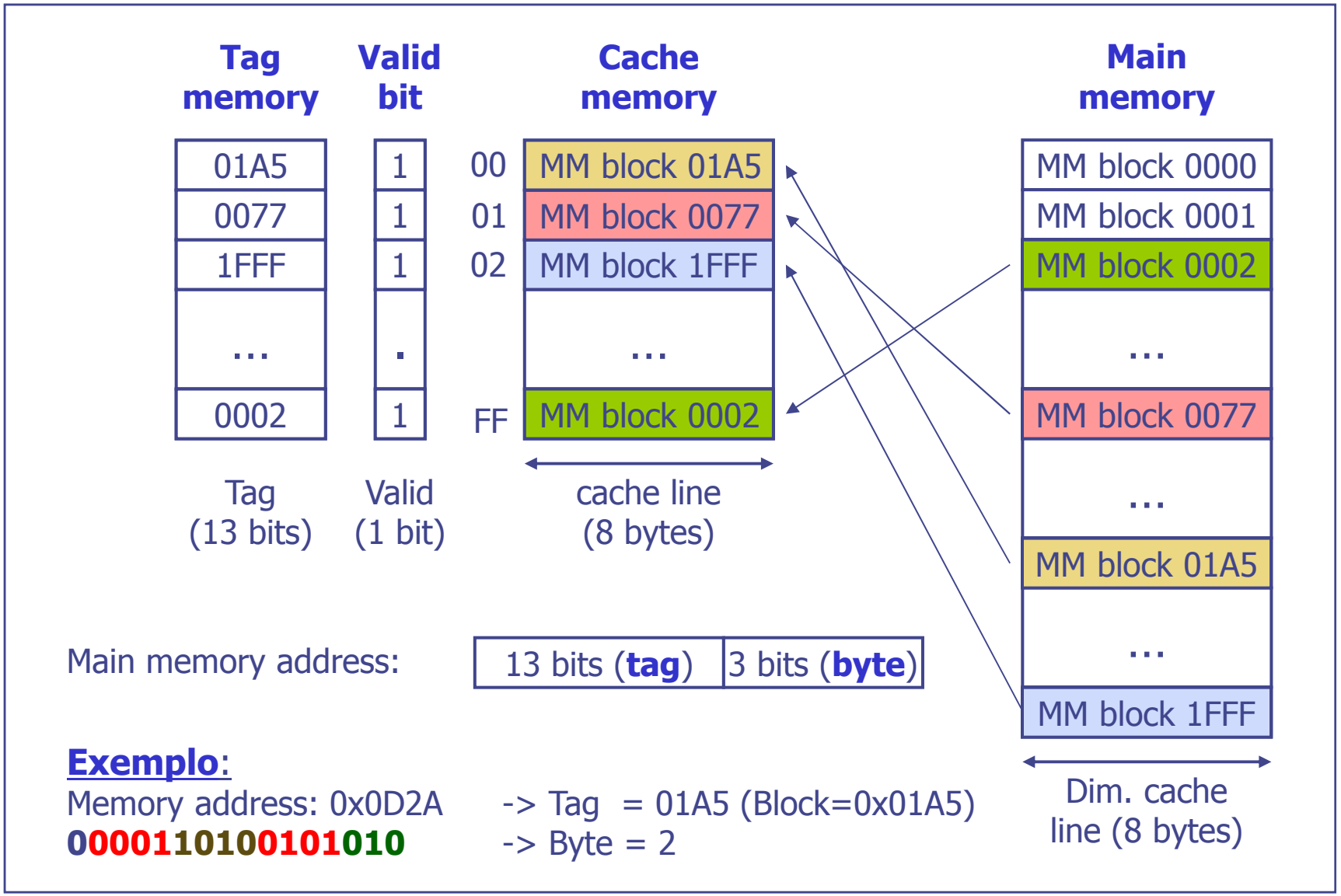
- Especificação-base para os exemplos de organização que se seguem:
  - Espaço de endereçamento de 16 bits (0x0000 a 0xFFFF)
  - Memória cache de 2 kBytes, em que cada linha tem 8 bytes (2^3), significa que a cache tem 256 linhas (2^{11} / 2^3 = 256)
- Com estes valores, a memória principal pode ser vista como sendo constituída por um conjunto de 2^{13} blocos contíguos, de 8 bytes cada ($2^{16}/2^3 = 2^{13}$): 8k blocos (numerados de 0000 a 0x1FFF)
- Assim, no endereço de 16 bits, os
  - 3 bits menos significativos identificam o **byte dentro do bloco**
  - 13 bits mais significativos identificam o **bloco**



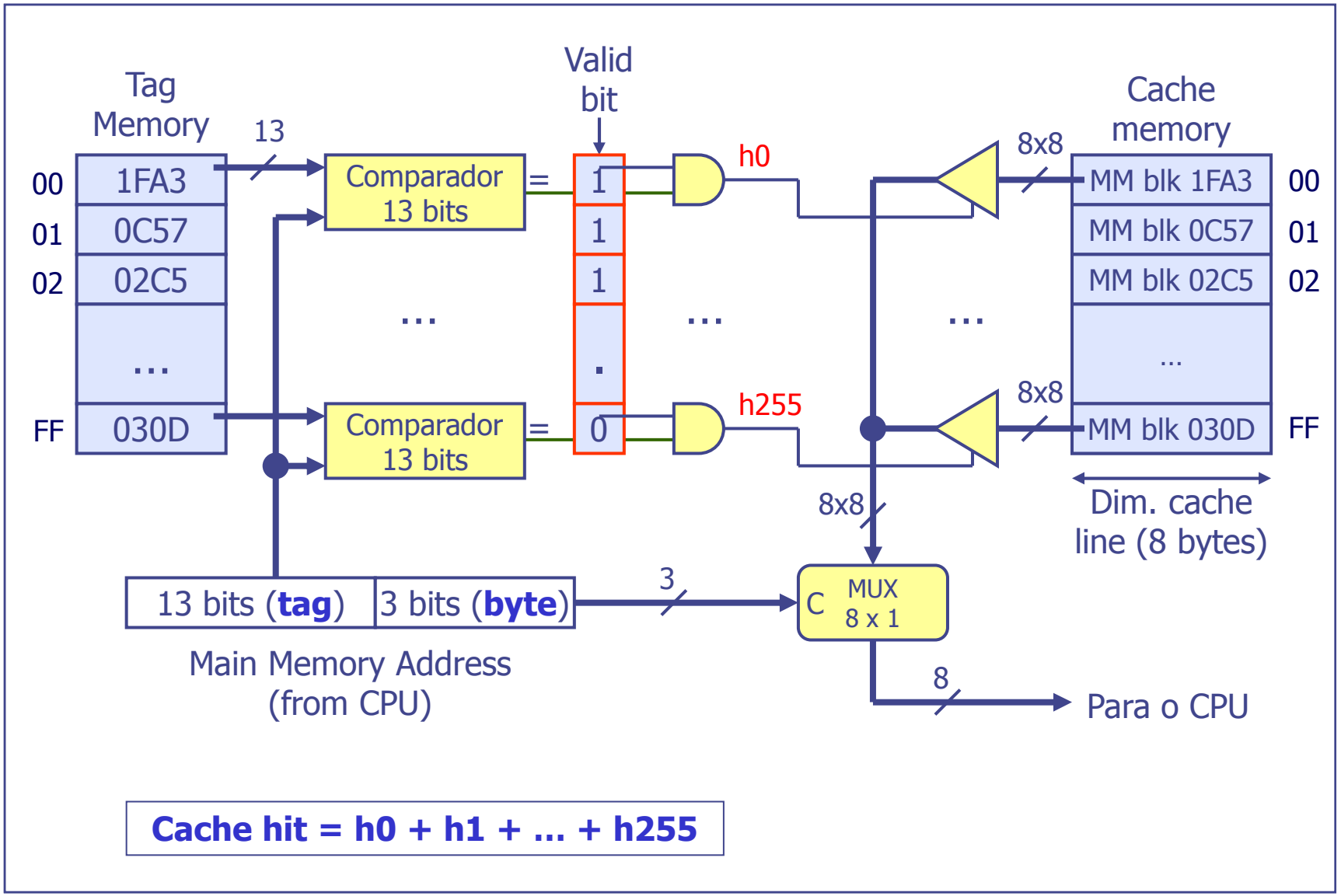
- Exemplo. Address = 0x001A : **0000000000011010**

  Bloco 3, de 0x0018 a 0x001F, byte 2 dentro do bloco

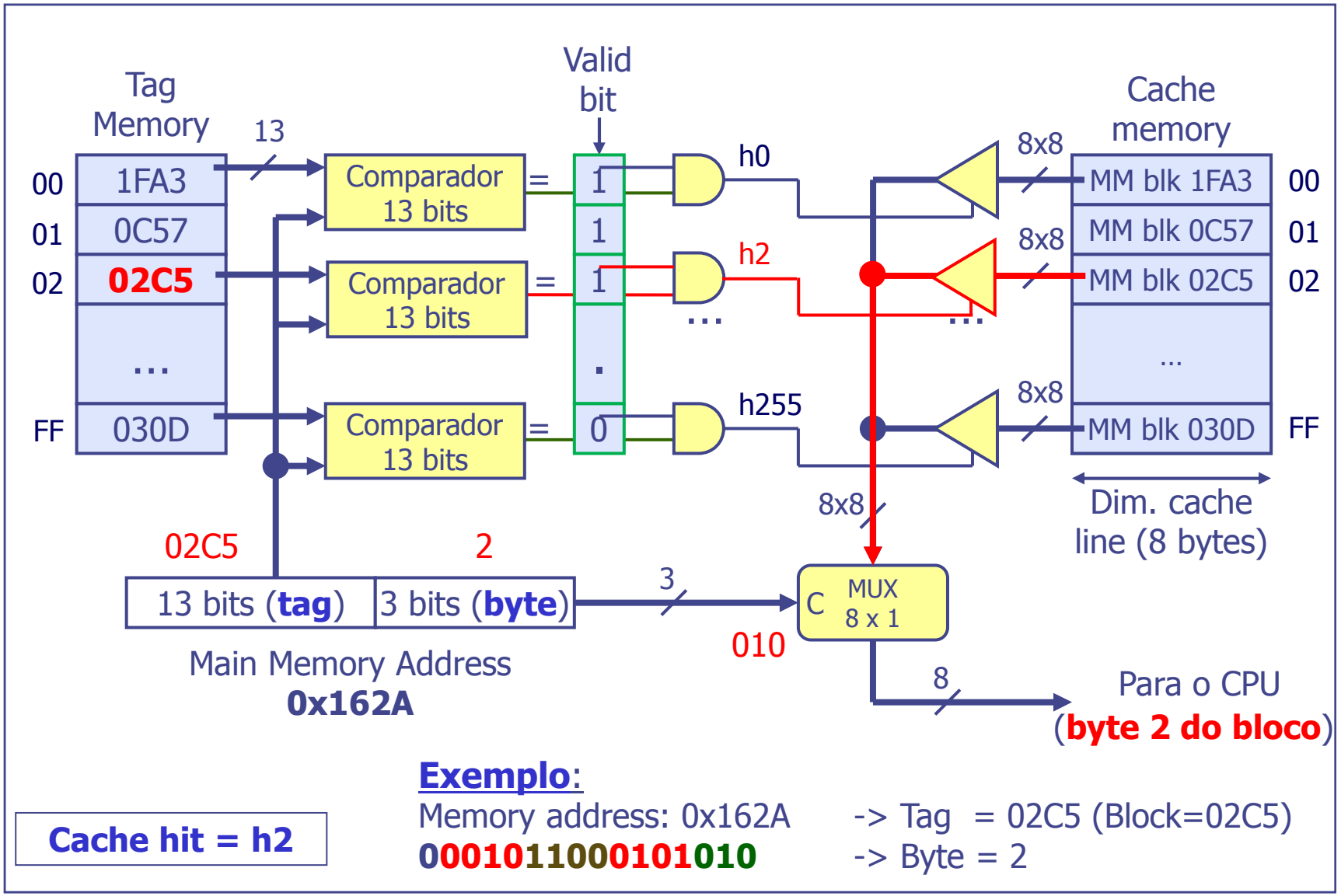
Cache com mapeamento associativo



Cache com mapeamento associativo



Cache com mapeamento associativo (exemplo)



Exemplo:

Memory address: 0x162A -> Tag = 02C5 (Block=02C5)

