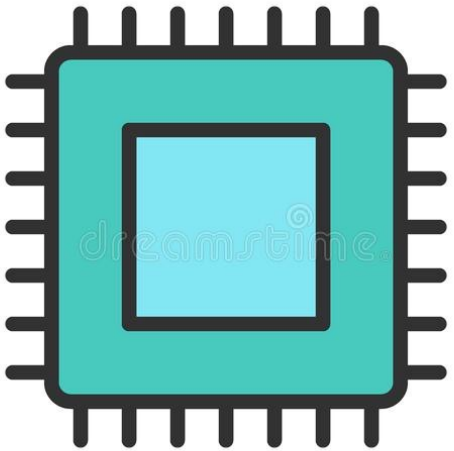
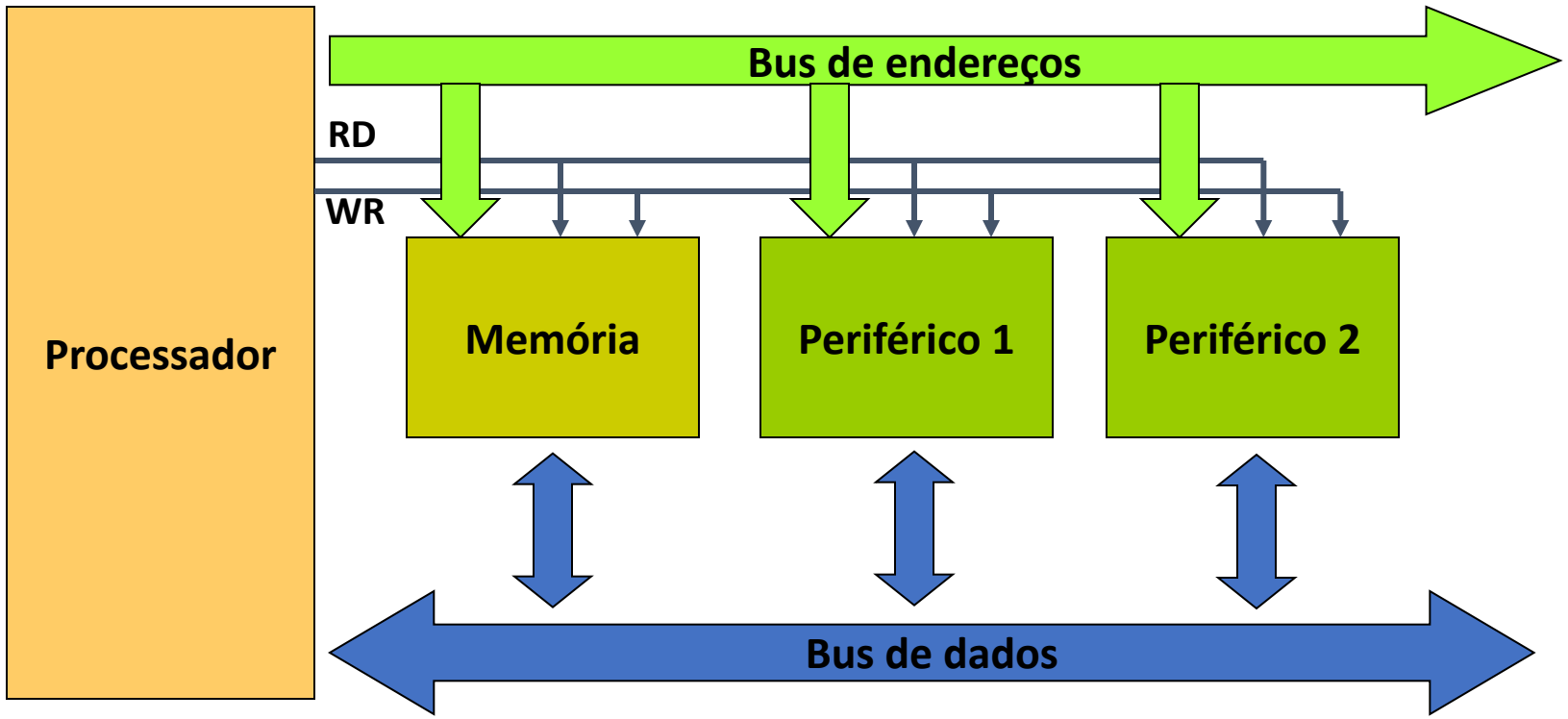
Introdução à arquitetura de sistema



