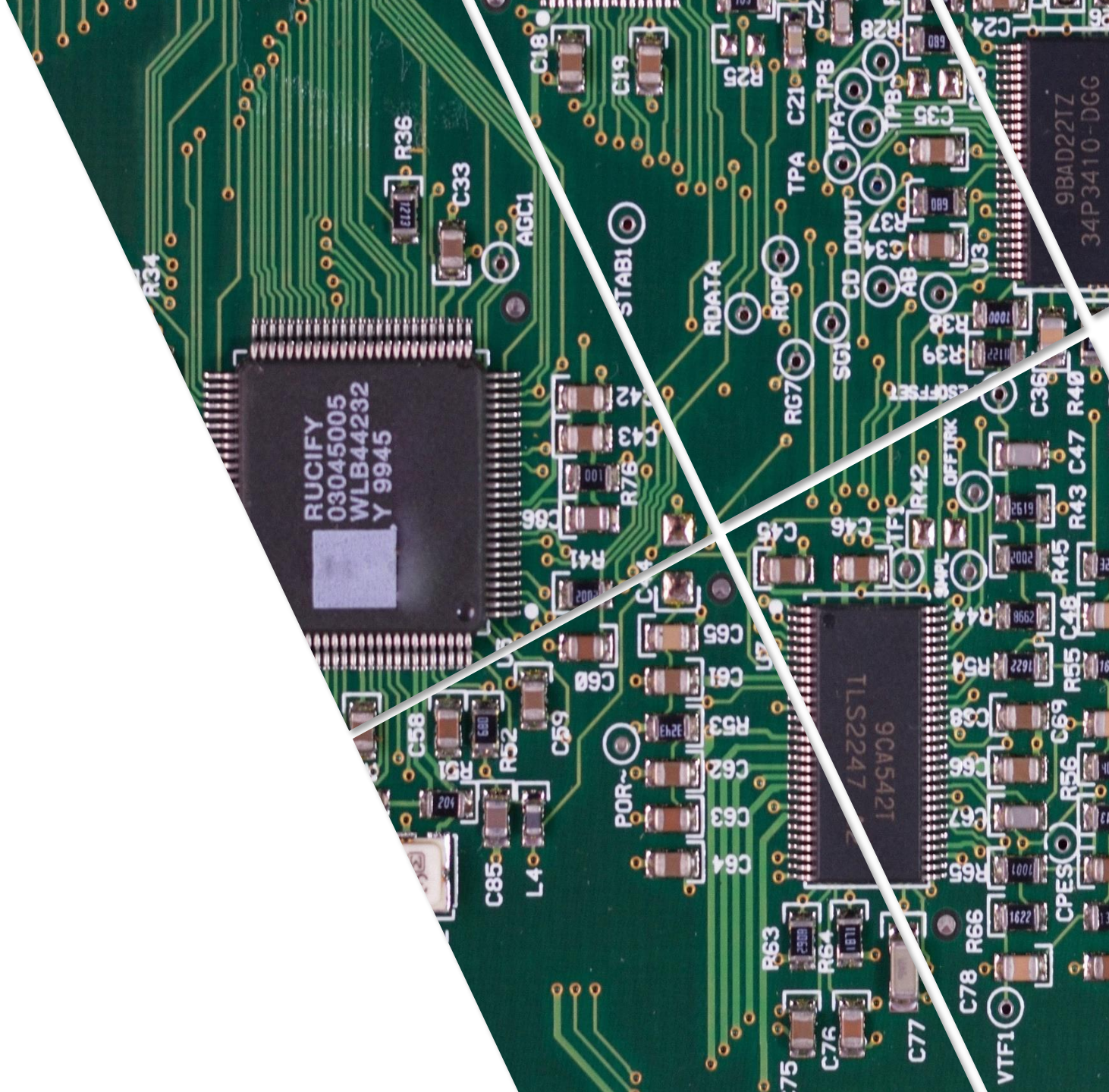
Performance em Sistemas Ciberfísicos

Aula 6 – CPU CACHE

Frank Coelho de Alcantara – 2023 -1



Como isso funciona

Se 95% de todos os acessos forem feitos na memória mais rápida ($1 \times 10^{-6} s$) e o restante na memória mais lenta ($100 \times 10^{-6} s$) o tempo médio de acesso será:

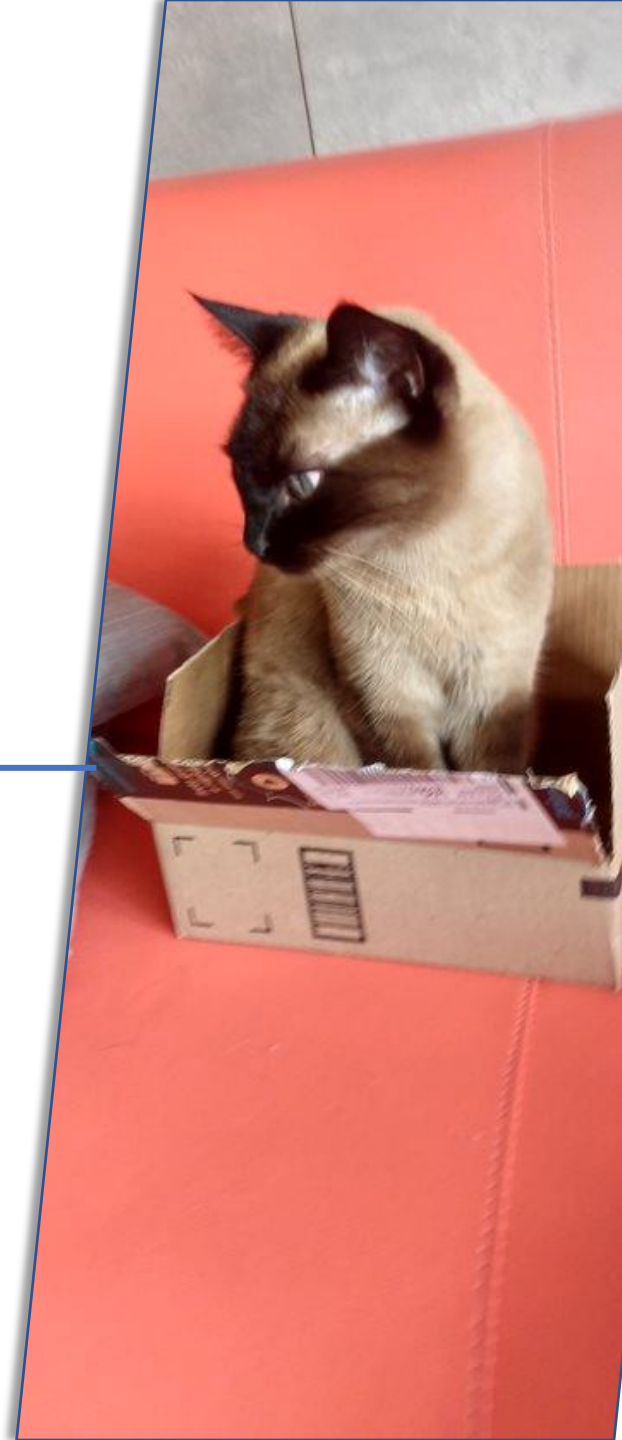
$$T_{med} = \frac{95 \times (1 \times 10^{-6}) + 5 \times (100 \times 10^{-6})}{100}$$