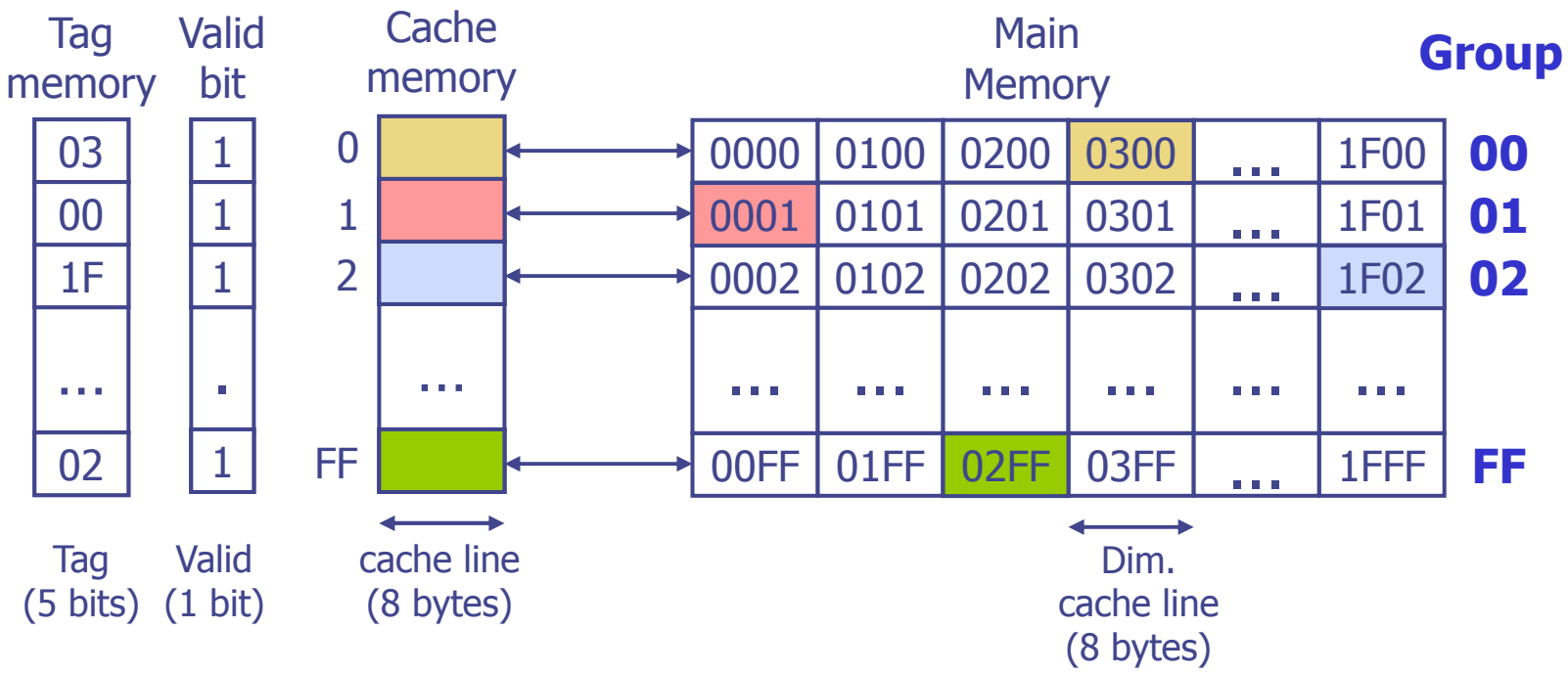
0001011000101010 -> Byte = 2

Cache hit = h2

Cache com mapeamento associativo

- Vantagens:
  - Qualquer bloco da memória principal pode ser colocado em qualquer posição da cache
- Inconvenientes:
  - A "tag" tem que ter todos os bits do número do bloco, o que numa implementação realista origina comparadores com uma dimensão elevada
  - Todas as entradas da memória "tag" têm de ser analisadas, de forma a verificar se um endereço se encontra na cache
  - Muitos comparadores, custo elevado

Cache com mapeamento direto



Main memory address:

5 bits (**tag**)	8 bits (**group**)	3 bits (**byte**)

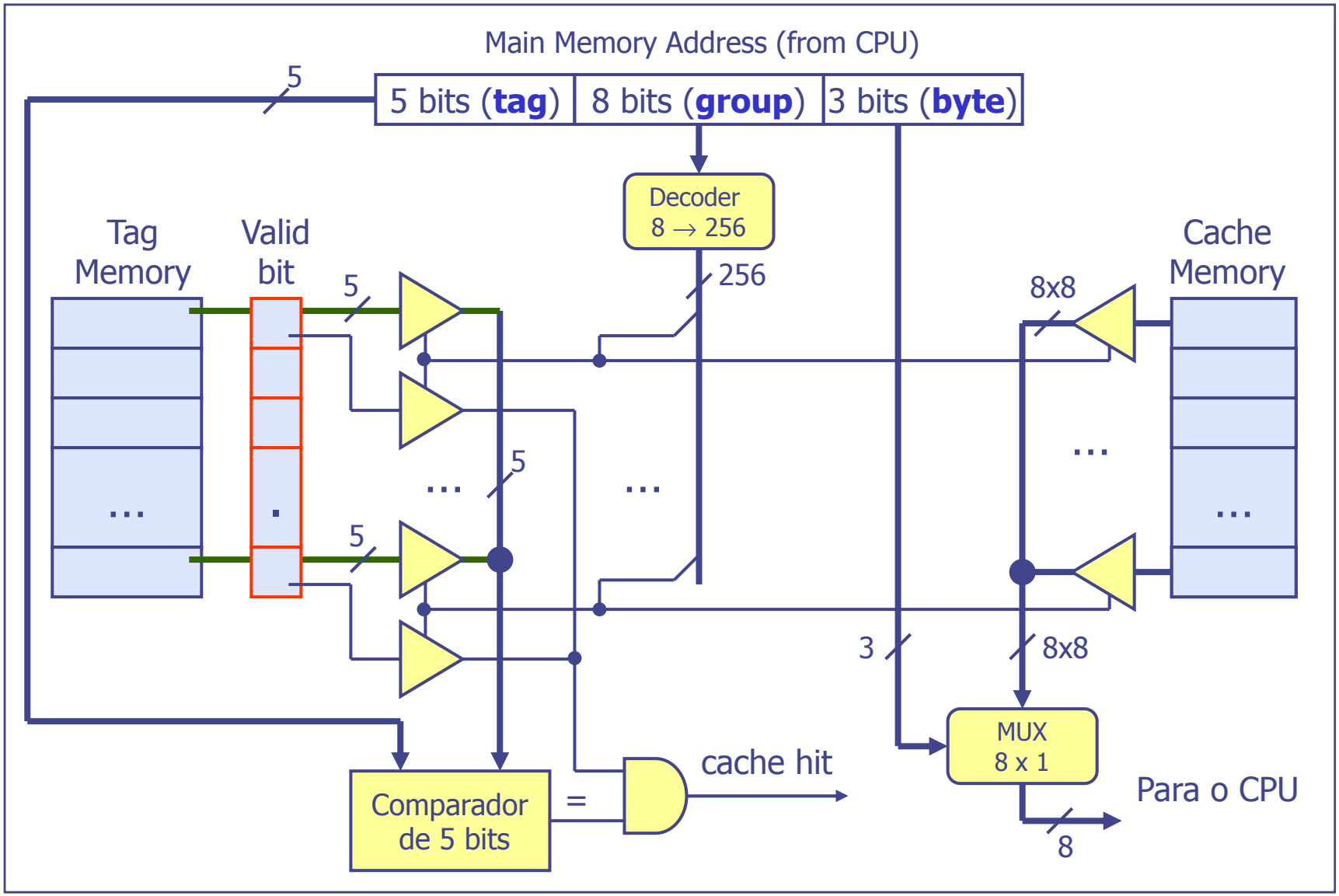
Exemplo:

Memory address = 0xF813 -> Tag = 0x1F (Block = 0x1F02)

1111100000010**011** -> Group = 0x02

-> Byte = 3

Cache com mapeamento direto



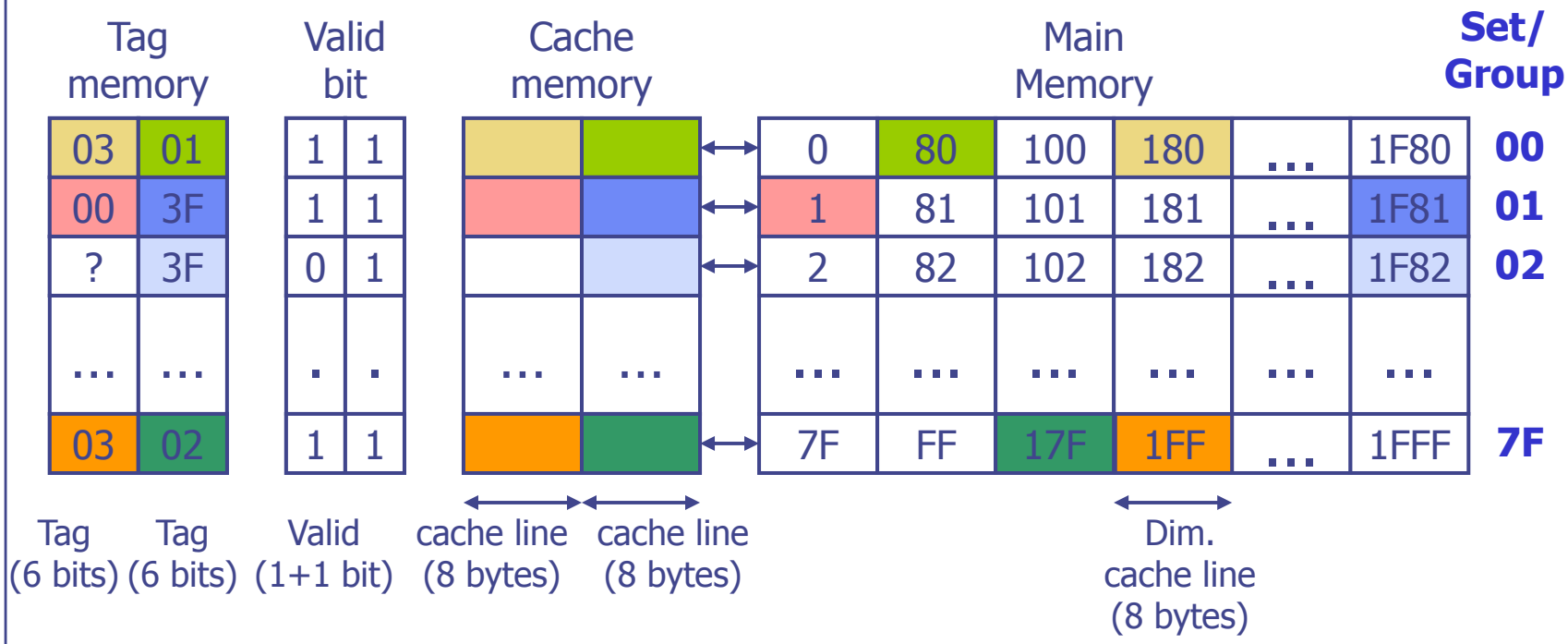
Cache com mapeamento direto

- A cada bloco da memória principal é associada uma linha da cache; o mesmo bloco é sempre colocado na mesma linha
- Maior vantagem:
  - Simplicidade de implementação
- Maior desvantagem:
  - Vários blocos têm associada a mesma linha
  - Num dado instante apenas um bloco de um dado grupo pode residir na cache, o que tem como consequência que alguns blocos podem ser substituídos e recarregados várias vezes, mesmo existindo espaço na cache para guardar todos os blocos em uso

- Uma melhoria óbvia deste tipo de cache seria permitir o armazenamento simultâneo de mais que um bloco do mesmo grupo
- É esta melhoria que é explorada na cache **parcialmente associativa ("set associative")**

Cache com mapeamento parcialmente associativo

(exemplo com associatividade de 2 ⇔ 2 vias)



Main memory address:

6 bits (**tag**)	7 bits (**set**)	3 bits (**byte**)

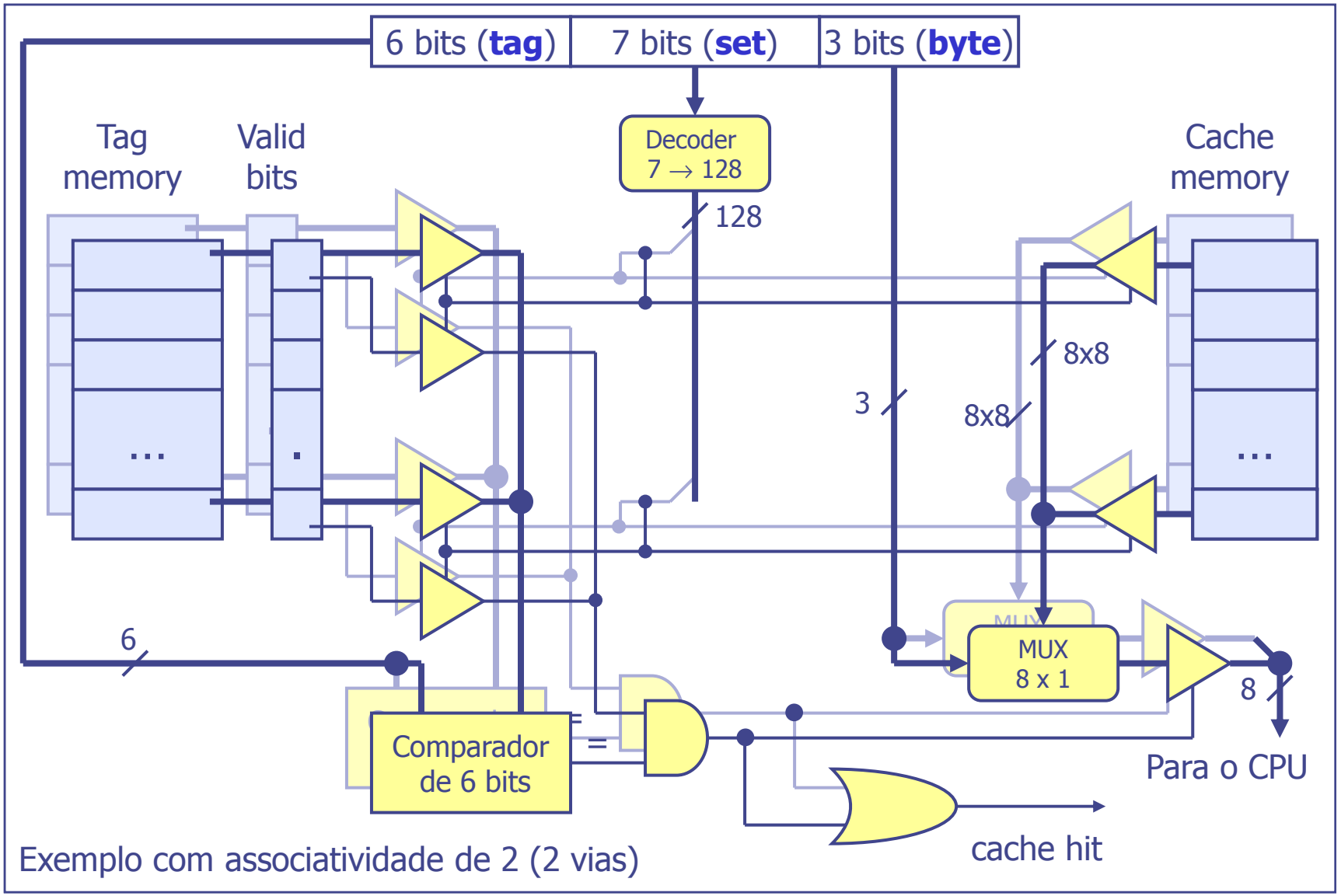
Exemplo:

Memory address = 0xFC16 -> Tag = 0x3F (Block=0x1F82)