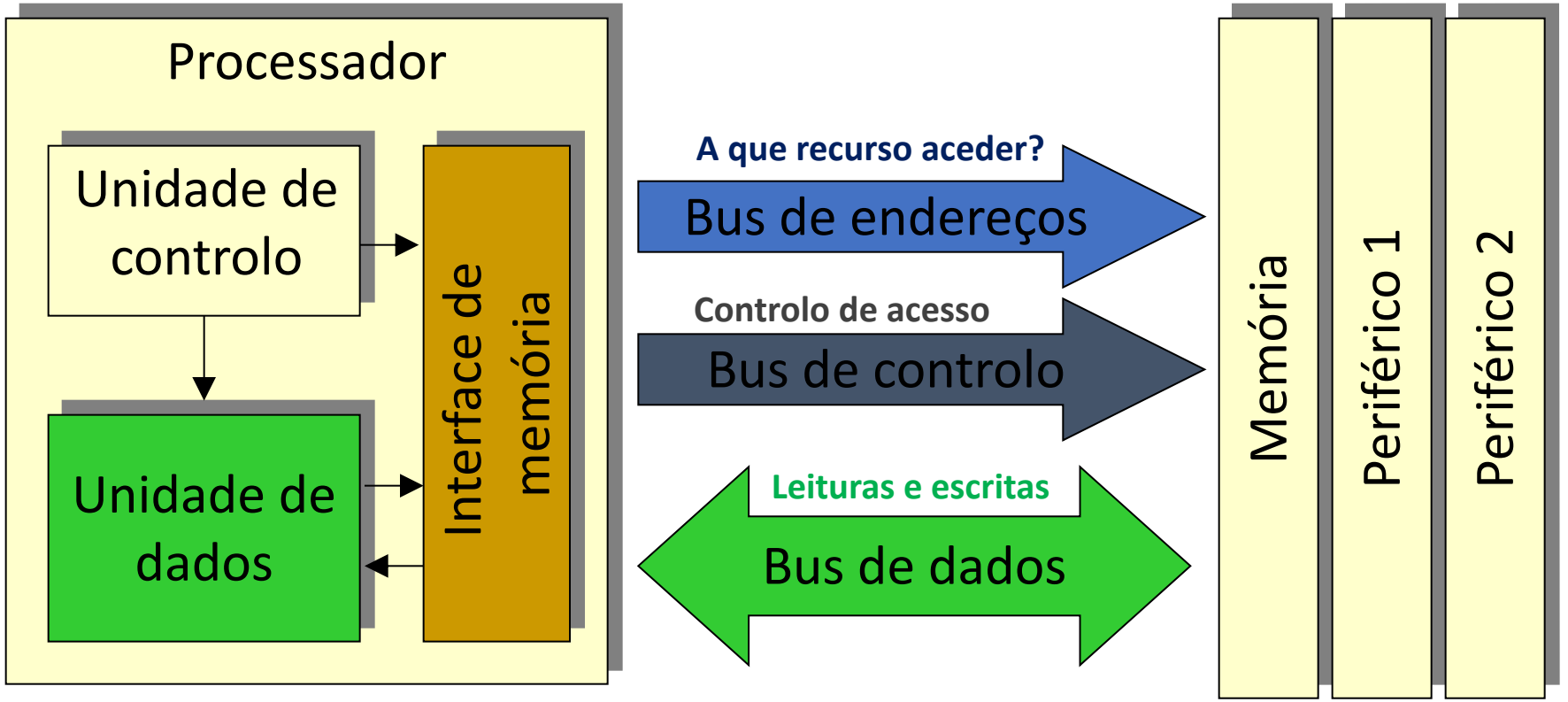
TÉCNICO LISBOA

Processador, memória e periféricos



Barramento (bus): meio físico de **interligação** dos vários dispositivos

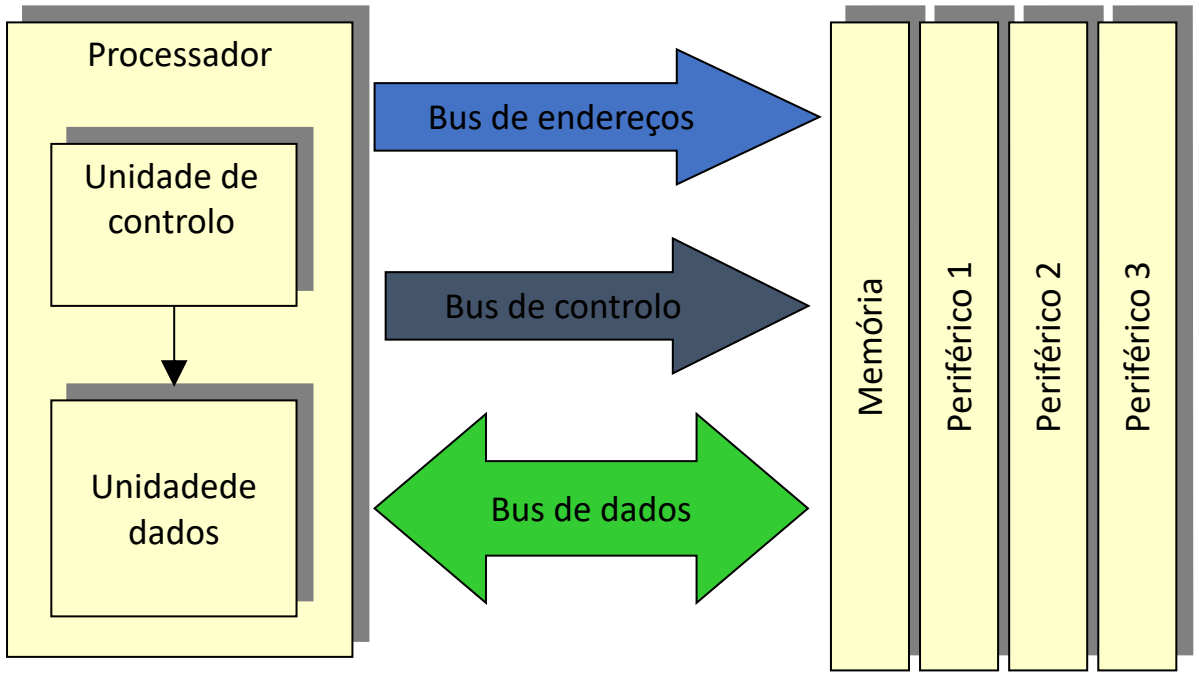
Acesso ao exterior do processador



- O processador especifica um endereço.
- Não sabe se é memória ou periférico.