$$\boldsymbol{T_{med} = 5,95 \times 10^{-6}}$$

Design de um Cache – Funções de Mapeamento

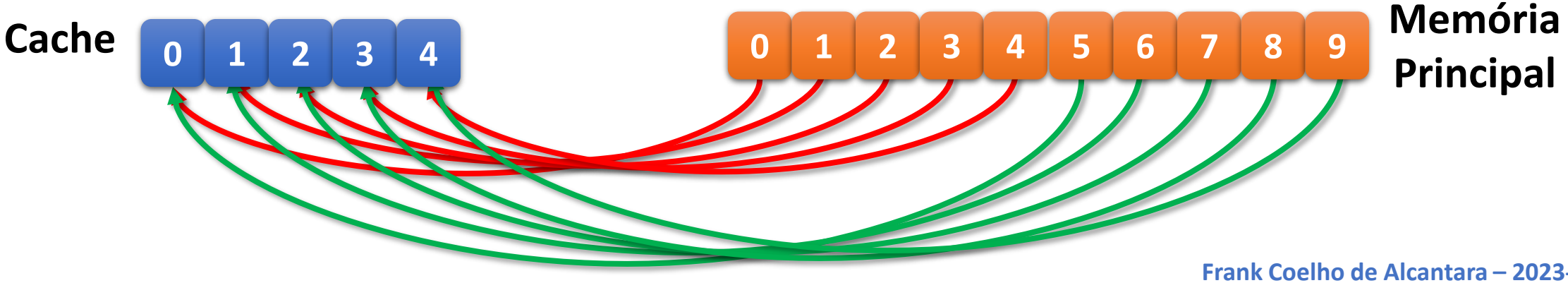
- A função de mapeamento é um dos métodos para localizar um endereço de memória dentro do Cache. Usamos uma função de mapeamento sempre que copiamos um bloco da memória para o Cache, ou quando precisamos encontrar um dado no Cache. Existem três tipos de Funções de Mapeamento:
  - Direto;
  - Associativo;
  - Associativo por Conjunto.