1111110000010110 -> Set/Group = 0x02

-> Byte = 6

Cache com mapeamento parcialmente associativo



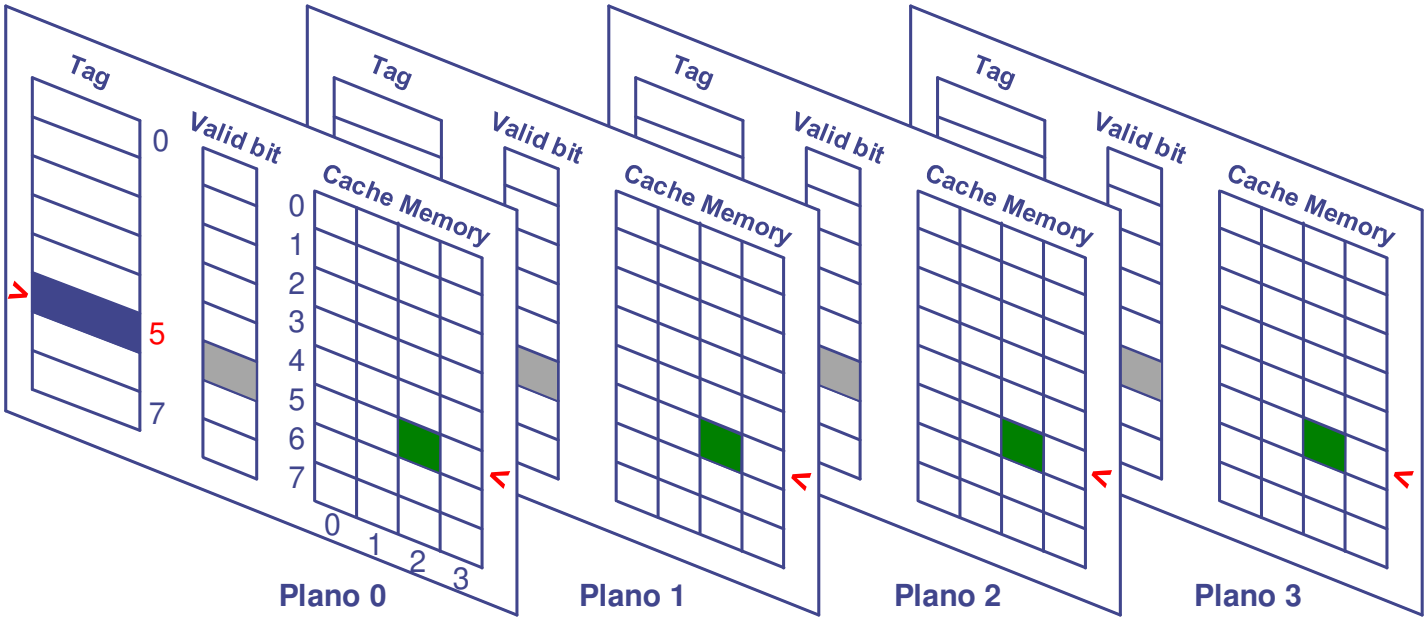
Cache parcialmente associativa

- A cache parcialmente associativa é semelhante à cache com mapeamento direto, mas a primeira permite que mais que um bloco de um mesmo grupo possa ser carregado na cache
- Uma cache com associatividade de 2, ou seja, com 2 vias, permite que 2 blocos do mesmo grupo possam estar simultaneamente na cache
- A divisão do endereço de memória é igual ao do mapeamento direto (o campo "group" é normalmente designado por "set" - conjunto), mas agora há "n" possíveis lugares onde um dado bloco de um mesmo grupo pode residir; "n" é o número de vias da cache e nesse caso diz-se que a cache tem associatividade de "n"
- As "n" vias onde o bloco pode residir têm que ser procuradas simultaneamente; o hit da cache resulta do OR dos hits de cada uma das vias
- (Block-set associative cache / n-way-set associative cache)

Cache com mapeamento parcialmente associativo

S

- Esquematização de uma cache com **associatividade de 4**, de **128 bytes**, com **blocos de 4 bytes** (128 / 4 = 32 bytes por via, 32 / 4 = 8 linhas)
- Endereço gerado pelo CPU (16 bits) 0x6A96: **0110101010010110**



- A comparação dos 11 bits mais significativos (A15-A5) do endereço com as "tags" é feita simultaneamente nas 4 vias
- A posição da memória "tag" onde é feita a comparação é determinada pelos bits A4-A2 do endereço (posição 5 no exemplo)
- A via onde ocorre o hit (se ocorrer) fornece o byte armazenado na cache na posição determinada pelos bits A1-A0 do endereço (posição 2 no exemplo)

Políticas de substituição de blocos

- As políticas de substituição de blocos na cache, na ocorrência de um miss, são várias. As mais utilizadas são as seguintes:
- **FIFO** – First in first out: é substituído o bloco que foi carregado há mais tempo
- **LRU – Least Recently Used**: é substituído o bloco da cache que está há mais tempo sem ser referenciado
- **LFU – Least Frequently Used**: substituído o bloco menos acedido
- **Random**: substituição aleatória (testes indicam que não é muito pior do que LRU)

Políticas de escrita

- **Write-through**
  - Todas as escritas são realizadas simultaneamente na cache e na memória principal
  - Se endereço ausente na cache, atualiza apenas a memória principal (**write-no-allocate**)
  - A memória principal está sempre consistente
- **Write-back**
  - Valor escrito apenas na cache; novo valor é escrito na memória quando o bloco da cache é substituído
  - "**Dirty bit**" (este bit é ativado quando houver uma escrita em qualquer endereço do bloco presente na linha da cache)
  - Se endereço ausente na cache, carrega o bloco para a cache e atualiza-o (**write-allocate**)
  - Mais complexo do que "write-through"

Exemplo – Intel Core i7-6500U

S

- Número de cores: 2
  - Core 1: L1 data (32 KB), L1 instr (32 KB), L2 data+instr (256 KB)
  - Core 2: L1 data (32 KB), L1 instr (32 KB), L2 data+instr (256 KB)
  - L3: (4 MB) partilhada pelos 2 cores

Cache	Size	Associativity	Line Size
L1 Data	2 x 32 KB	8-way set associative	64 bytes
L1 Instructions	2 x 32 KB	8-way set associative	64 bytes
L2	2 x 256 KB	4-way set associative	64 bytes
L3	4 MB (shared cache)	16-way set associative	64 bytes

- Qual a dimensão do bloco das caches L1, L2 e L3?
- Qual o número de linhas das caches L1, L2 e L3?

Exercícios

S

1. Qual a posição dos endereços de bloco 1 e 29 numa cache de mapeamento direto com 8 linhas?
2. Considere um sistema computacional com um espaço de endereçamento de 32 bits e uma cache com 256 blocos de 8 bytes cada um.
   - Qual o tamanho em bits dos campos tag, group e byte supondo que a cache é: 1) associativa; 2) mapeamento direto; 3) com associatividade de 2.
3. Para a implementação de uma cache com 64KB de dados e blocos de 4 bytes num sistema computacional com um espaço de endereçamento de 32 bits, quantos bits de armazenamento são necessários supondo:
   - uma cache associativa;
   - uma cache com mapeamento direto;
   - uma cache com associatividade de 2.

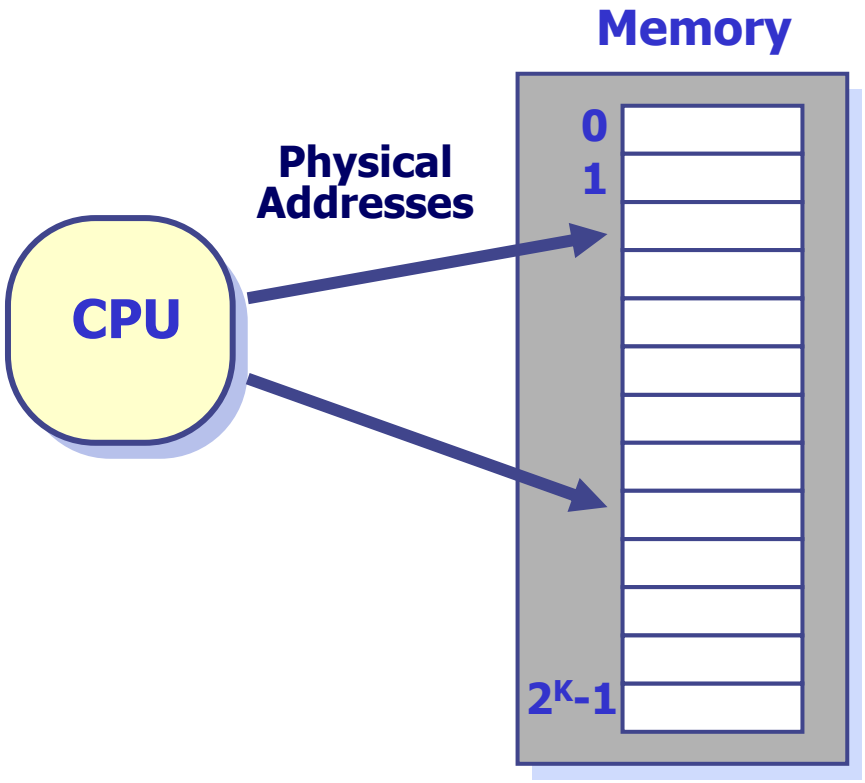
Aulas 22, 23 e 24

- Memória Virtual
  - Motivações para a sua utilização
  - Endereço virtual, endereço físico
  - *Page Table*. Tradução de endereços. *Page Fault*
  - "Translation-lookaside buffer" (TLB)
  - Integração da memória virtual com TLB e *cache*

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

Sistema apenas com memória física

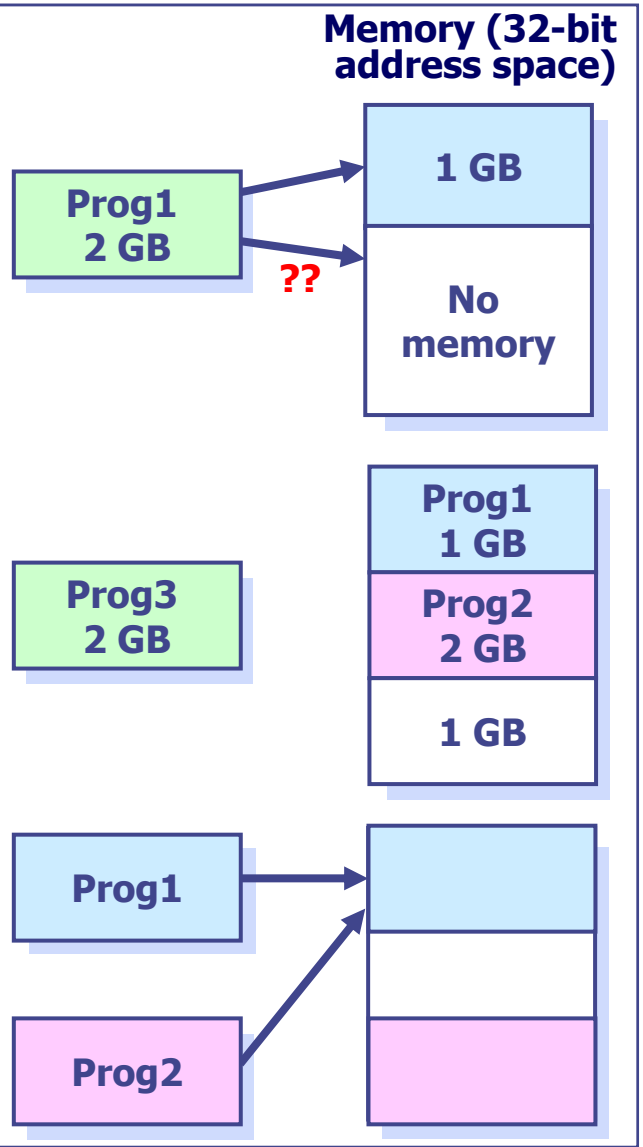
- Num sistema apenas com memória física



- Os endereços gerados pelo CPU apontam diretamente para as posições de memória a que este pretende aceder

Sistema apenas com memória física – problemas

- Memória insuficiente
  - O que acontece se o programa Prog1 tentar aceder a mais do que 1 GB de memória?
- Gestão da memória disponível
  - Supondo que o sistema tem 4 GB de memória, os programas Prog1 e Prog2 usam 3 GB. Quando Prog1 terminar ficam 2GB disponíveis, mas Prog3 não pode executar porque não existe um bloco de 2 GB
  - Fragmentação da memória
- Segurança
  - Um programa pode escrever fora da zona de memória que lhe está atribuída



Sistema apenas com memória física – problemas

- Num sistema multi-processo, a memória disponível tem de ser partilhada pelos processos em execução
  - Cada processo tem as suas próprias necessidades de memória
- A memória pode facilmente ser corrompida se um processo escrever (de forma involuntária ou intencional) na zona de memória atribuída a outro processo
  - O processo que viu a sua zona de memória alterada falhará por razões que nada têm a ver com a sua lógica de funcionamento
- Como gerir adequadamente a memória disponível entre os vários processos?
- O que fazer se a memória necessária para executar todos os processos for superior à memória física disponível?
- Como garantir que um processo não escreve na zona de memória atribuída a outro processo?

