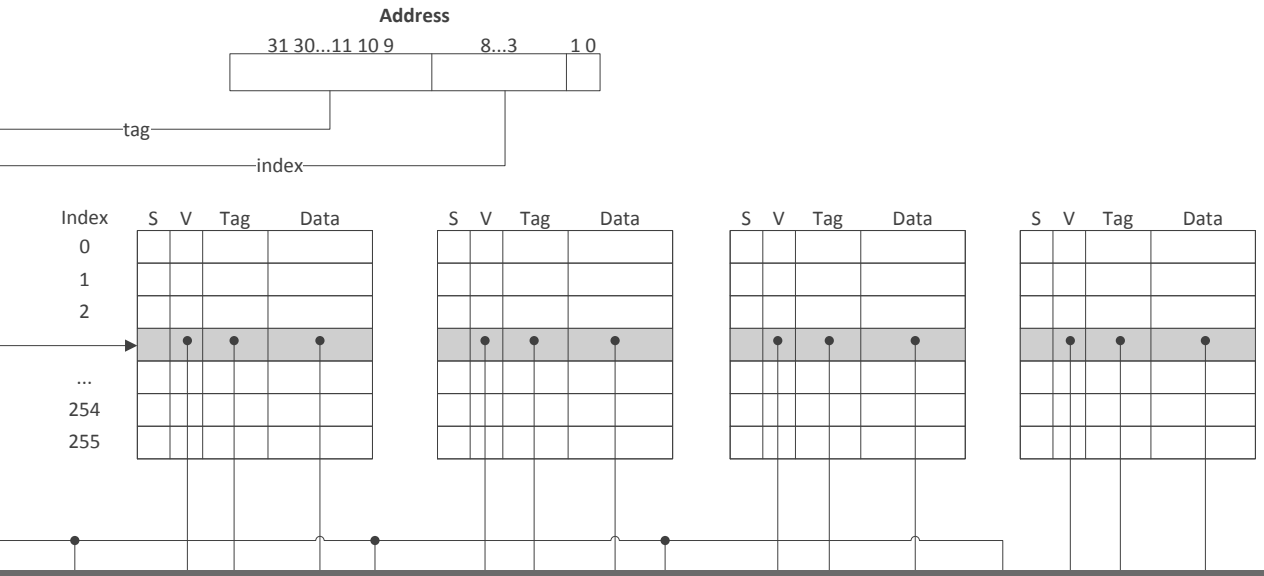


Um Modelo Transparente de Memória *Scratchpad* para Arquiteturas de Propósito Geral

Felipe Silva Loredo
Orientadora: Mariza Andrade da Silva Bigonha
Coorientador: Claudionor José Nunes Coelho Junior

```
%16 = ptrtoint i32* %add.ptr52 to i32
%arrayidx54 = getelementptr inbounds [512 x [512 x i32]]* %graph, i32 0
call void @spload(i32 %16, i32* %arrayidx54) #1
%add56 = add nsw i32 %i48.0160, 16
```