Mapeamento Direto

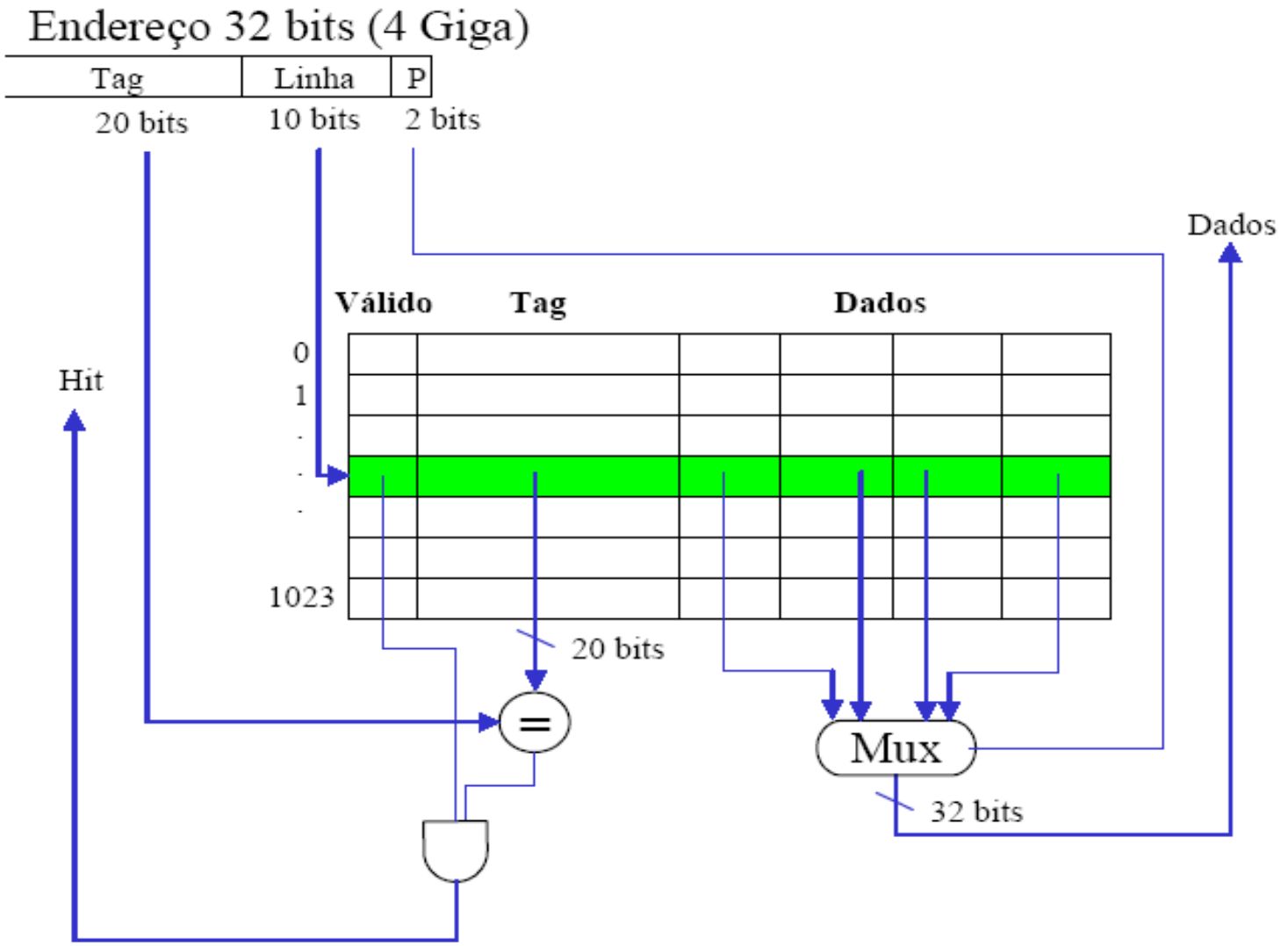


Mapeamento Direto

Por exemplo se o cache contem 5 blocos, o bloco 0 da memória principal mapeia para o bloco 0 do cache, o bloco 1 da memória principal para o bloco 1 do cache, o bloco 2 da memória principal para o bloco 2 do cache... O bloco 5 da memória principal para o bloco 0 do cache...



Exemplo Divisão de Blocos



- Exemplo da divisão de blocos em um Cache com $1024\ (2^{10})$ linhas e blocos com 4 palavras de 32 bits.
- Bit de validade e Tag

Para Melhorar o Entendimento

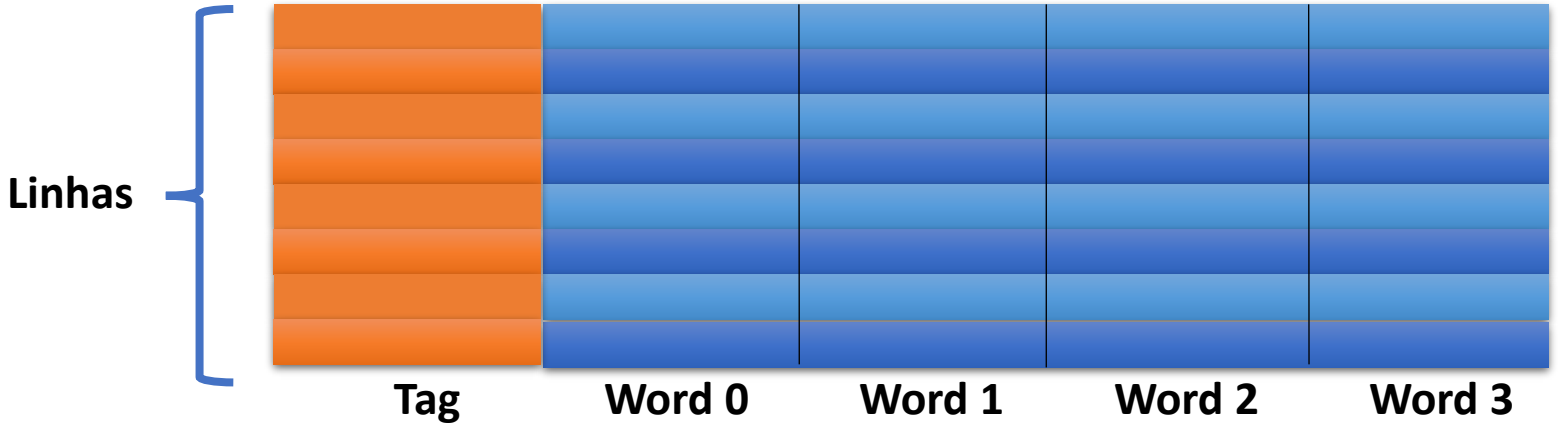
- O Mapeamento Direto faz uma associação entre os endereços na memória e os endereços no Cache. Como o Cache é menor que a memória principal. É mais fácil considerar que a memória principal e dividida em blocos.
- Ou seja, não vamos olhar para a Memória Principal como um conjunto de células de memória. Vamos olhar como um conjunto de blocos.

Continuando os Mapeamentos: Direto

- Neste modelo, cada bloco da Memória Principal poderá ser colocado em qualquer uma das linhas do Cache. Não há uma posição pré-definida;
- Em uma requisição de memória, podemos colocar o bloco 150 da Memória Principal na linha 35 do Cache, em outra requisição ele pode ser colocado na linha 100. *Como saber qual bloco está em cada uma das linhas da cache?*
- Podemos usar alguns bits para isso, vamos chamar de *Tag.*

Porque a Tag?

- É preciso guardar em cada linha do Cache, além das informações provenientes da Memória Principal, o dado, uma marcação (*tag*) esta tag informa o número do bloco, na Memória Principal que está nesta linha do Cache.