Memória virtual

- É uma abstração que permite uma utilização eficiente do sistema de memória em sistemas multi-processo (mantendo o modelo de memória hierárquica)
- Esta técnica é usada em sistemas de computação geral baseados em microprocessador, com sistema operativo (não usado em sistemas simples baseados em microcontrolador)
- O conceito de "memória virtual" não é novo: descrito pela primeira vez por Kilburn et al. em 1962

Motivações para a utilização de Memória virtual

- Eficiência na utilização da memória
  - Memória deve ser partilhada pelos vários processos em execução
  - Em memória apenas deve residir a informação necessária
- Segurança
  - Devem existir mecanismos de segurança que impeçam que um processo altere as zonas de memória dos outros processos
- Transparência
  - Um processo deve ter acesso à memória de que necessita (eventualmente mais do que a memória física DRAM)
  - Um processo deve correr como se toda a memória lhe pertencesse
- Partilha de memória
  - Vários processos devem poder aceder à mesma zona de memória (de forma controlada)

Memória virtual

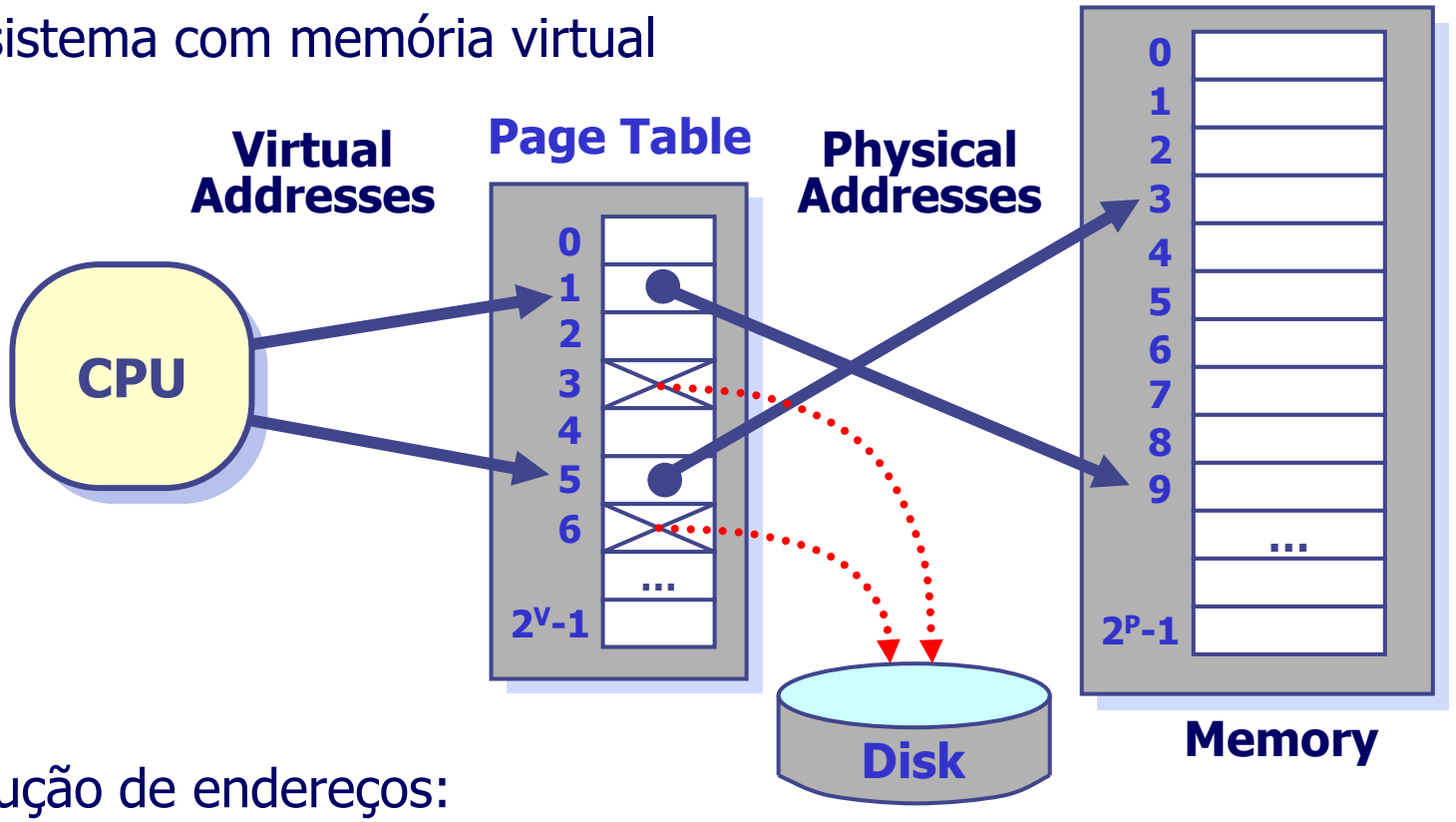
- Utilização eficiente da memória física:
  - Vários programas podem estar em execução "simultânea" no computador
  - A memória pode estar fragmentada
  - A quantidade de memória total necessária para executar todos os programas pode ser superior à memória física disponível
  - No entanto, apenas uma fração dessa quantidade de memória está ativamente a ser usada num determinado instante de tempo
  - Assim, na memória principal apenas é necessário ter as "zonas ativas" de todos os programas (instruções e dados) a correr no computador (as restantes podem estar em disco)
  - O sistema operativo pode reservar mais memória para um processo ou libertar memória que já não esteja a ser usada, de acordo com as necessidades

Memória virtual

- Segurança:
  - Múltiplos processos partilham a mesma memória física
  - É necessário usar mecanismos de proteção que assegurem que um dado processo apenas acede (leitura e/ou escrita) às zonas de memória que lhe foram atribuídas
  - Endereços usados pelos processos não são endereços da memória física
- Transparência
  - Cada processo tem o seu próprio espaço de endereçamento que pode usar na totalidade, de forma exclusiva
  - Cada processo executa como se fosse o único a ocupar a memória
- Partilha de memória
  - Com a implementação de mecanismos de proteção torna-se possível a partilha de zonas de memória entre processos

Princípio de funcionamento

- Um sistema com memória virtual



- Tradução de endereços:
  - Os endereços virtuais gerados pelo CPU são convertidos para endereços físicos através de uma tabela, designada por "Page Table"
  - A tradução de endereços tem que ser temporalmente eficiente

Nomenclatura-base

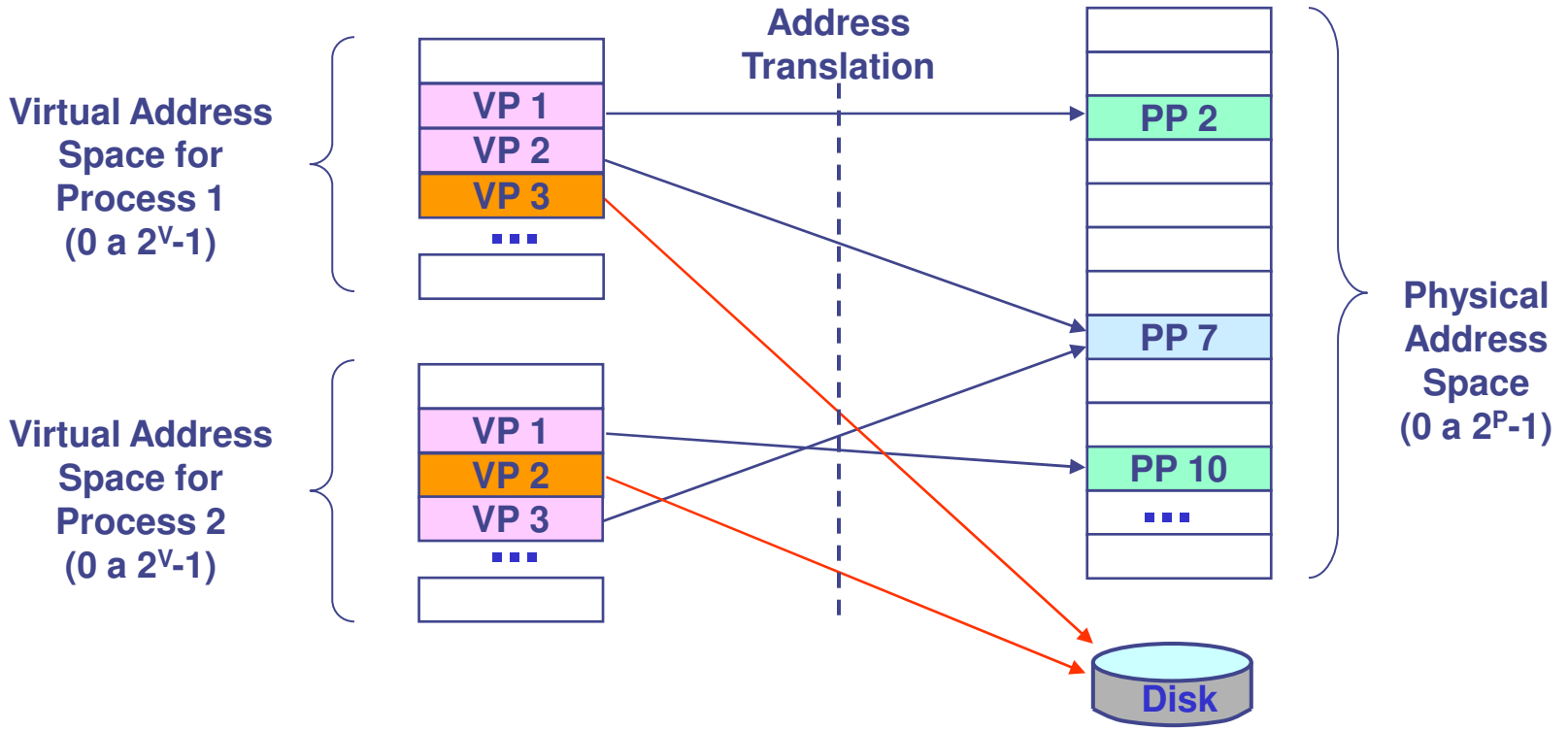
- **Endereço virtual** (*virtual address*)
  - Um endereço gerado pelo CPU
  - O **espaço de endereçamento virtual** é a coleção de todos os endereços virtuais (0 a 2^V-1, num espaço de endereçamento de V bits)
- **Endereço físico** (*physical address*)
  - Um endereço da memória principal (e.g. DRAM) ou do disco
  - O **espaço de endereçamento físico** é a coleção de todos os endereços físicos (0 a 2^P-1, num espaço de endereçamento de P bits)
- **Tradução de endereços** (*address translation*)
  - O processo pelo qual um endereço virtual é traduzido num endereço físico
  - Um endereço virtual é traduzido num endereço físico quando o CPU acede à memória (para ler uma instrução ou para aceder a uma palavra de dados)

Princípio de funcionamento

- Os espaços de endereçamento virtual e físico são divididos em blocos; os blocos têm a mesma dimensão nos dois espaços de endereçamento
  - Na terminologia de memória virtual estes blocos designam-se por "**páginas**"
- Cada processo tem o seu próprio **espaço de endereçamento virtual**
  - O espaço de endereçamento virtual de todos os processos pode seguir o mesmo modelo: ter início no mesmo endereço, ter a primeira instrução no mesmo endereço, ter o *data segment* na mesma zona da memória, etc.
  - A atribuição de **páginas físicas** a **páginas virtuais** é feita pelo sistema operativo
  - Quando o processo é carregado em memória, o sistema operativo faz a atribuição dos endereços físicos a endereços virtuais (**relocation** – construção da *Page Table*)
  - A relocação permite que um programa possa ser carregado em qualquer página de memória

Princípio de funcionamento

- Espaço de endereçamento virtual de V bits
- Espaço de endereçamento físico de P bits

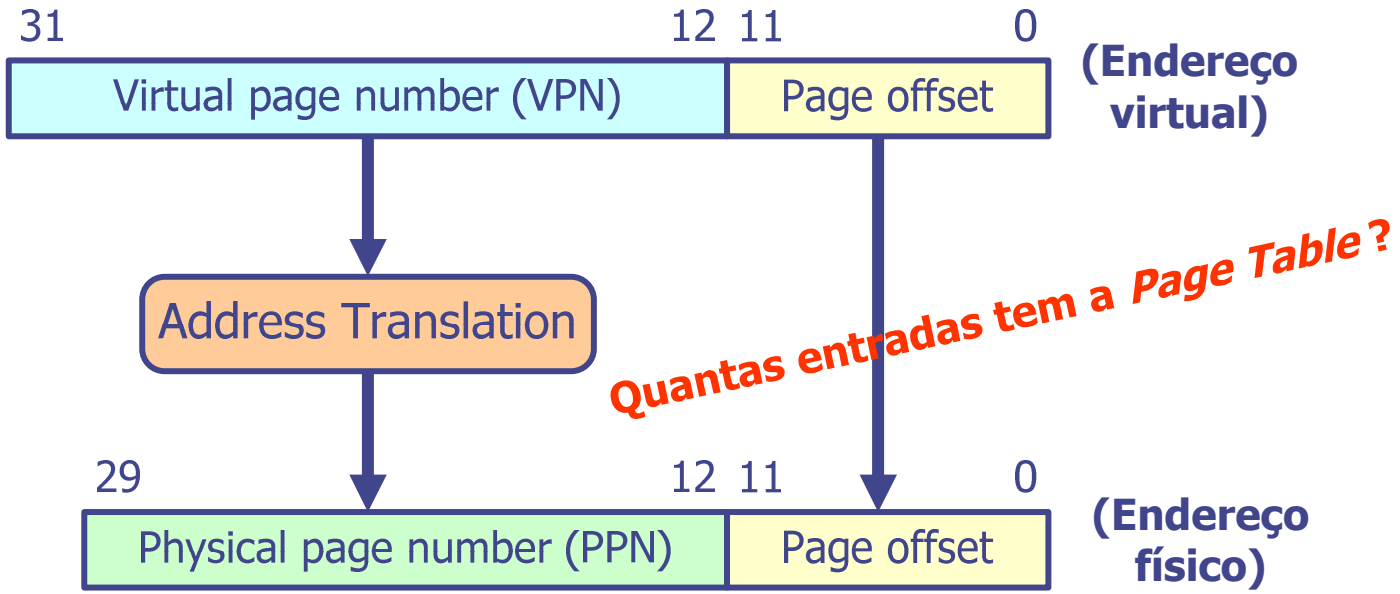


VP ≡ Virtual page
PP ≡ Physical page

Neste exemplo a página física 7 é partilhada pelos dois processos com endereços virtuais diferentes (exemplo de um zona de memória partilhada: leitura ou leitura/escrita)

Princípio de funcionamento

- Mapeamento entre um endereço virtual (V bits) e um endereço físico (P bits) - exemplo c/ **V = 32 bits** e **P = 30 bits** e páginas de 4 kBytes