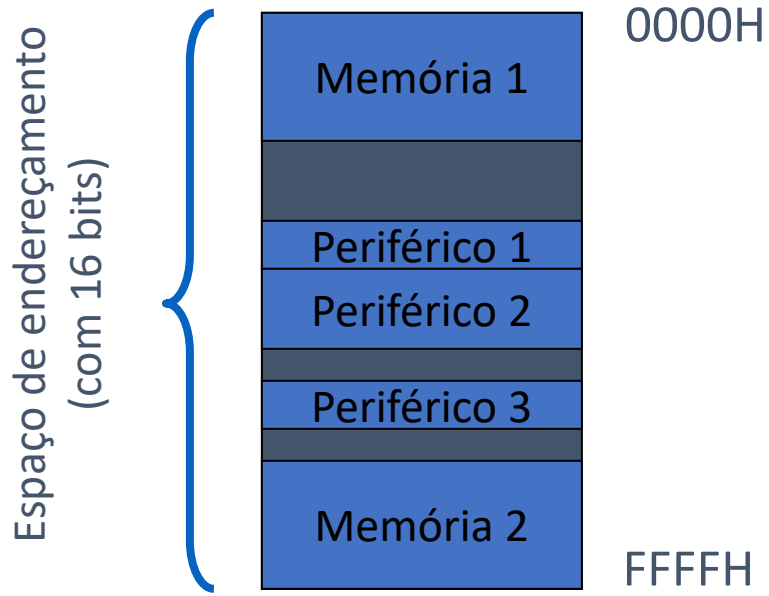
Descodificação de endereços

Espaço de endereçamento

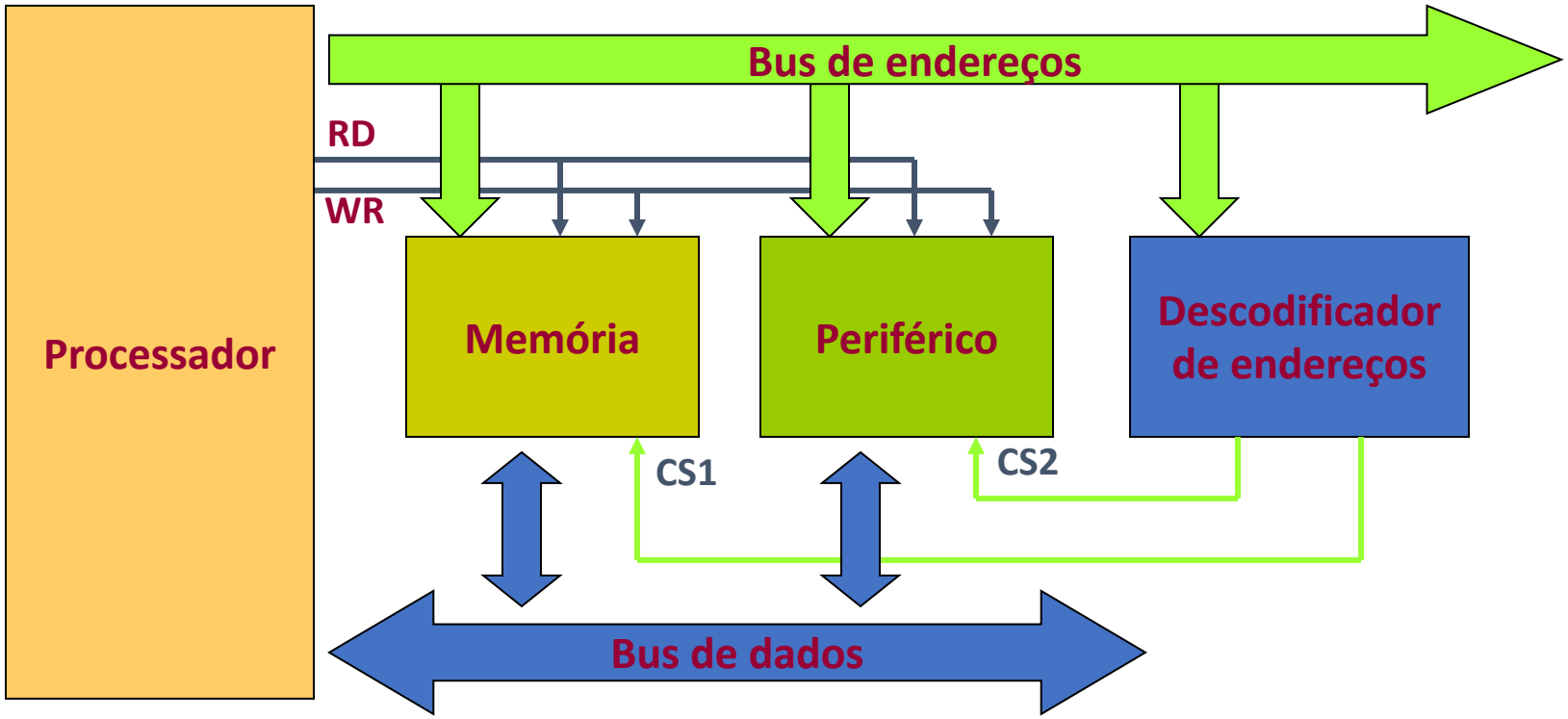


- Memória e periféricos **coexistem** no mesmo espaço de endereços.
- Como distingui-los?

Mapa de endereços



Descodificação de endereços



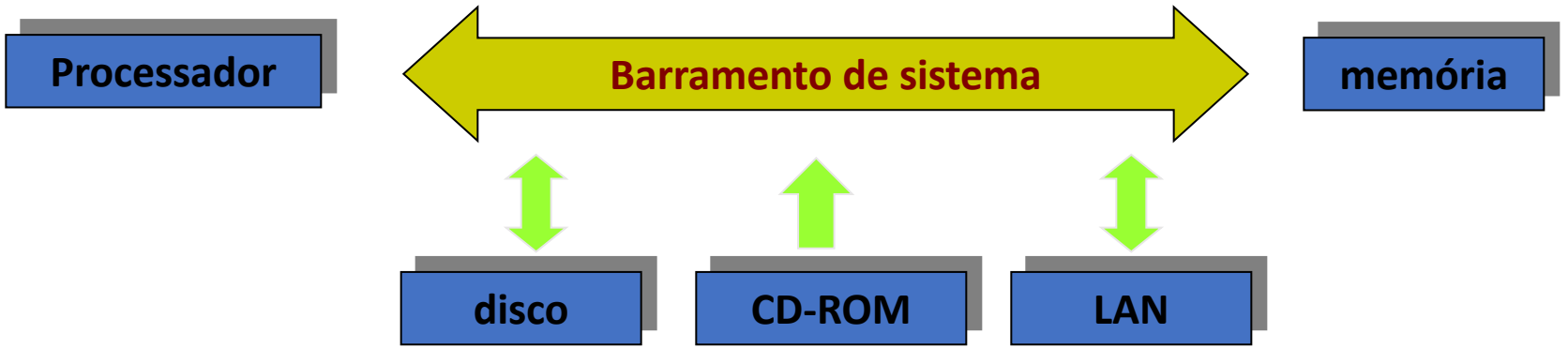
- Cada dispositivo liga aos bits de **menor peso** do bus de endereços
- O **descodificador de endereços** liga aos bits de **maior peso** do bus de endereços e gera os *chip select* (CS)
  - Quando está **ativo** indica que o dispositivo está a ser acedido

refletir.com

- Supondo um processador de 8 bits com este mapa de endereços:
  - Quantos bits deve ter no mínimo o bus de endereços? **24**
  - Qual o espaço de endereçamento deste mapa de endereços? **16 MiB**
  - Qual a capacidade da RAM? **2 MiB**
  - Qual a capacidade da ROM? **8 MiB**
  - Qual o espaço reservado para periféricos? **4 KiB**
  - Qual o espaço livre? **6 MiB – 4 KiB**
  - Quantos bits de endereço devem ligar a cada módulo de RAM? **20**
  - E à ROM? **23**