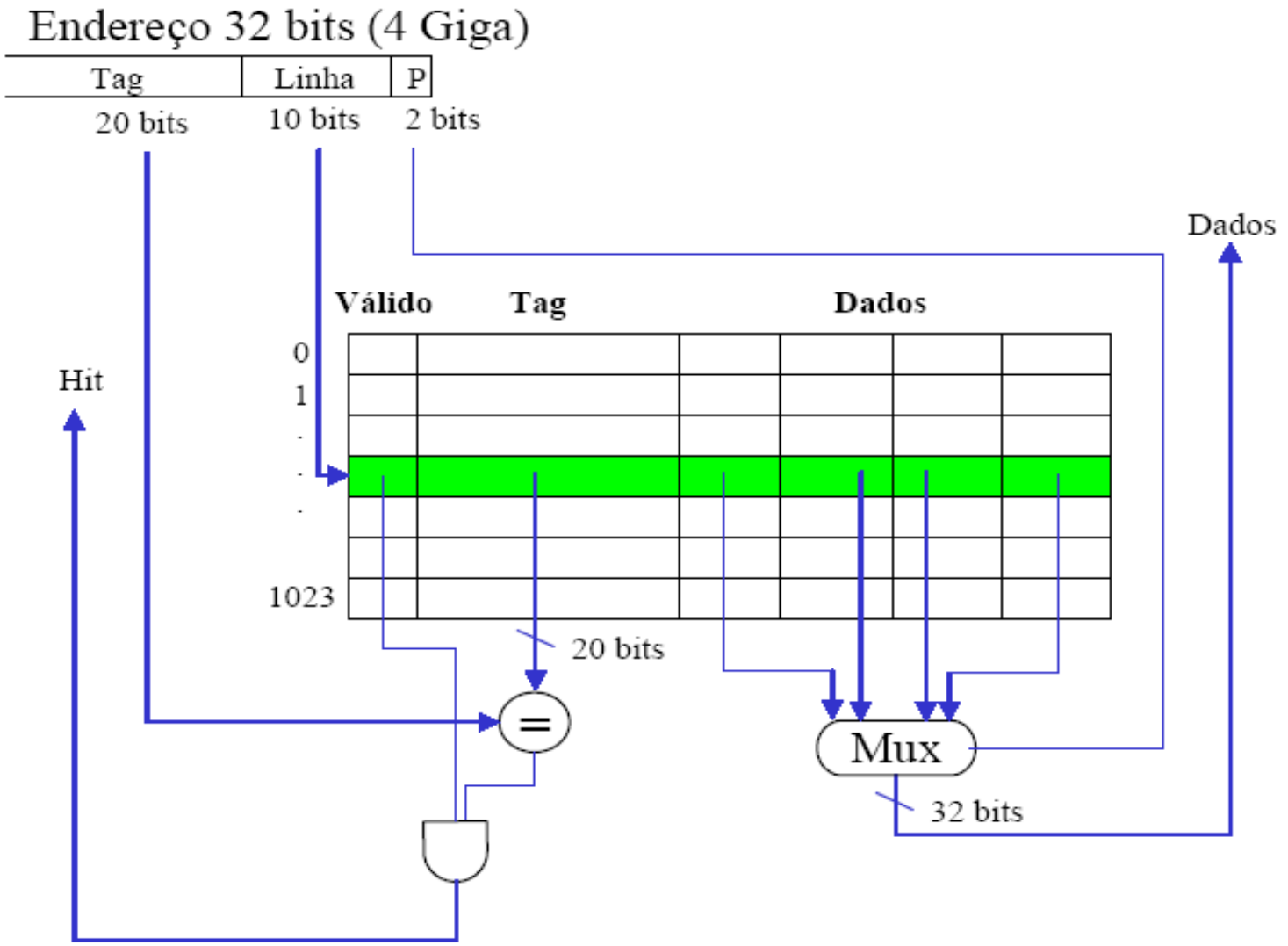
Como isso funciona

- Quando a CPU pede uma informação através de um endereço da Memória Principal, este endereço será dividido em duas partes:
  - A primeira parte indica em qual bloco está a informação desejada;
  - A segunda parte indica qual é a word procurada dentro do bloco;

O Hardware entra em cena!

- O dispositivo que controla o Cache verifica se em alguma das linhas do Cache existe um *tag* idêntico ao número do bloco pedido pela CPU;
  - Se existir, a word necessária é obtida da linha do Cache e entregue à CPU;
  - Se não existir, o Cache pega um bloco inteiro da Memória Principal e coloca-o em alguma linha que esteja disponível no Cache. Depois repassa à CPU a word pedida.

Exemplo Divisão de Blocos



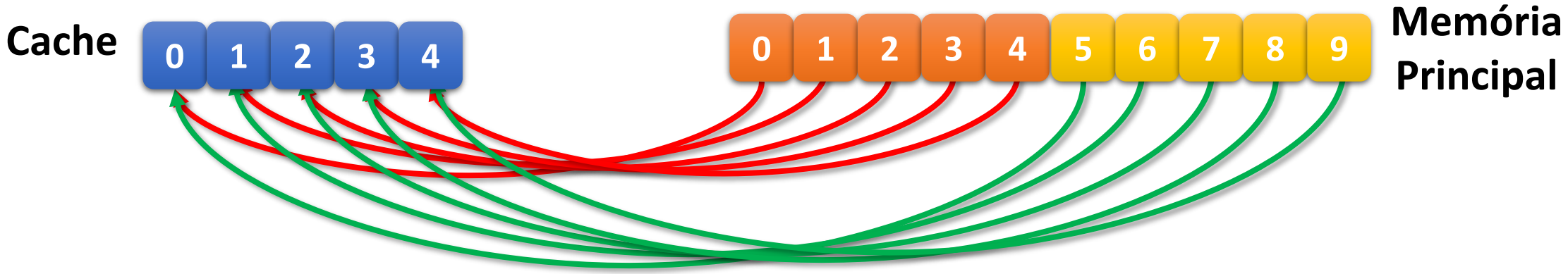
- Exemplo da divisão de blocos em um Cache com $1024\ (2^{10})$ linhas e blocos com 4 palavras de 32 bits.
- Bit de validade e Tag

Voltando ao Mapeamento Direto

- Neste modelo, cada bloco da Memória Principal irá aparecer em apenas uma, e somente uma, linha do Cache;
- Existe um posicionamento prévio de onde cada bloco pode estar.
- E cada bloco estará apenas neste lugar. Ou seja, Se o bloco não estiver na linha correspondente, não estará em mais lugar algum do Cache;