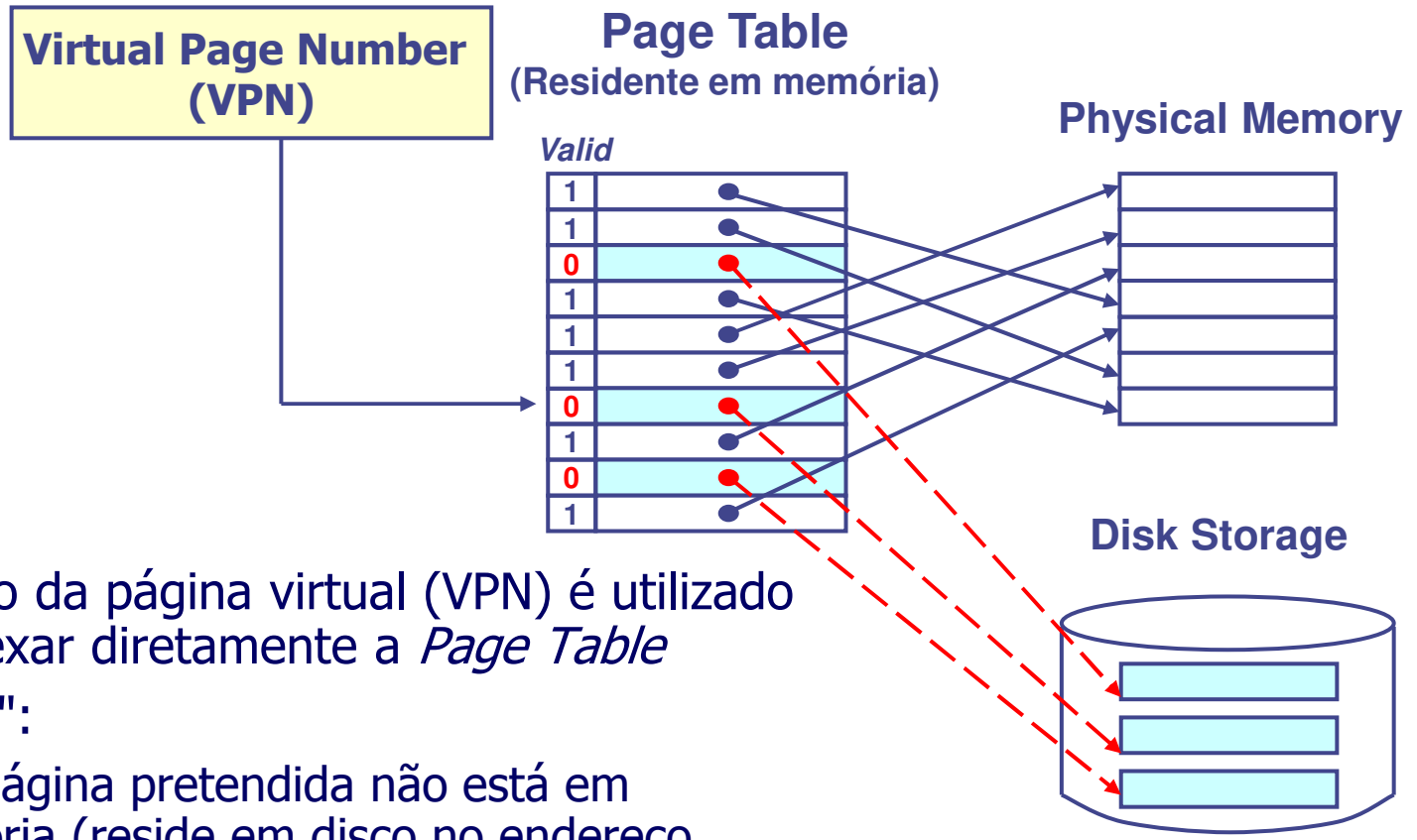


  - Espaço de endereçamento virtual: 2^{32} = 4 GB
  - Espaço de endereçamento físico: 2^{30} = 1 GB
  - Dimensão da página: 2^{12} = 4 kB
  - Número de páginas da memória virtual: 2^{20} = 1 M
  - Número de páginas da memória física: 2^{18} = 256k

Tradução de endereços

- A *Page Table* contém um número de entradas igual ao número máximo de páginas virtuais (para o exemplo dado anteriormente, a tabela teria 2^{20} entradas). Por esta razão não é necessária uma "tag"
- O número da página virtual (VPN) é utilizado para indexar diretamente a *Page Table*
- A tradução de endereços tem que ser rápida, uma vez que ocorre em cada acesso do CPU à memória. Realizada por hardware!

Tradução de endereços

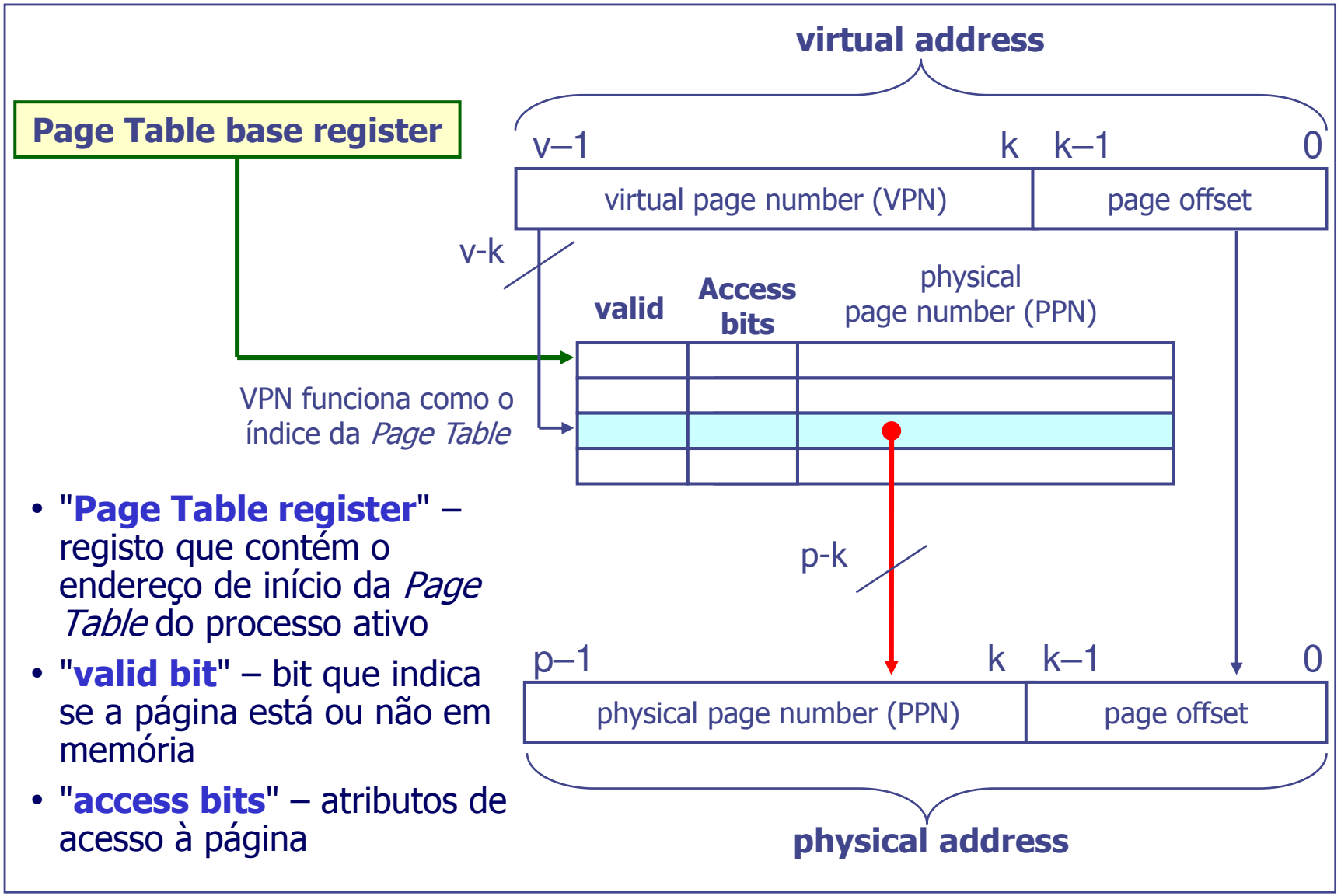


- O número da página virtual (VPN) é utilizado para indexar diretamente a *Page Table*
- "Valid bit":
  - **0**: a página pretendida não está em memória (reside em disco no endereço especificado – *Page Fault*, ou não existe)
  - **1**: a *Page Table* fornece o número da página física (a partir do qual se obtém o endereço inicial da página em memória)

Operação

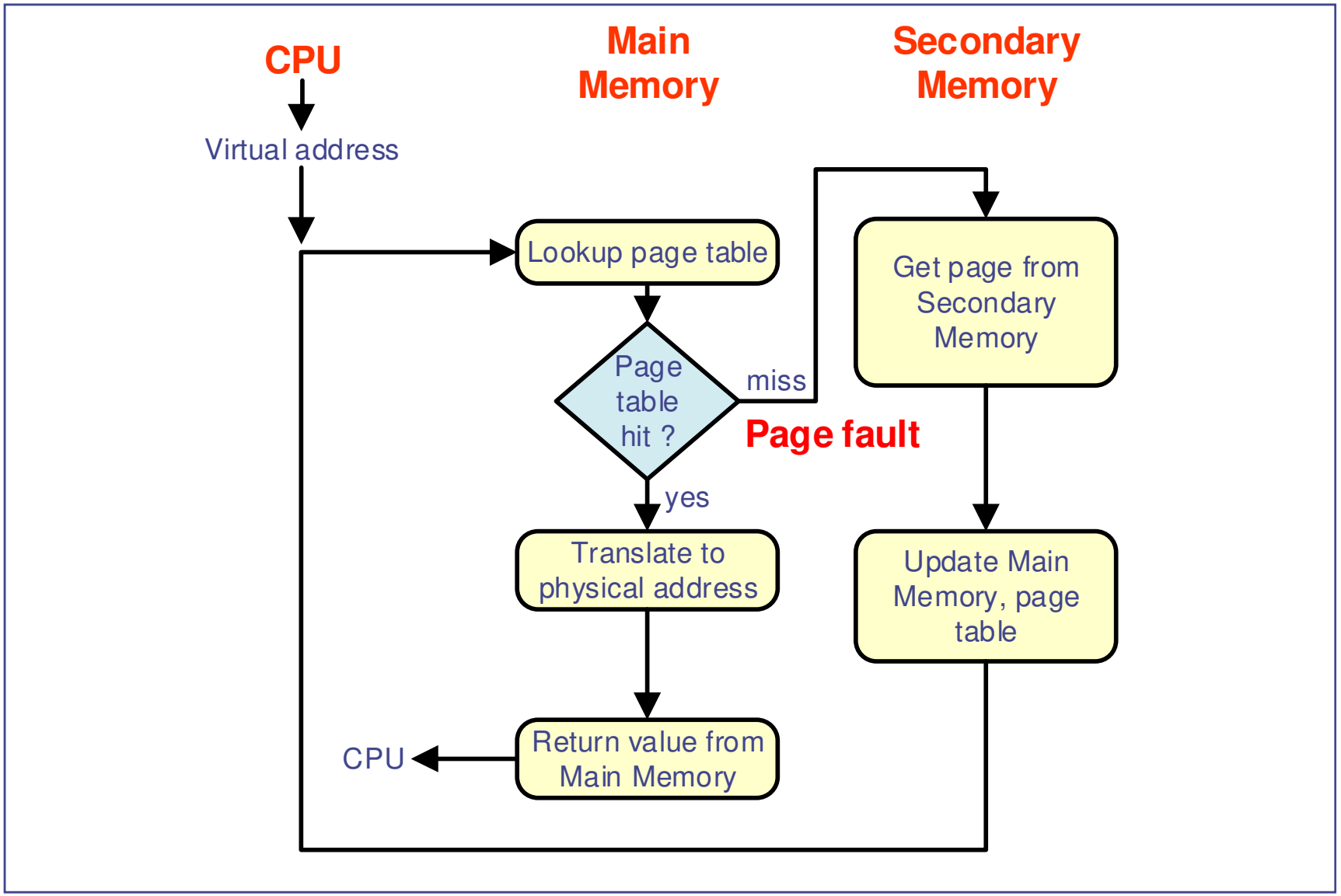
- Tradução
  - Uma *Page Table* por processo
  - O Virtual Page Number (VPN), obtido do endereço gerado pelo processador, forma o índice de acesso à *Page Table*
- Obtenção do endereço físico da página
  - A entrada da tabela correspondente ao VPN contém informação sobre a página
  - Se "valid bit = 1" a página reside na memória
    - O endereço de acesso é obtido da entrada da tabela correspondente ao VPN concatenado com o "page offset"
  - Se "valid bit = 0" a página reside no disco ou não existe
    - *Page Fault*
    - Copiar para a memória a página pretendida (pode envolver a substituição de uma página residente em memória)
    - "valid bit" passa a "1"
- Proteção
  - Os atributos de acesso à página (Read, Read/Write, Execute) são verificados em cada tradução (para cada VPN)

Tradução de endereços



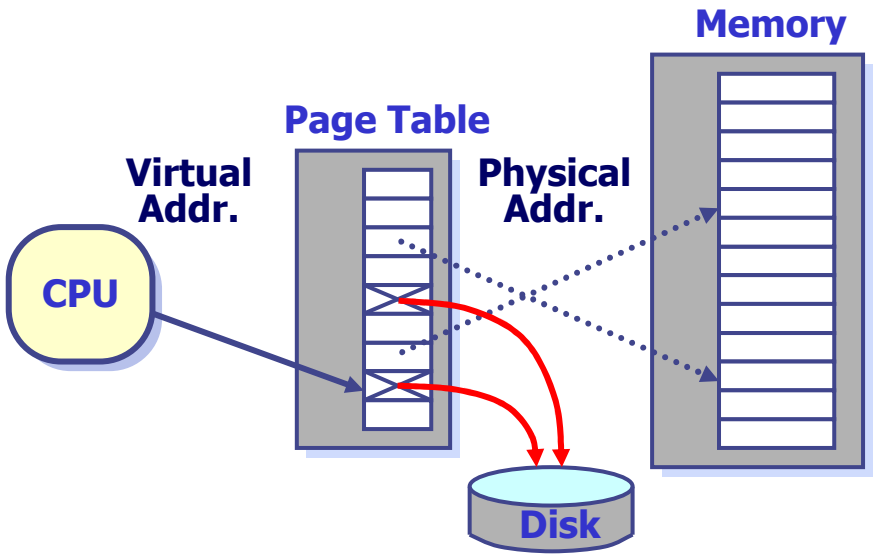
- "**Page Table register**" – registo que contém o endereço de início da *Page Table* do processo ativo
- "**valid bit**" – bit que indica se a página está ou não em memória
- "**access bits**" – atributos de acesso à página