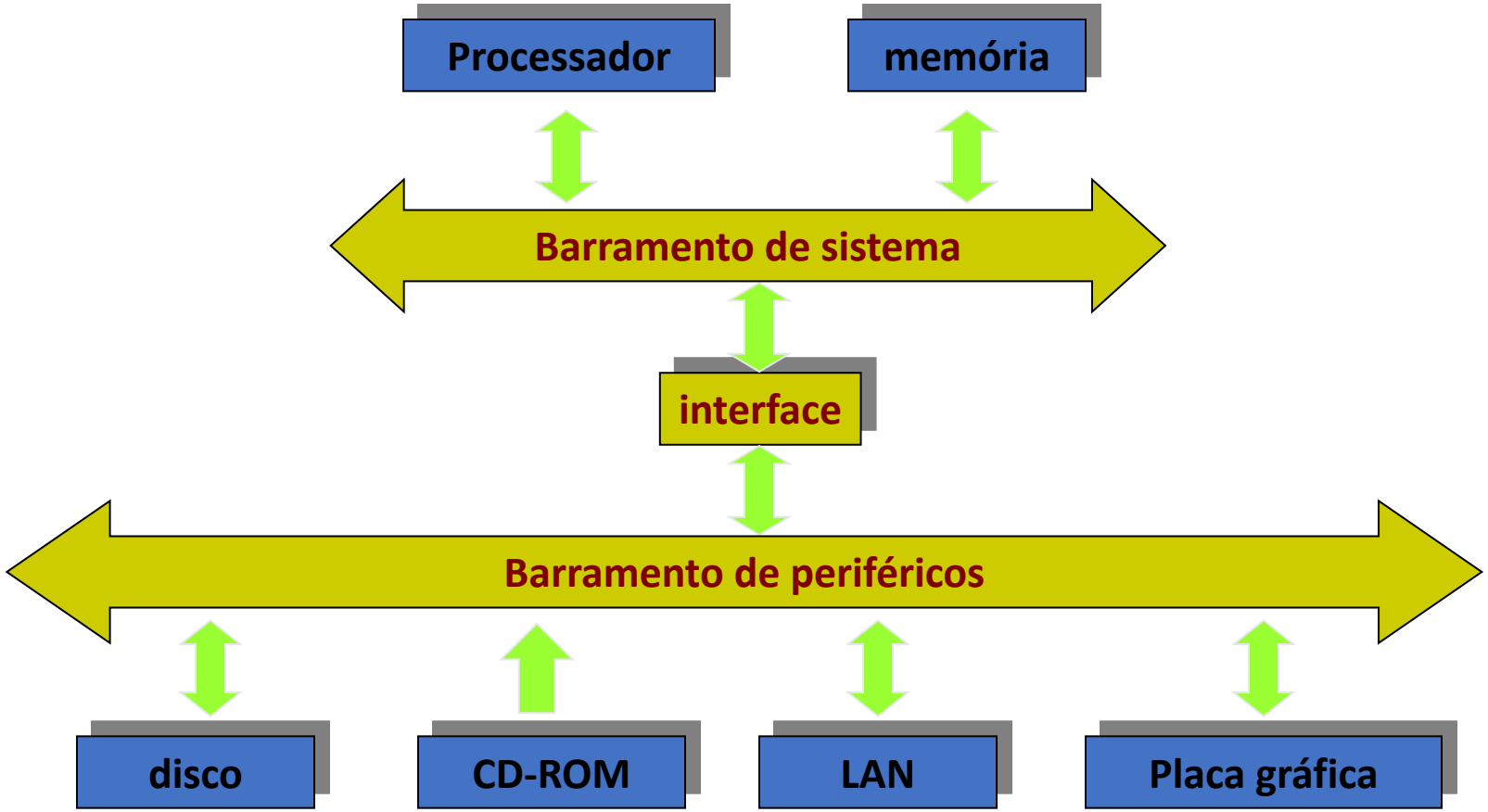
Arquitetura do sistema de periféricos

Problema



- Com o desempenho dos processadores (frequência do seu relógio) aumentou muito rapidamente, tornou-se **impraticável** a ligação direta a todos os periféricos (que são regra geral **muitos mais lentos**)

Solução: barramentos hierárquicos



Modos de transferência de dados

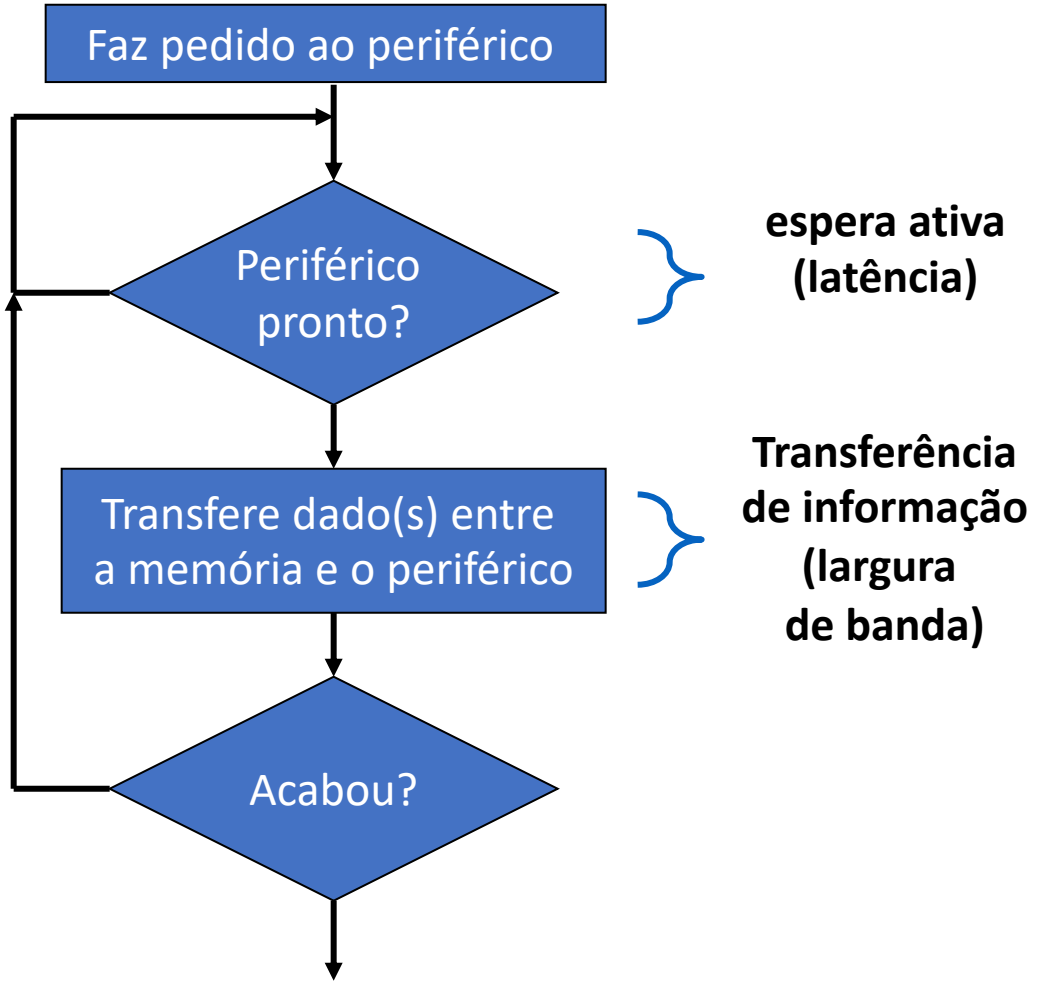
- Modos de transferência de informação entre o processador/memória e os periféricos:
  - Sob controlo do programa (***polling***)
  - Por **interrupção**
  - Por acesso direto à memória (**DMA** – Direct Memory Access)
  - Com um **co-processador** de entradas/saídas

- Num extremo (*polling*), o **processador** trata de tudo.
- No outro, o processador limita-se a programar o **co-processador**.
- Dado que as entradas/saídas são lentas, a ideia é **reduzir o tempo** que o processador gasta à espera dos periféricos (libertando-o para outras tarefas).