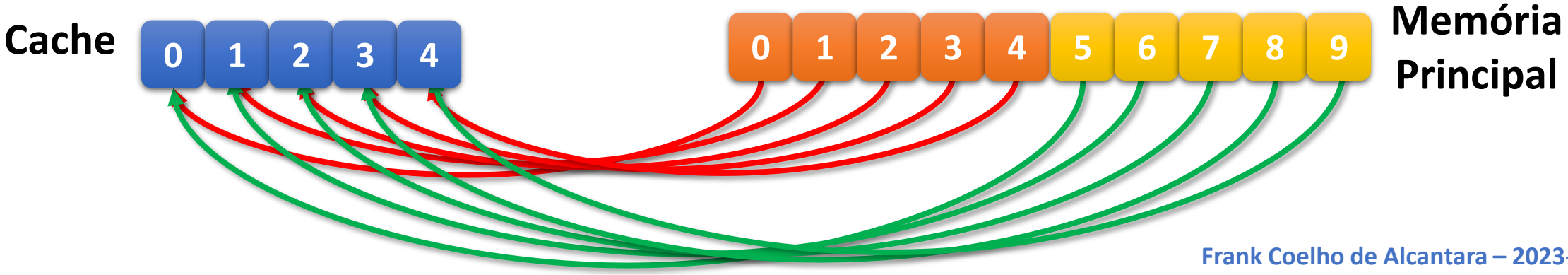
Mapeamento Direto

Por exemplo se o cache contem 5 blocos, o bloco 0 da memória principal mapeia para o bloco 0 do cache, o bloco 1 da memória principal para o bloco 1 do cache, o bloco 2 da memória principal para o bloco 2 do cache... O bloco 5 da memória principal para o bloco 0 do cache...



Voltando na Tag

- A *Tag* serve para identificar qual dos blocos da Memória Principal está ocupando aquela linha da Cache.
- Vários blocos podem ocupar a mesma linha da Cache, mas um mesmo bloco ou está na linha específica para ele ou não está no Cache. Por isto, esta técnica é chamada de mapeamento direto.



Tudo isso precisa ser calculado

- Antes de qualquer coisa, lembre que a função do Cache é armazenar os dados, e instruções, que serão necessárias a seguir. Segundo o Princípio da Localidade.
- Segundo, a Memória Principal será dividida em Blocos. Se a memória tem X bytes e cada bloco será composto por Y bytes, a quantidade de blocos QB será:

$$QB = X/Y$$

Ainda Calculando

- Se a memória Cache tem L linhas, isto significa que os blocos serão divididos em L grupos distintos. E a quantidade de blocos por grupo BG será dada por:

$$BG = QB/L$$

- A quantidade de bits que a *Tag* irá precisar dependerá apenas da quantidade de blocos por grupo (BG). E pode ser facilmente calculada por:

$$Bits = \log_2 BG$$

Exemplo

- A memória principal de um computador possui 4GBytes. O Cache possui 1024 linhas, podendo armazenar 64 Bytes de dados em cada linha. Sendo assim, teremos:
  - A quantidade de blocos é $4G/64 = 64M$ blocos.
  - Como temos 64M blocos Na Memória Principal e existem 1024 linhas de Cache, cada linha poderá conter 64k blocos (um de cada vez), já que:

$$BG = \frac{QB}{L} = \frac{64\text{M}}{1024} = 64\text{K}$$

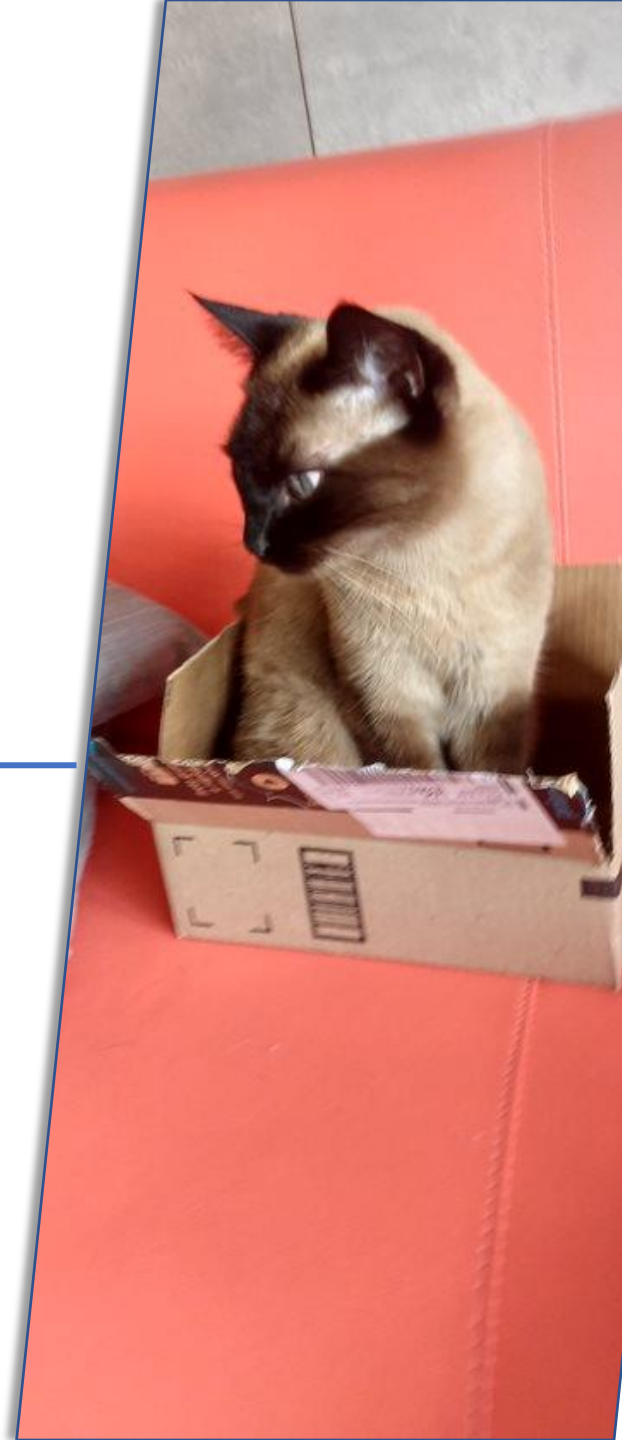
Exemplo

- A quantidade de blocos é $4G/64 = 64M$ blocos.
- A quantidade de Blocos por grupo $\mathrm{BG} = \frac{64\mathrm{M}}{1024} = 64\mathrm{K}$.
- O tamanho do tag será: $Bits = \log2\,(64K) = 16\ bits$.
- Um endereço de memória será composto por 32 bits, pois $2^{32} = 4G$.