"Page Fault"

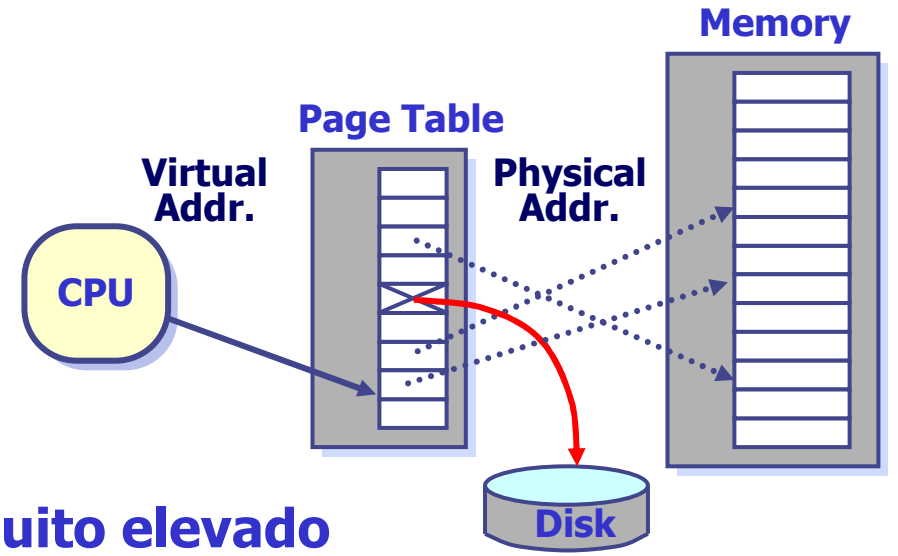

CPU
Main Memory
Secondary Memory
Virtual address
Lookup page table
Page table hit ?
miss
Page fault
Get page from Secondary Memory
yes
Translate to physical address
Update Main Memory, page table
CPU
Return value from Main Memory

"Page Fault"

- Ocorrência de um *Page Fault*



- Após um *Page Fault*



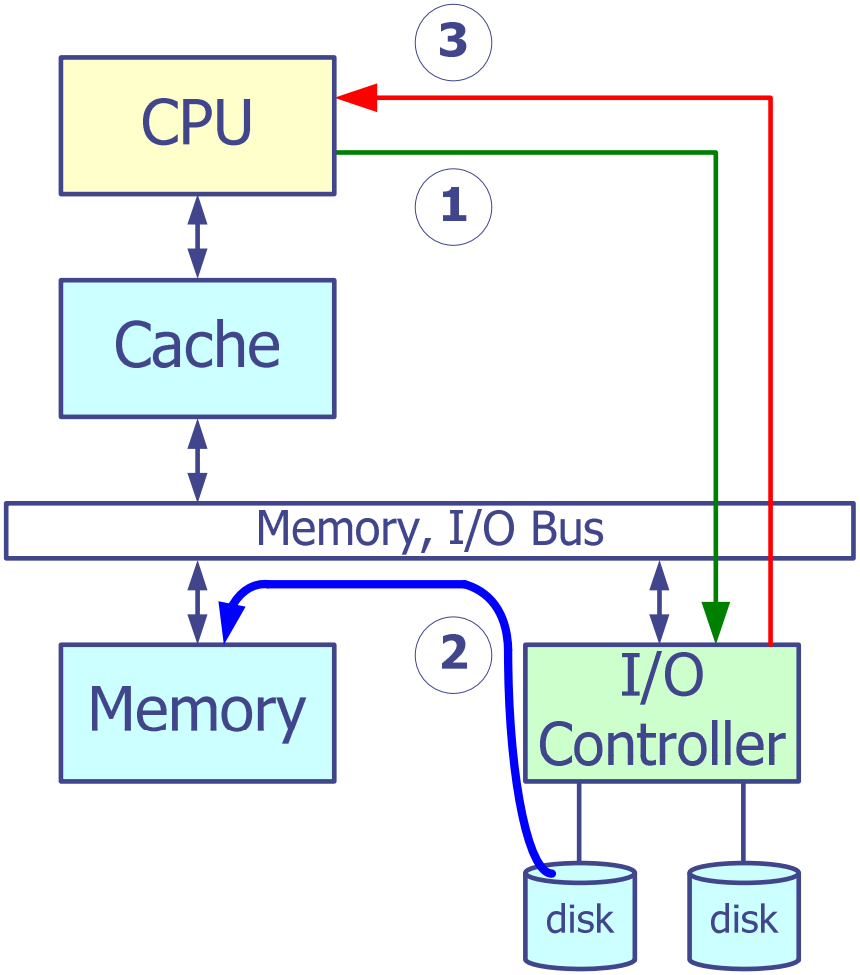
custo de um *Page Fault* é muito elevado

"Page Fault"

- Se a página que o CPU pretende aceder não está em memória ("Valid bit" da *Page Table* com o valor 0), é gerado um *Page Fault*
- Um *Page Fault* tem um custo elevado em termos temporais
- Os *Page Faults* são tratados por software:
  - O gestor de memória (Memory Management Unit - MMU) gera uma exceção que transfere o controlo para o Sistema Operativo
  - O Sistema Operativo decide onde colocar, na memória, a página em falta e desencadeia a respetiva transferência a partir do disco. Se não houver espaço na memória o SO tem que começar por substituir uma página existente e isso pode envolver a cópia dessa página para o disco
  - O processamento completo de um *Page Fault* pode demorar dezenas de milhões de ciclos de relógio
  - O Sistema Operativo suspende o processo corrente e lança outro
  - O processo suspenso é retomado quando o *Page Fault* for resolvido, numa decisão também da responsabilidade do Sistema Operativo

Parte do processamento de um Page Fault

1. SO configura o "I/O controller" para transferir, para memória, a página em falta (bloco):
   - Dimensão do bloco
   - Endereço de início no disco
   - Endereço destino na memória (escolhido pelo SO)
2. A transferência processa-se por DMA
3. O "I/O controller" sinaliza o fim da transferência:
   - Gera uma interrupção
   - O SO retoma a execução do processo suspenso



Política de substituição de páginas na memória

- Quando ocorre um *Page Fault* e a memória física está toda ocupada, é necessário decidir qual a página que tem que sair
- A política normalmente usada é:
  - LRU (Least Recently Used - é substituída a página que está há mais tempo sem ser referenciada)
  - NRU (Not Recently Used) - é uma versão simplificada do LRU
- Implementação do NRU
  - Cada VPN de uma *Page Table* contém um "reference bit"
  - Periodicamente os "reference bits" são colocados a zero pelo Sistema Operativo
  - Quando uma página é acedida (*touched*) o "reference bit" é colocado a 1
  - As páginas candidatas a serem substituídas são as que têm o "reference bit" a 0 (é substituída aleatoriamente uma dessas páginas)

Política de escrita

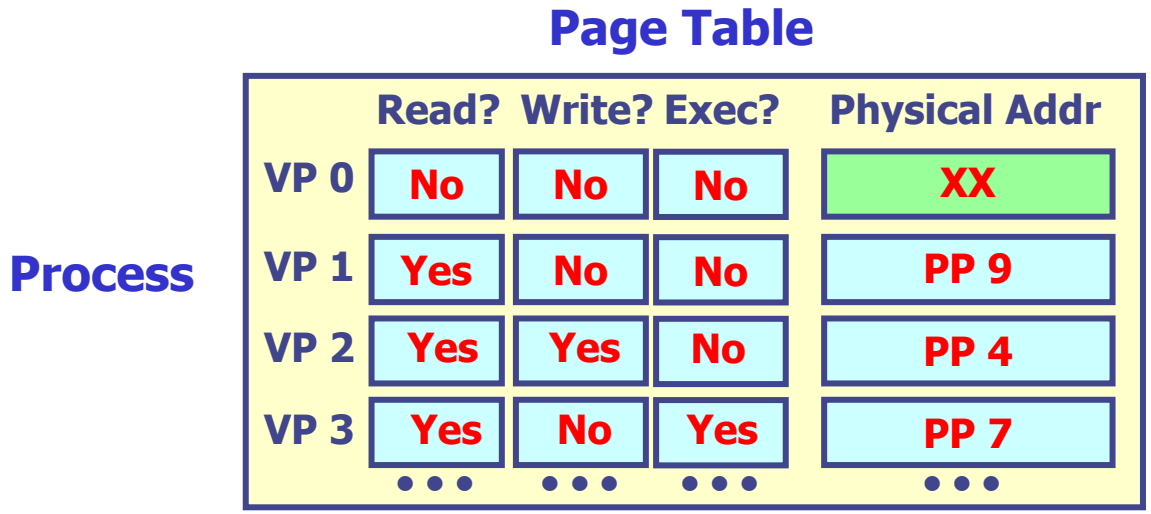
- **Write-back** - a utilização de um esquema do tipo "write-through" seria impraticável uma vez que a escrita de uma página no disco demora um tempo que penalizaria fortemente o desempenho do sistema
  - Escrita em disco página a página
  - Página só é escrita em disco quando necessita de ser retirada da memória física
- **Dirty bit**
  - Se "dirty bit" = 0, a página não necessita de ser escrita em disco aquando da sua substituição (*overwrite*)
  - Se "dirty bit" = 1, a página que vai ser substituída foi alterada. Antes de ser substituída, essa página é copiada para o disco; após essa operação, a nova página é então copiada do disco para a zona da memória que ficou livre
- **Write-allocate**
  - Escrita numa página que não reside em memória física: carrega essa página para a memória e escreve

Implementação de mecanismos de proteção

- Um processo não pode interferir de maneira nenhuma com o funcionamento de outro (seja de forma involuntária ou intencional). Isso é garantido porque:
  - Os espaços de endereçamento de cada processo são independentes
  - Cada processo tem a sua própria *Page Table*, gerida exclusivamente pelo Sistema Operativo
- Dentro do mesmo processo:
  - Diferentes zonas do espaço de endereçamento podem ter diferentes permissões de acesso. Por exemplo, uma zona de instruções não tem permissão de escrita, uma zona de dados não tem permissão de execução, etc.
- Os atributos de acesso a cada página (Read, Read/Write, Execute) são verificados em cada tradução de endereços (para cada VPN)

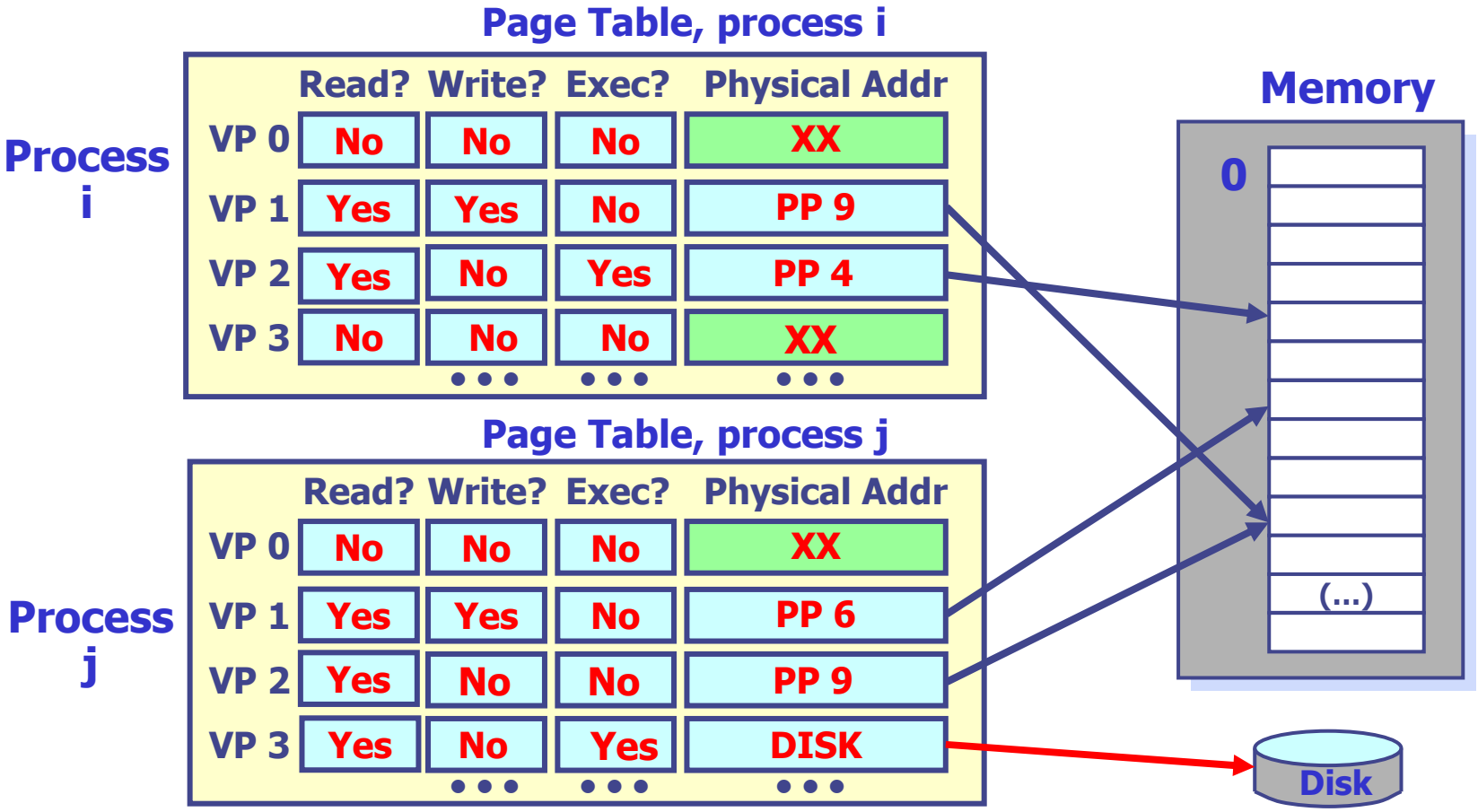
Implementação de mecanismos de proteção

- Cada entrada da *Page Table* contém informação relativa a atributos de acesso do processo respetivo:
  - Read
  - Read / Write
  - Execute