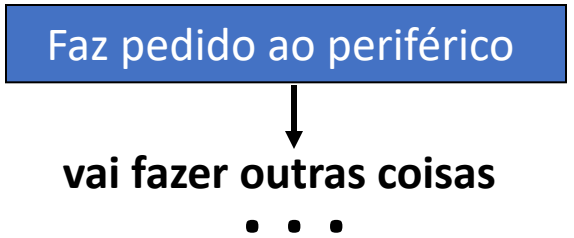
Polling

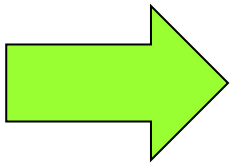
- O programa controla tudo.
- O processador faz espera ativa contínua (senão pode perder dados) sobre periféricos lentos
- A transferência é feita por software.

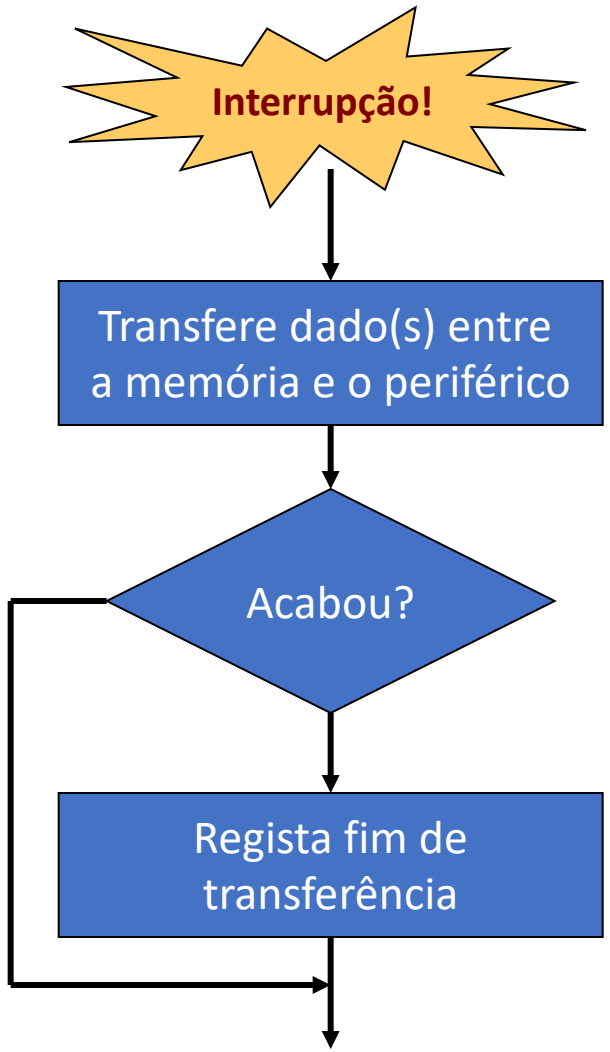
- Problema: muito **ineficiente**; o processador poderia estar a executar outras tarefas em vez de estar à espera de dispositivos lentos



Transferência por interrupção







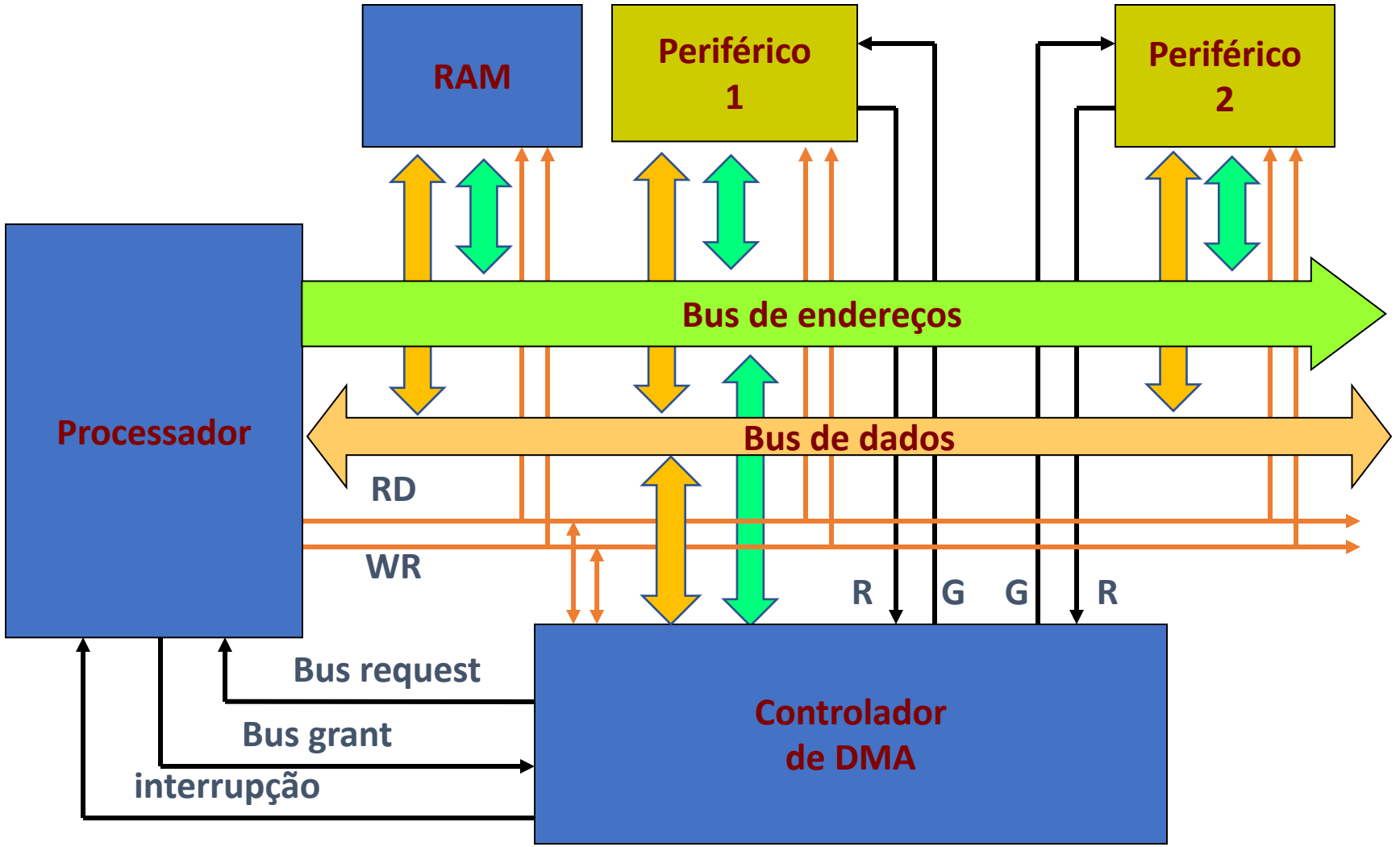
- A espera já não é ativa.
- O processador só é “incomodado” quando há coisas para fazer.