Atenção

- Se o seu programa ficar alternando entre dados que estão em blocos diferentes da Memória Principal que não caibam no Cache estes blocos serão carregados e descarregados do Cache alternadamente. Levando a uma perda de eficiência por causa do Princípio da Proximidade.

Mapeamento Associativo



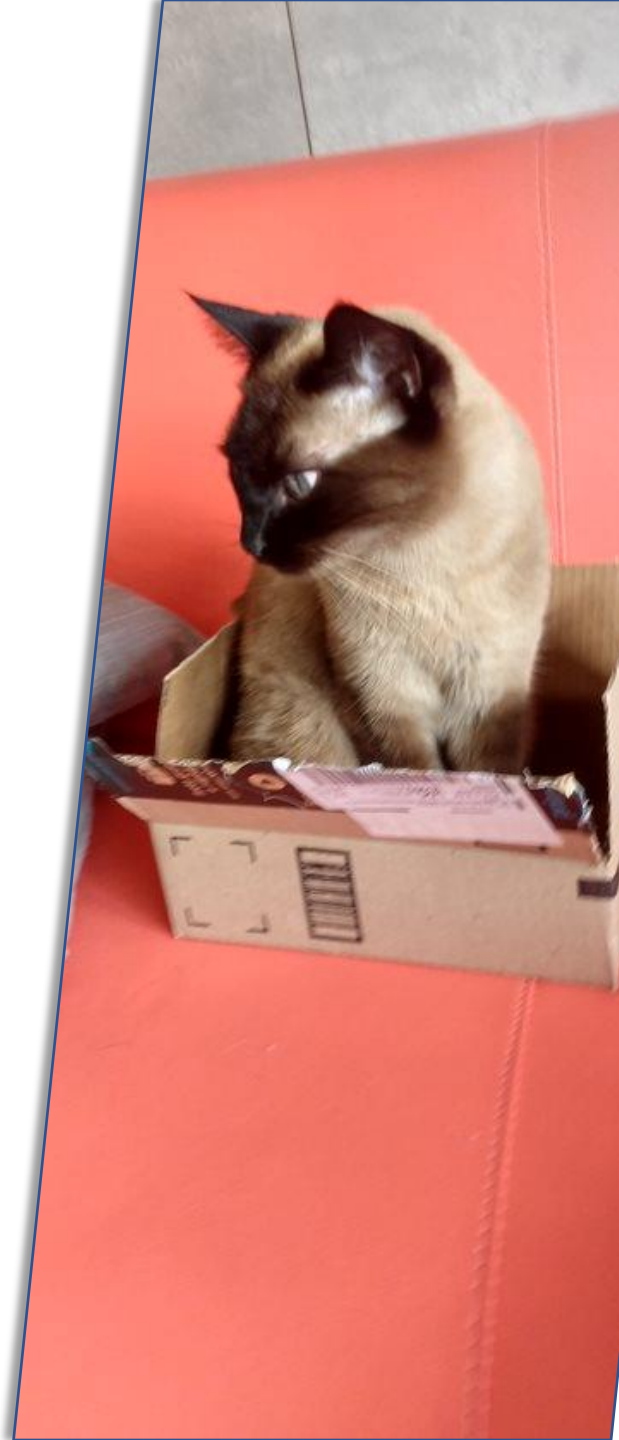
Mapeamento Associativo

- Permite que cada bloco da Memória Principal seja carregado em qualquer linha do Cache.
- Para a identificação contamos apenas com a *Tag* e um identificador para as *words*.
- A *Tag* identifica um bloco da Memória Principal de forma unívoca. Para saber se um endereço está no Cache o Hardware compara a *Tag* do endereço da Memória Principal com as *Tags* que estão em todas as linhas do Cache.