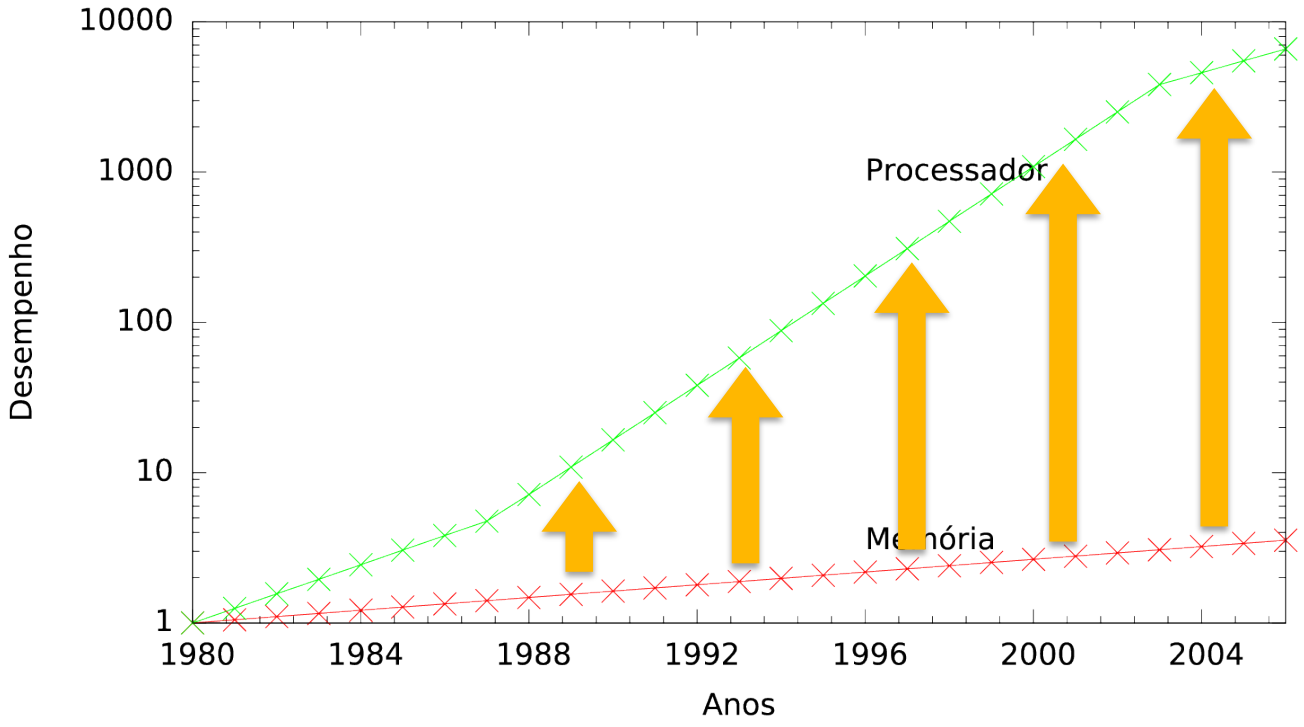
Sumário

Revisão

Burks, Goldstine e Von Neumann (1946)

“Idealmente seria desejável uma **memória de capacidade indefinidamente grande**, tal que qualquer agregado particular de 40 dígitos binários, ou palavras, pudesse **estar imediatamente disponível**, isto é, em um tempo que é comparável ou consideravelmente menor que o tempo de operação de um multiplicador eletrônico rápido. . .” “Parece **não ser fisicamente possível** alcançar tal capacidade. Nós somos, portanto, forçados a reconhecer a possibilidade de construir uma **hierarquia de memórias**, cada uma com uma **capacidade maior** que a anterior, mas que é **menos rapidamente acessível**”