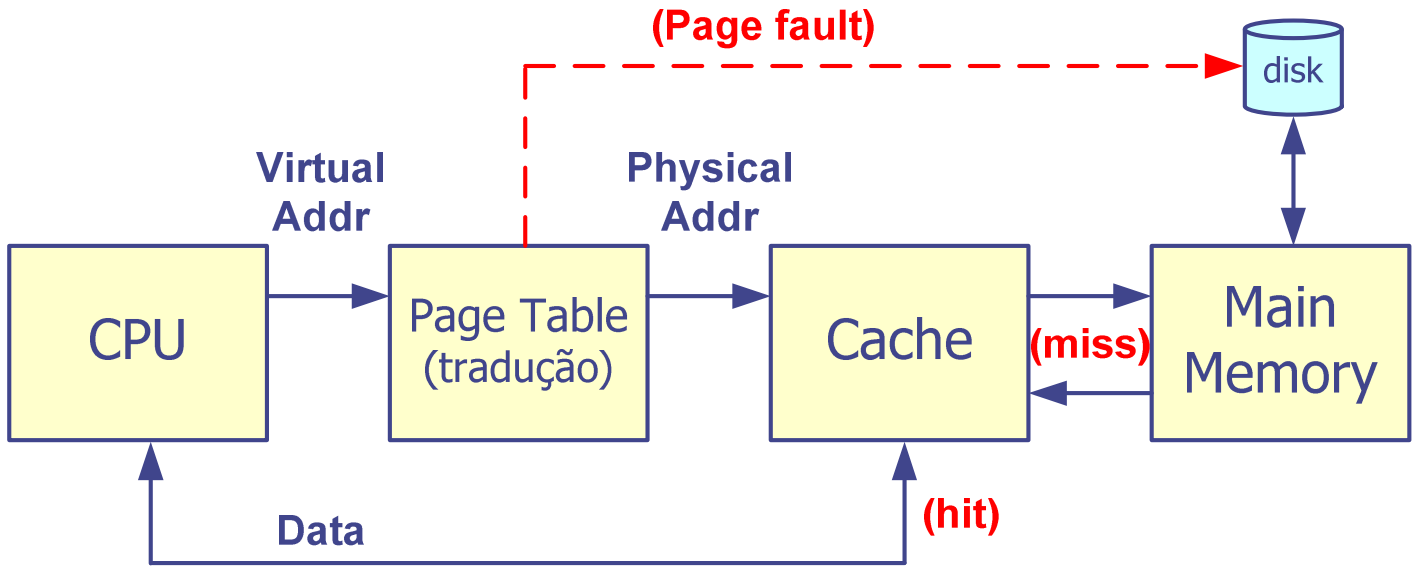


Implementação de mecanismos de proteção

- O hardware gera uma exceção se ocorrer uma tentativa de violação dos atributos de acesso (e.g. no SO Windows: "General protection fault", no SO Linux: "Segmentation fault")



Memória virtual + cache



- A *Page Table* reside na memória principal. Isso significa que cada acesso à memória virtual implica dois acessos à memória física:
  - 1 acesso para indexar a *Page Table* e obter o endereço físico
  - 1 acesso para ler/escrever os dados
- Será esta uma solução viável? Como resolver este problema?

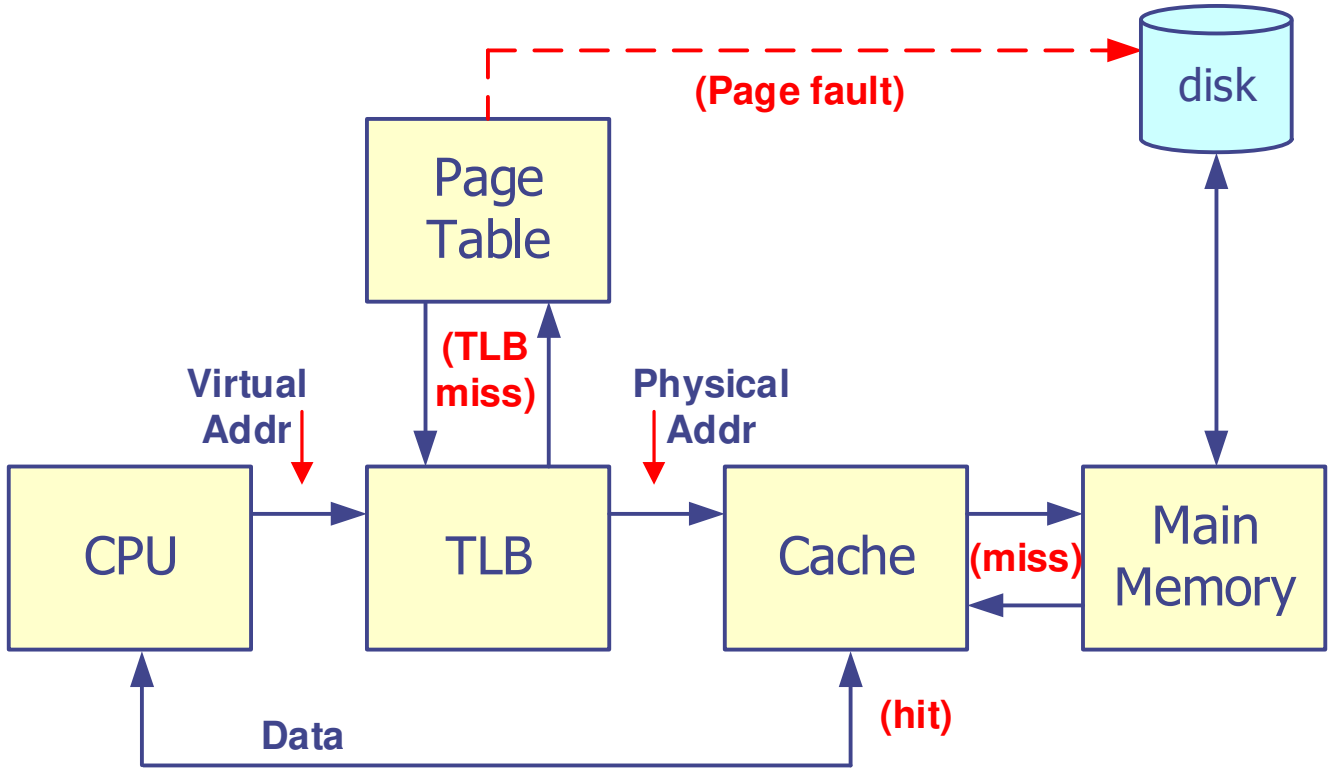
Translation-lookaside buffer (TLB)

- Os acessos à memória virtual contêm localidade espacial e temporal
- Uma forma de resolver o problema é ter uma parte da *Page Table* numa memória rápida, semelhante a uma cache, onde se encontram as entradas da tabela mais recentemente utilizadas
- Esta memória é normalmente designada por "translation-lookaside buffer" - TLB
- A TLB é tipicamente implementada como uma memória associativa com procura paralela
- Páginas residentes em disco não são referenciadas na TLB

Translation-lookaside buffer (TLB)

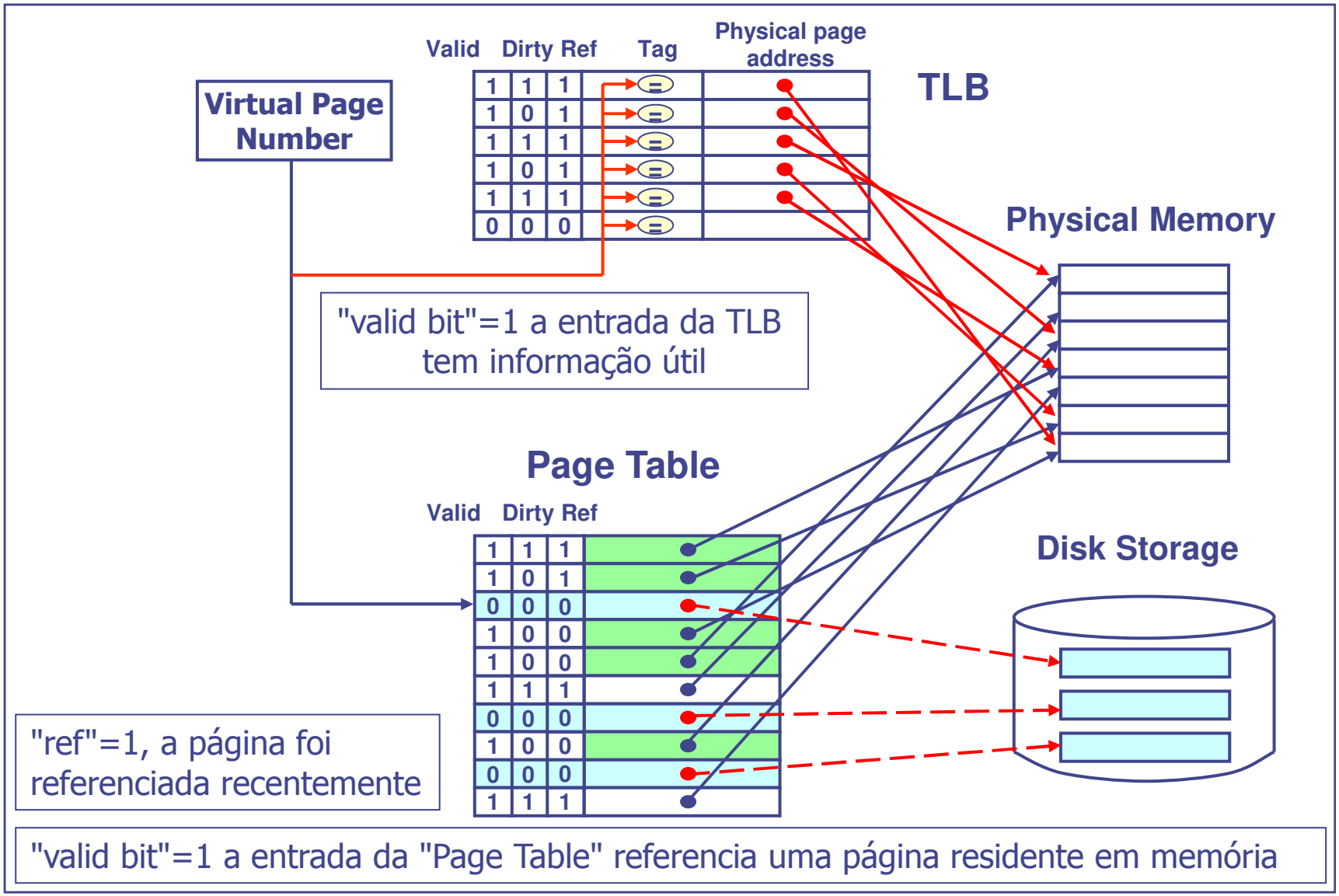
- Em cada acesso, no processo de tradução, verifica-se se o número da página virtual (VPN) está na TLB
- Se VPN está na TLB: "Hit" – o endereço da página física é obtido de imediato da TLB
- Se VPN não está na TLB: "Miss" – a *Page Table* é verificada e pode originar, ou não, um *Page Fault.*
  - Se o acesso à *Page Table* não originar um *Page Fault*, o endereço físico e os atributos da página são copiados da *Page Table* para a TLB (pode envolver a substituição de uma entrada na TLB)
  - Se o acesso à *Page Table* originar um *Page Fault*, é necessário fazer todo o processamento do *Page Fault* que culmina com a atualização da *Page Table*. A seguir o endereço físico e os atributos da página são copiados da *Page Table* para a TLB (pode envolver a substituição de uma entrada na TLB)

TLB + Memória virtual + cache

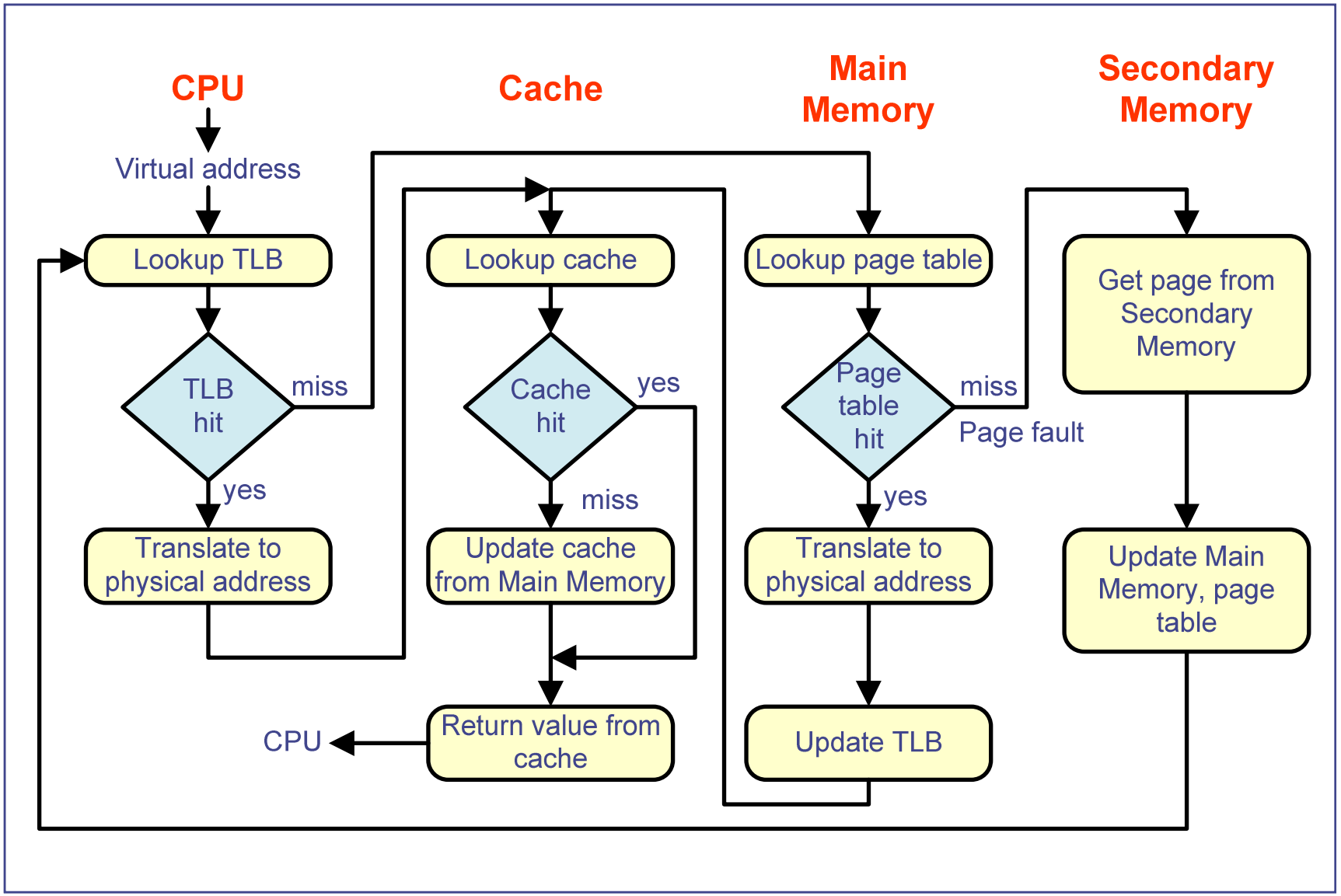


- Se a página virtual está na TLB (hit) a tradução é muito rápida
- Se a página virtual não está na TLB (miss) é necessário consultar a *Page Table*, que reside na memória principal, e atualizar a TLB
- No acesso à *Page Table* pode ocorrer um *Page Fault*