Mapeamento Associativo - Pesquisa

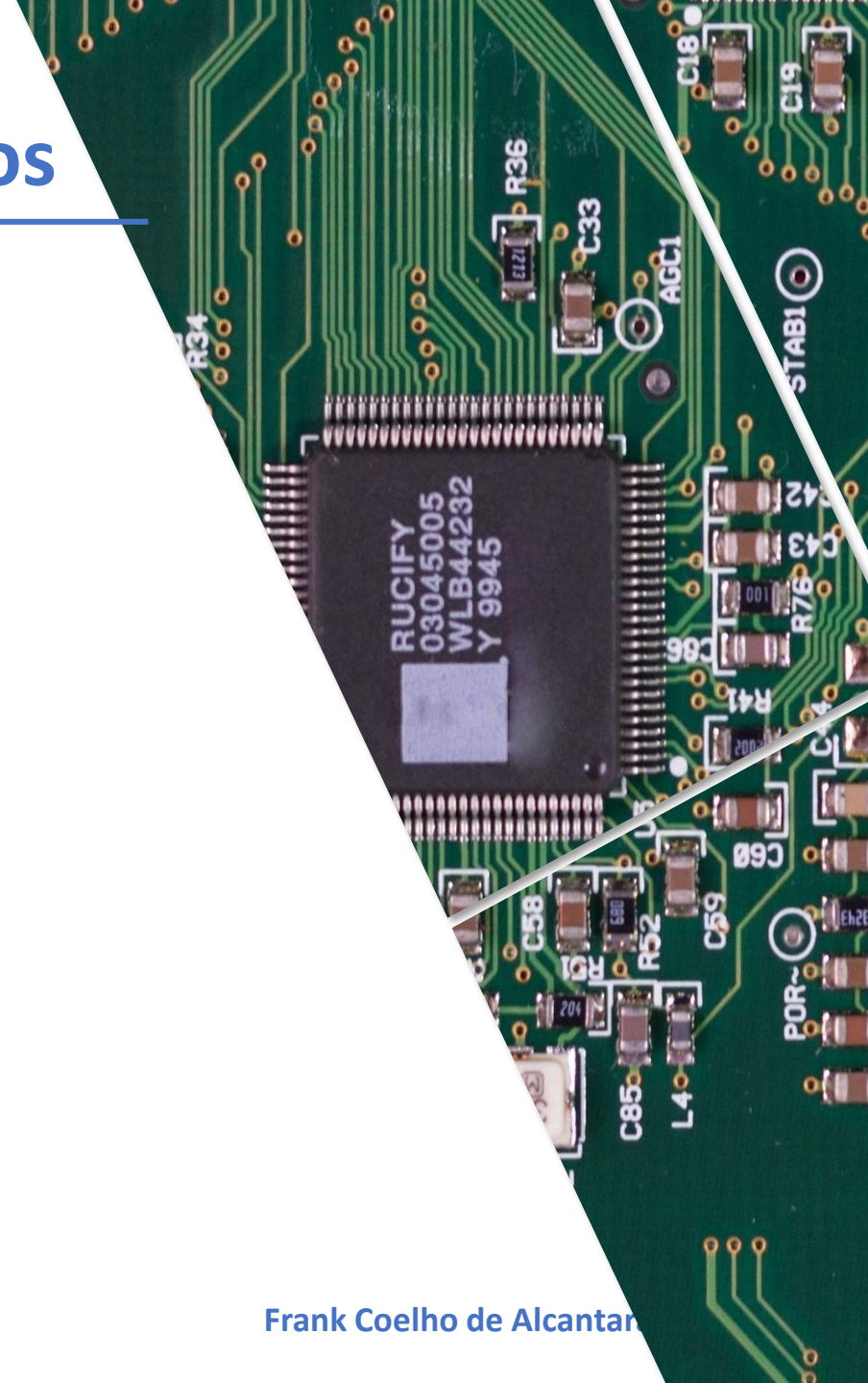
- Para saber se um determinado endereço está no Cache temos que procurar todas as *Tags* armazenadas.
- O custo deste processo é alto. A busca é feita em paralelo usando vários comparadores, um por linha por exemplo.
- Este esquema só compensa para Caches muito pequenos.
- Em comparação com o Mapeamento Direto, aqui não temos divisão por linhas.

Mapeamento Associativo por Conjuntos



Mapeamento Associativo por Conjuntos

- Cada linha do Cache pode conter mais de um bloco.
- Diversos blocos, um conjunto, podem estar relacionados com a mesma linha.
- Quando a linha for descoberta, será preciso ainda analisar cada conjunto de blocos para saber se o bloco requerido está ou não no Cache.



Mapeamento Associativo por Conjuntos

- Para O Cache funcionar corretamente ainda é necessário considerar o que acontecerá se o cache estiver cheio, ou seja, é preciso trazer um dado da Memória Principal e colocar em uma linha do Cache onde já existe um dado.
  - No mapeamento direto, o novo dado só pode ocupar um único lugar. Logo, o dado antigo será substituído.
  - Nos Mapeamento Associativo e Associativo por Conjunto, precisaremos escolher que bloco será retirado.

Técnicas de Reposição

- LRU (*least recently used*) – o controlador da cache escolhe o bloco que foi utilizado há mais tempo. Este critério tem por base o princípio da localida temporal;
- LFU (*least frenquently used*) – o sistema escolhe o bloco que foi menos utilizado;
- FIFO – o sistema escolhe para ser retirado, o bloco que foi colocado primeiro na cache, independente do uso dele;
- aleatória: um bloco qualquer é escolhido;

Memória Principal – Leitura e Escrita

- A Memória Principal permite que se faça dois tipos de operações: leitura e escrita;
- A Memória Principal pode ser acessada tanto pela CPU quanto por componentes E/S. Um dado pode ter sido alterado na cache e não na Memória Principal (desatualizada). Um componente E/S pode ter alterado o dado diretamente na Memória Principal e não na cache (desatualizada).

Memória Principal – Leitura e Escrita

- Uma mesma Memória Principal pode ser acessada por diversas CPUs, cada uma contendo sua própria cache. A alteração feita em um Cache deve ser refletida na Memória Principal e, consequentemente, nos demais Caches.
- Existem técnicas específicas para isso.

Técnicas de Escrita e Leitura

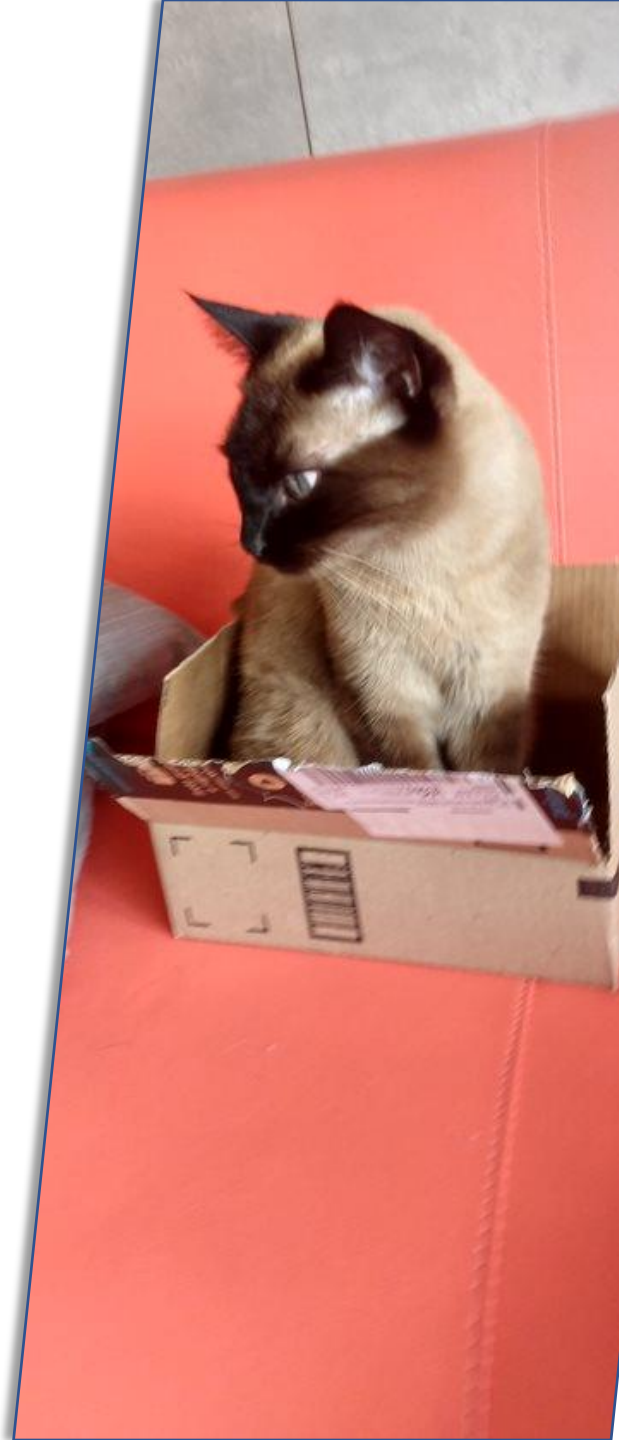
- Escrita em ambas (*write through*):
  - Cada escrita em uma palavra do Cache resulta na escrita da palavra correspondente da Memória Principal. Se houver outros Cache, elas também serão alteradas;
- Escrita somente no retorno (*write back*):
  - Não faz atualização simultânea, mas somente quando o bloco for retirado do Cache e se ele tiver sido alterado.