- Vantagem: mais **eficiente**
- Problema: a transferência de dados (por software) **pode ser lenta**.

DMA (Direct Memory Access)

- A transferência de informação entre o processador/memória e os periféricos é **feita em hardware por um controlador especializado**.
- O processador só tem de **programar o controlador de DMA**, escrevendo em portos próprios do controlador (que em si também é um periférico):
  - Endereço de origem
  - Endereço de destino
  - Número de palavras a transferir
  - Qual o modo de DMA
- Durante a transferência, os endereços de origem e destino são incrementados **automaticamente**

Controlador de DMA



Lidar com vários periféricos

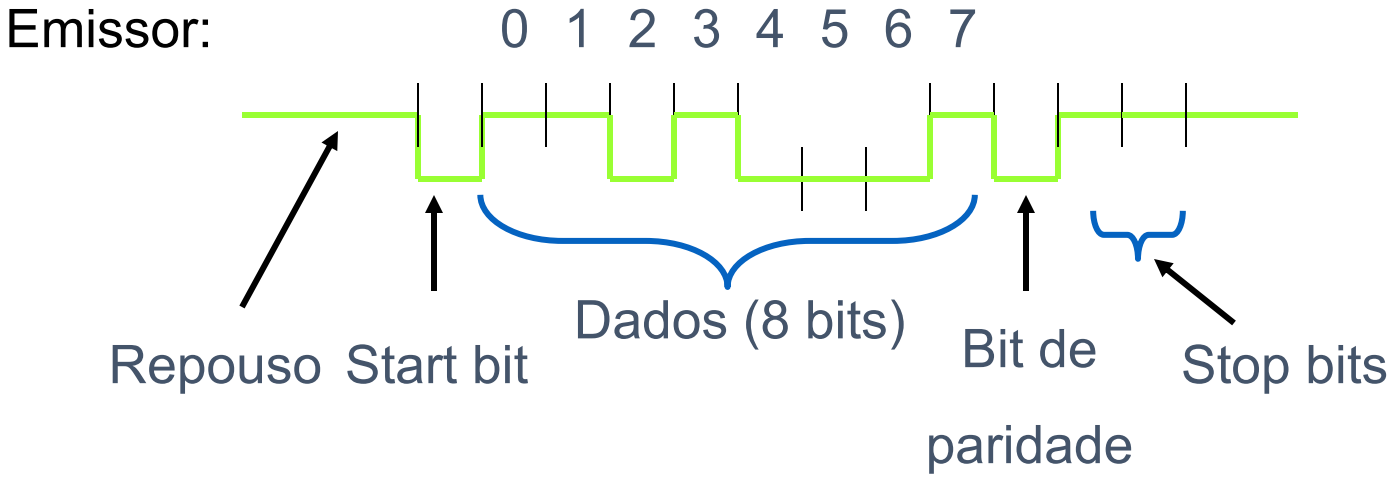
- Um computador tem normalmente vários periféricos e pode **misturar os vários modos** de transferência de dados.
- Deve-se ter em atenção:
  - A transferência sob controlo do programa (***polling***) deve ser reservada para **periféricos lentos**, sem temporizações críticas e com protocolos que possam ser interrompidos;
  - A transferência por interrupções é mais eficiente, mas **pesada** para transferência de **grandes quantidades** de informação (a transferência em si é feita por software);
  - A transferência por DMA (ou com co-processador) é a **mais eficiente**, mas o processador pode não conseguir atender interrupções durante uma transferência.

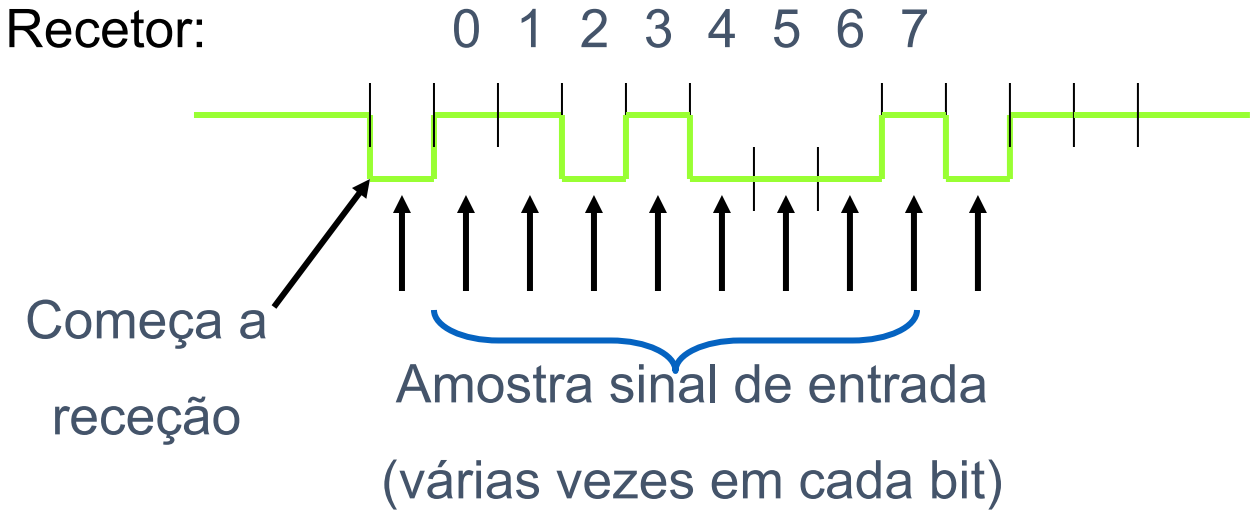
Periféricos de comunicação: barramentos

Exemplo: barramentos série assíncronos

- A comunicação é **orientada ao byte**, serializado
- Barramento está normalmente em repouso (1)
- Quando o **emissor decide transmitir**:
  - coloca a linha a 0 durante um bit (*start bit*)
  - envia os 8 bits do byte em sequência
  - envia um bit de paridade (para deteção de erros)
  - envia de 1 a 2 *stop bits* a 1 (para sincronização)
- A cadência de transmissão dos bits (***baud-rate***) tem de ser **aproximadamente** a mesma em todos os dispositivos no barramento (não tem de ser exatamente igual)
- O assincronismo deriva do tempo arbitrário entre bytes. Usa-se em aplicações de **baixo ritmo de transmissão** (sistemas de controlo, por exemplo)