Translation-lookaside buffer (TLB)



Operação da hierarquia de memória

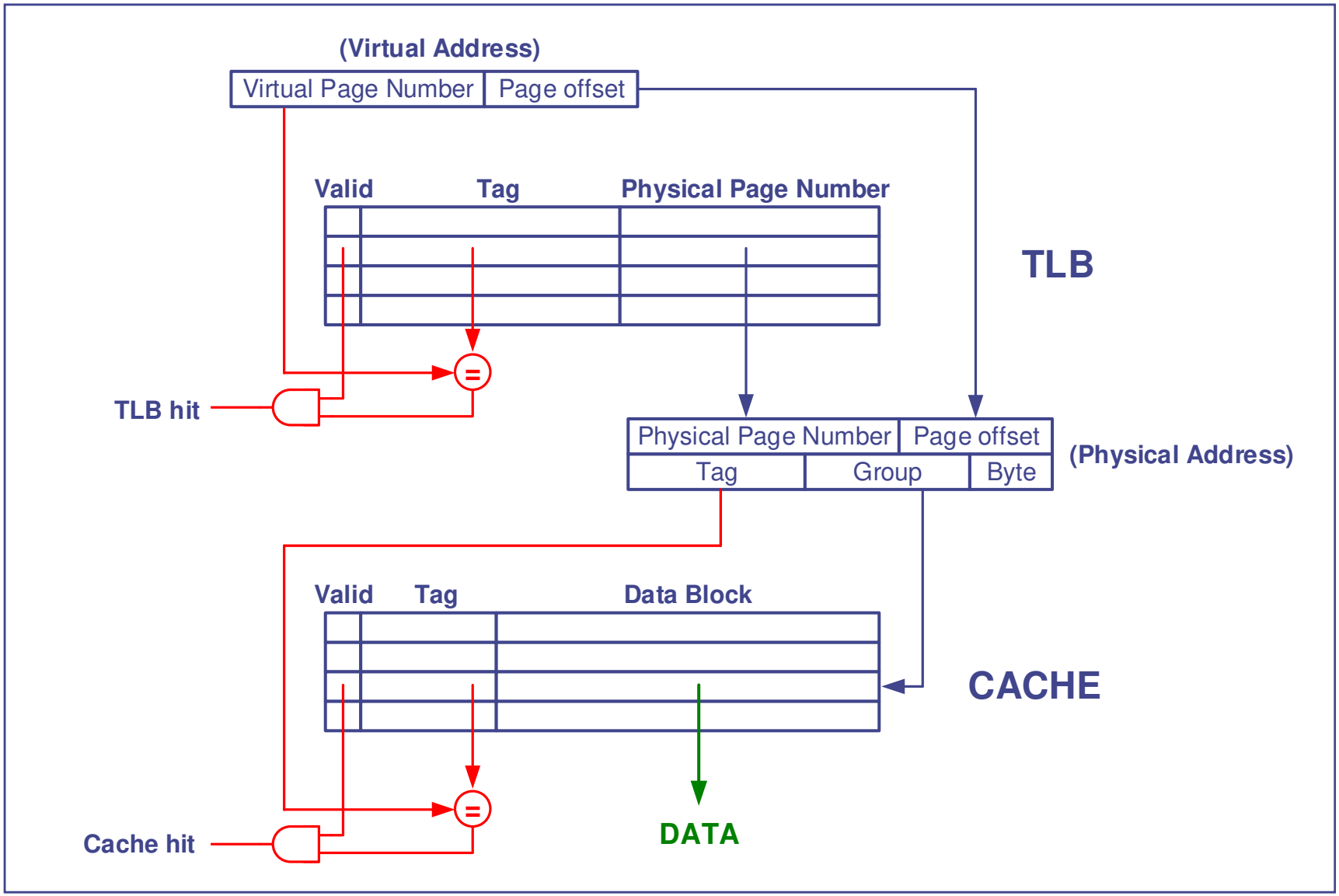


Tradução de endereços com a TLB

- Combinações possíveis dos eventos iniciados por
  - TLB (miss / hit)
  - Page Table (miss / hit, miss ≡ Page Fault)
  - Cache (miss / hit)

TLB	Page Table	Cache	Possível?
miss	miss	miss	Sim! Os dados não estão em memória
miss	miss	hit	Não! A página não está em memória, hit na cache impossível
miss	hit	miss	Sim! A página está memória, dados não disponíveis na cache
miss	hit	hit	Sim! A página está em memória, dados na cache
hit	miss	miss	Não! A página não está referenciada na "Page Table"
hit	miss	hit	Não! A página não está referenciada na "Page Table"
hit	hit	miss	Sim! Página referenciada na TLB, dados não disponíveis na cache
hit	hit	hit	Sim! Página referenciada na TLB, dados disponíveis na cache

Tradução de endereços com a TLB

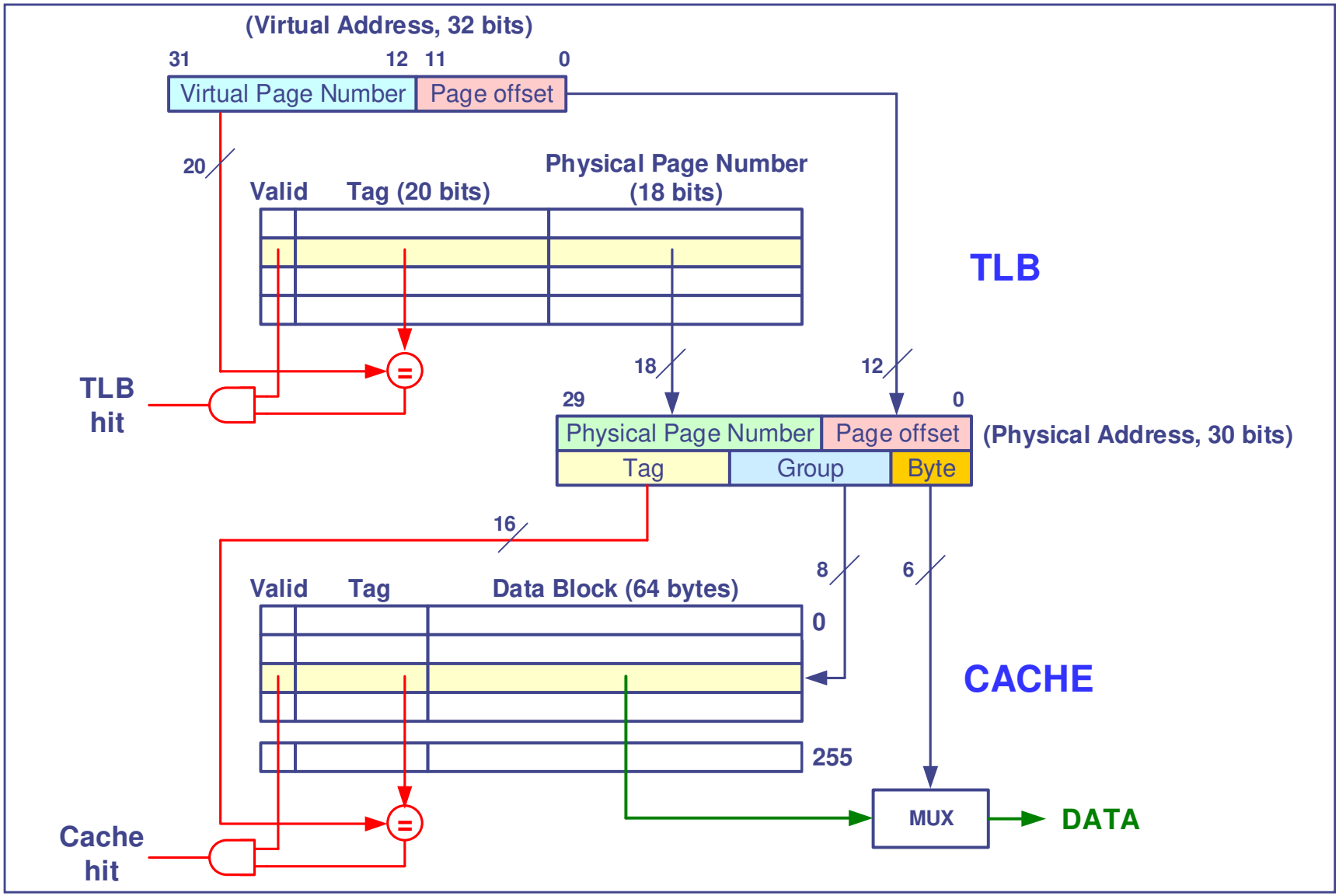


Tradução de endereços com a TLB - Exemplo

S

- Memória virtual:
  - Espaço de endereçamento virtual: 32 bits
  - Espaço de endereçamento físico: 30 bits
  - Dimensão da página: 4 KB (12 bits)
    - VPN – 20 bits
    - PPN – 18 bits
- Cache de 16 kBytes
  - Mapeamento direto
  - Nº de linhas: 256 (8 bits)
  - Dimensão do bloco 64 bytes (6 bits)

Tradução de endereços com a TLB - Exemplo



Em resumo...

- A memória virtual introduz uma indireção entre o endereço virtual gerado pelo CPU e o endereço físico da memória
- Isto permite
  - Mapear memória em disco (memória "ilimitada")
  - Gerir de forma eficiente a memória disponível e evitar a fragmentação
  - Impedir que um processo aceda à zona de memória de outro processo ou que escreva em zonas de memória atribuídas ao processo, mas onde não tem permissão de escrita (segurança)
- Cada acesso à memória envolve a tradução do endereço virtual para um endereço físico
  - A memória é organizada em páginas
  - Cada processo tem uma *Page Table*, localizada em memória, com o mapeamento entre páginas virtuais e páginas físicas e com os respetivos atributos de acesso
  - A tradução de endereços é acelerada através da TLB que contém um subconjunto das entradas da *Page Table*

Exercício

S

- Complete a seguinte tabela, preenchendo as quadrículas em falta e substituindo o ? pelo valor adequado. Utilize as seguintes unidades: K = 2^{10} (Kilo), M = 2^{20} (Mega), G = 2^{30} (Giga), T = 2^{40} (Tera), P = 2^{50} (Peta) ou E = 2^{60} (Exa).

Virtual address size (n)	# Virtual addresses (N)	Maior endereço virtual (hexadecimal)
8	$2^8 = 256$	0xFF
	$2^? = 64$ K	
		0xFFFFFFFF
	$2^? = 256$ T	
64		

Exercício

S

- Determine o número de entradas da *Page Table* (PTE) para as seguintes combinações de número de bits do espaço de endereçamento virtual (n) e dimensão da página (P):

n	P	# PTEs
16	4 KB	
16	8 KB	
32	4 KB	
32	8 KB	
48	4 KB	

Exercício

S

- Suponha um espaço de endereçamento virtual de 32 bits e um espaço de endereçamento físico de 24 bits:
  - determine o número de bits dos campos: VPN (virtual page number), VPO (virtual page offset), PPN (physical page number), PPO (physical page offset) para as dimensões de página P:

P	VPN	VPO	PPN	PPO	# virtual pages	# physical pages
1 KB						
2 KB						
4 KB						
8 KB						

  - para cada dimensão de página, determine o número de páginas virtuais e físicas

Exercício

S

- Considere um sistema de memória virtual com um espaço de endereçamento virtual de 26 bits, páginas de 512 bytes, em que cada entrada da "Page Table" está alinhada em endereços múltiplos de 2, tem 16 bits de dimensão, e está organizada do seguinte modo:

  Valid bit, **Dirty** bit, **Read** flag, **Write** flag, **PPN**

  - em quantas páginas está organizado o espaço de endereçamento virtual? Quantas entradas tem a "Page Table"?
  - qual a dimensão do espaço de endereçamento físico?
  - em quantas páginas está organizado o espaço de endereçamento físico?
- Suponha que o "Page Table register" do processo em execução tem o valor 0x1A0 e que no endereço 0x252 (e 0x253) está armazenado o valor 0xA26C
  - quais os atributos da página física referenciada por essa entrada da tabela? Onde reside a página física?
  - qual é o VPN representado nessa entrada da "Page Table"?
  - qual o endereço inicial e final da página física?
  - qual a gama de endereços virtuais que indexa esta entrada da "Page Table"?