Técnicas de Escrita e Leitura

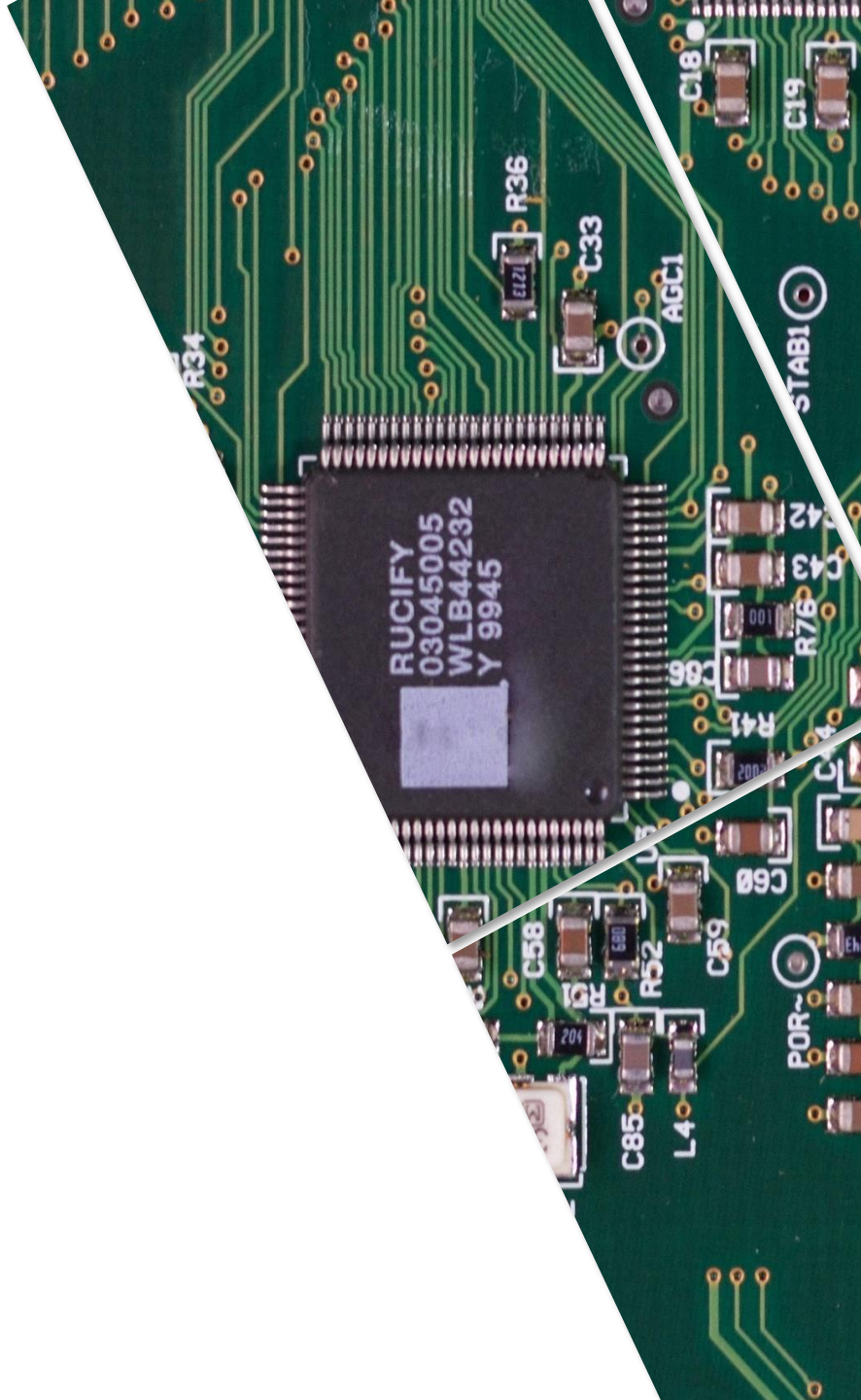
- Escrita uma única vez (*write once*):
  - O bloco da Memória Principal é atualizado quando o bloco do Cache for alterado pela primeira vez. Os demais componentes são alertados de que houve uma alteração e são impedidos de usar o dado. Outras alterações ocorrem apenas no Cache e a Memória Principal só é atualizada quando o bloco sair da cache.

Atividade Formativa

Modifique o código do Simulador de Cache disponível aqui para incluir o algoritmo FIFO e de Substituição Aleatória.



Obrigado!



Frank Coelho de Alcantara – 2023 -1