Comunicação série assíncrona





É possível haver ligeiros desvios

Comunicação série assíncrona

- Há diversas ***baud-rates* normalizadas**:
  - 110 bit/s
  - 75 bit/s e seus múltiplos: 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
  - 14400 bit/s e seus múltiplos: 28800, 43200, 57600
- Outros parâmetros:
  - Paridade: par, ímpar ou nenhuma
  - Stop bits: 1, 1.5 ou 2
- Existem já chips que implementam este protocolo:
  - UART (*Universal Asynchronous Receiver and Transmitter*)
  - USART (suporta também o protocolo síncrono)

refletir.com

1. Suponha que estabeleceu uma ligação entre dois computadores usando uma linha série com um protocolo assíncrono, usando um bit de paridade, dois stop bits e um ritmo de transmissão de 19200 baud.
   a) Quanto tempo demora, no mínimo, a transmitir 10.000 bytes?
   b) Supondo que o ritmo de transmissão foi o máximo possível, qual a percentagem do tempo gasto a transmitir os bits de dados, face ao tempo total de transmissão?
   c) Imagine que, devido ao processamento local da informação, o emissor não consegue transmitir os dados ao ritmo máximo possível (a não ser durante os períodos de envio do byte). Supondo que não altera nada no recetor, como é que o recetor lida com esta situação?
   d) Suponha agora que o problema está no recetor, isto é, que tem de processar os bytes recebidos e que não consegue recebê-los e processá-los ao ritmo máximo possível. Indique possíveis soluções.

Desempenho

Desempenho de processadores

- A melhor forma de medir o desempenho de um processador (relativamente a outros) é **medir o tempo de execução** de um programa.
- Equação básica do **desempenho**:

$$T = \frac{N * D}{F}$$

T – **tempo de duração** do programa

N – **número de instruções** no programa

D – Duração média (em **ciclos de relógio**) **de cada instrução** (CPI, *Clock Cycles per Instruction*)

F – Frequência do relógio (**ciclos/segundo**)

Os limites do desempenho

- N, D e F **não são independentes**:
  - Para reduzir N, cada instrução tem de fazer mais, o que pode aumentar D e/ou reduzir F;
  - Para reduzir D, as instruções têm de ser mais simples, o que obriga a ter mais instruções para fazer o mesmo;
  - Para aumentar F (sem melhorar a tecnologia), só com uma arquitetura mais simples, o que obriga a aumentar N.
- Um processador de F = 2 GHz pode ser mais rápido do que outro de F = 2.5 GHz, se tiver um menor valor de D ou de N.
- Os processadores têm evoluído por:
  - melhor **tecnologia** (F mais elevado);
  - melhor **arquitetura** (menor valor de D);
  - melhores **compiladores** (menor valor de N).

$$T = \frac{N * D}{F}$$

Avaliação do desempenho

- Problema típico: **comparar** o desempenho de dois ou mais computadores.
  - Comparar os fatores individuais não faz sentido (porque são dependentes uns dos outros)
- Métrica simples: MIPS (Mega Instructions Per Second)
  - Ou seja, o fator F/D não chega. O valor de N pode ser diferente.
- Fabricantes divulgam normalmente o valor máximo do MIPS e não médio (porque depende do peso relativo da ocorrência das várias instruções)

$$T = \frac{N * D}{F}$$

Mas… um computador **não é apenas** o processador!

O computador como um todo: *benchmarks*

- Um *Ferrari* numa estrada urbana à hora de ponta não anda mais depressa do que o meu carro (pois, eu não tenho um *Ferrari* ☺)
  - Ao comparar os dois carros, não interessa apenas medir a rotação máxima ou a potência do motor. Tem de se analisar o **resultado global** da sua utilização.

- Em computadores: em vez de MIPS, usam-se ***benchmarks***, que são programas que exercitam os vários aspetos de um computador (processador, memória e periféricos).
- Valor do *benchmark*: número de vezes/segundo que o *benchmark* executa.

As limitações dos periféricos

- Taxas de transferência típicas:
  - Teclado (depende de quem está a teclar...) – 10 bytes/seg
  - LAN, 100 Mbits/seg – 12.5 Mbytes/seg
  - LAN, 100 Gbits/seg – 12.5 Gbytes/seg
  - Disco – 100 Mbytes/seg a 300 Mbytes/seg
  - Bus de dados a 200 MHz (64 bits) – 1,6 Gbytes/seg
  - Registos internos a 2 GHz (64 bits) – 16 Gbytes/seg
- Um processador com o dobro do relógio não corre necessariamente programas em metade do tempo!

  Tempo total = tempo execução em memória + tempo periféricos
- Se o tempo gasto à espera dos periféricos for de 50%, duplicar a velocidade do processador apenas reduz o tempo total em 25%

A lei de Amdahl

- Assumindo que se melhora um fator que afeta **apenas parte** do tempo de execução:

$$Tempo\_melhorado = \frac{Tempo\_parte\_afectada}{n^{o}\_vezes\_mais\_rápido} + Tempo\_parte\_não\_afectada$$

- Mesmo que se melhore um dos fatores (e.g., velocidade do processador), os restantes podem **limitar severamente** a melhoria global.
- Deve-se procurar **otimizar** os fatores usados **mais frequentemente**, isto é, com mais peso no programa.

Medidas de desempenho do I/O

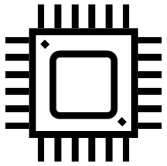
- Há 2 grandezas fundamentais:
  - **Latência**: tempo **até se iniciar** a transferência (relacionado com o tempo de procura de informação, tempo de inicialização do canal de transferência, etc.).
  - **Largura de banda**: máxima **quantidade de informação** transferida por unidade de tempo.

- Cada acesso a um periférico inclui um período de latência e outro de transferência
- A velocidade de transferência efetiva (média) depende do peso relativo da latência.

Comunicação via rede

- Fatores fundamentais na comunicação:
  - Tempo de **acesso** à informação (disco, por exemplo)
  - Tempo de **processamento local** (normalmente desprezável, mas pode ser importante se os dados tiverem muito processamento – compressão, por exemplo)
  - Tempo de **comunicação** (tal como o acesso aos discos, inclui **latência** e **tempo** de transmissão)
- O tempo total de comunicação é o somatório destes tempos parciais.
- Normalmente, o fator limitativo é o disco, mas uma rede lenta pode estrangular a comunicação

“Latency numbers every programmer should know”

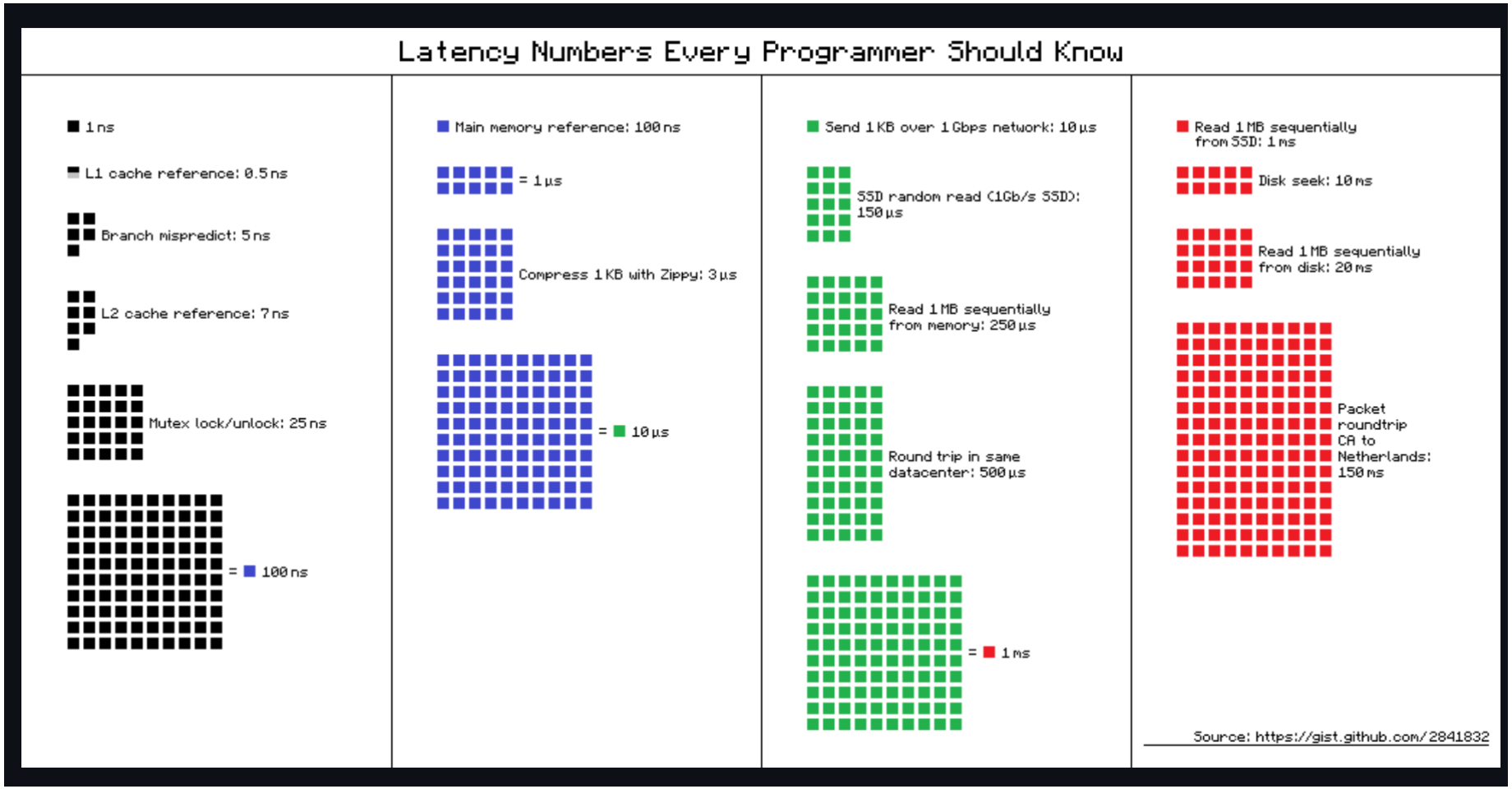


```
L1 cache reference ..........................  0.5 ns
Branch mispredict ............................    5 ns
L2 cache reference ...........................    7 ns
Mutex lock/unlock ...........................   25 ns
Main memory reference ......................  100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

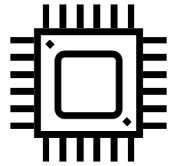
TÉCNICO LISBOA

https://gist.github.com/hellerbarde/2843375

“Latency numbers every programmer should know”, visualmente



“Latency numbers every programmer should know”, multiplicado por 10^9



Minute:

L1 cache reference	0.5 s	One heart beat (0.5 s)
Branch mispredict	5 s	Yawn
L2 cache reference	7 s	Long yawn
Mutex lock/unlock	25 s	Making a coffee

Hour:

Main memory reference	100 s	Brushing your teeth
Compress 1K bytes with Zippy	50 min	One episode of a TV show (including ad breaks)

Day:

Send 2K bytes over 1 Gbps network	5.5 hr	From lunch to end of work day

Week

SSD random read	1.7 days	A normal weekend
Read 1 MB sequentially from memory	2.9 days	A long weekend
Round trip within same datacenter	5.8 days	A medium vacation
Read 1 MB sequentially from SSD	11.6 days	Waiting for almost 2 weeks for a delivery

Year

Disk seek	16.5 weeks	A semester in university
Read 1 MB sequentially from disk	7.8 months	Almost producing a new human being
The above 2 together	1 year	

Decade

Send packet CA->Netherlands->CA	4.8 years	Average time it takes to complete a bachelor's degree

refletir.com

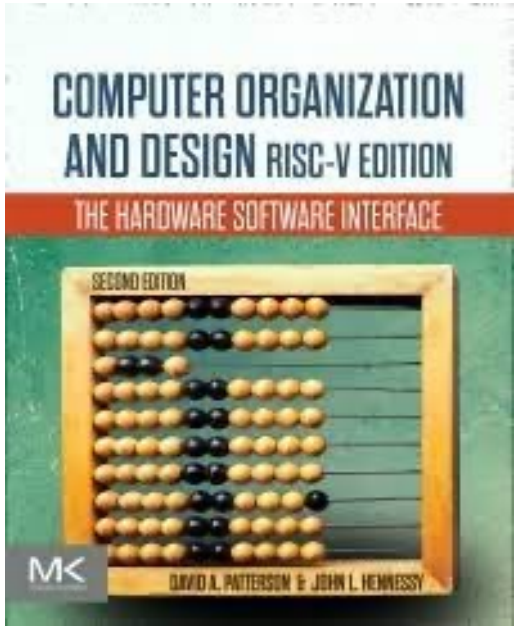
Suponha que um computador com **2 GHz de frequência** de relógio corre um programa que executa **1000 instruções**, das quais cerca de **20% demora 1 ciclo** de relógio a executar, **30% demoram 2 ciclos** e **50% acedem à memória**, gastando **2 ciclos** de relógio mais o **tempo de acesso** à memória (**4 nanossegundos**).

1. Estime qual o **tempo total** de execução do programa.
2. Imagine que se arranjou uma **memória duas vezes mais rápida**, com tempo de acesso de **2 nanossegundos**. Estime **qual a melhoria** no tempo de execução do programa.

Bibliografia

Recomendada

- **[Delgado&Ribeiro_2014]**
  - **Secções 6.1.1 a 6.1.3.2**
  - **Secções 6.1.4 a 6.1.5.1**
  - **Secções 6.3.1, 6.3.2, 6.3.4.3**
  - **Secções 6.4 e 6.6**



Secundária/adicional

- [Patterson&Hennessy_2021]
  - DMA e polling (OL6.9; conteúdo online), desempenho (secção 1.6)