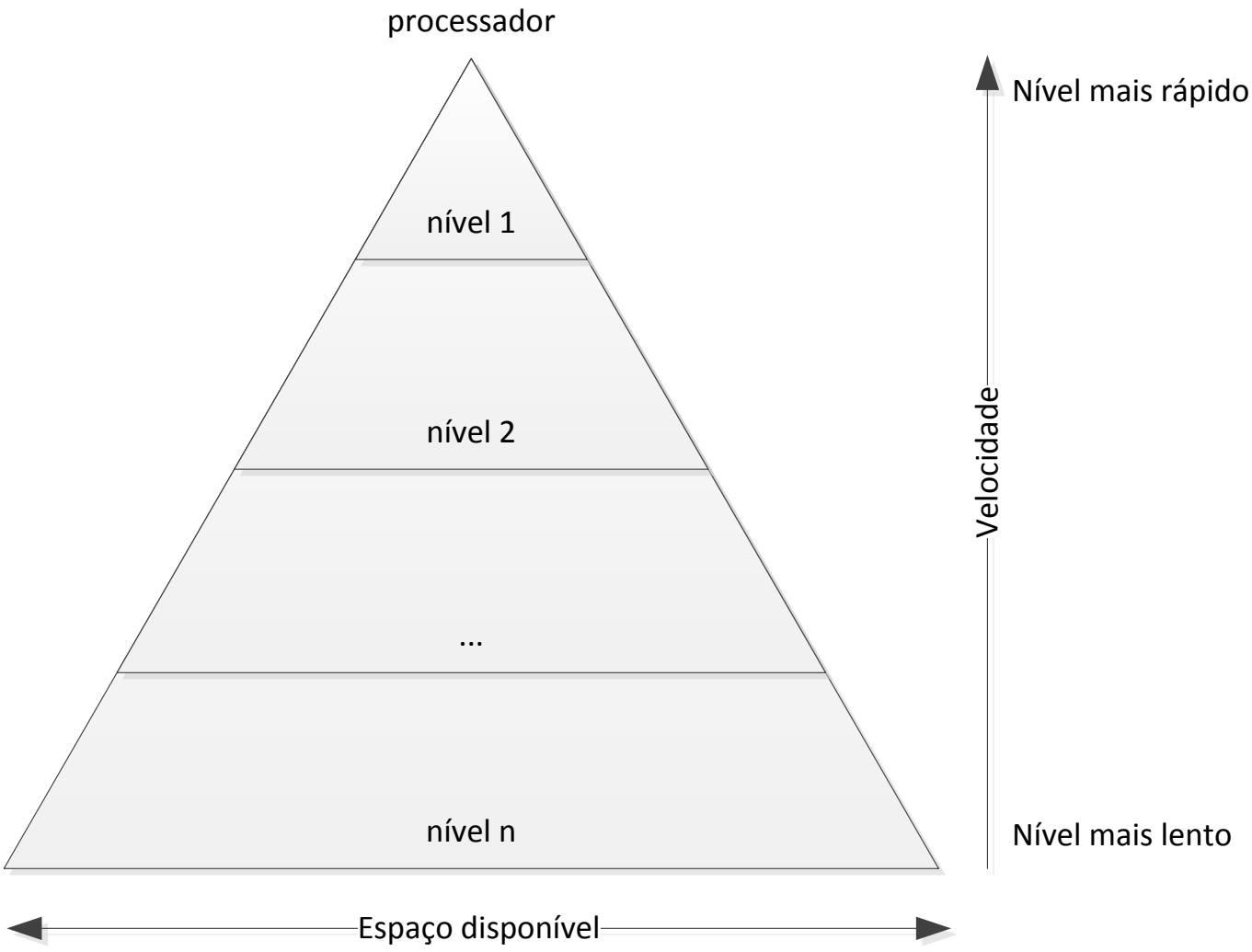
Disparidade entre memória e processador



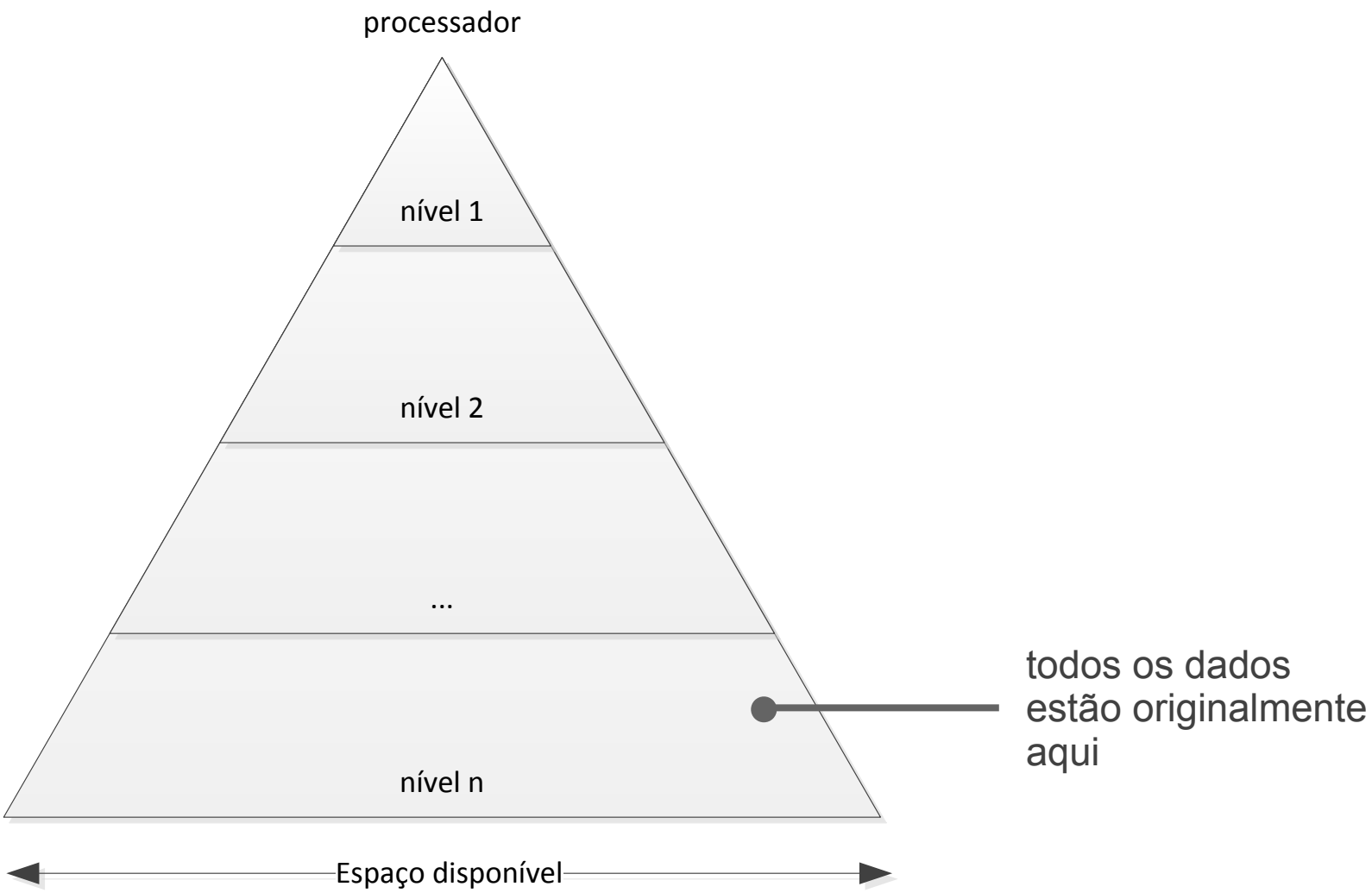
Dados de Patterson & Hennessy (2012)

A **disparidade** entre a velocidade do processador e da memória **aumenta** ao longo dos anos.

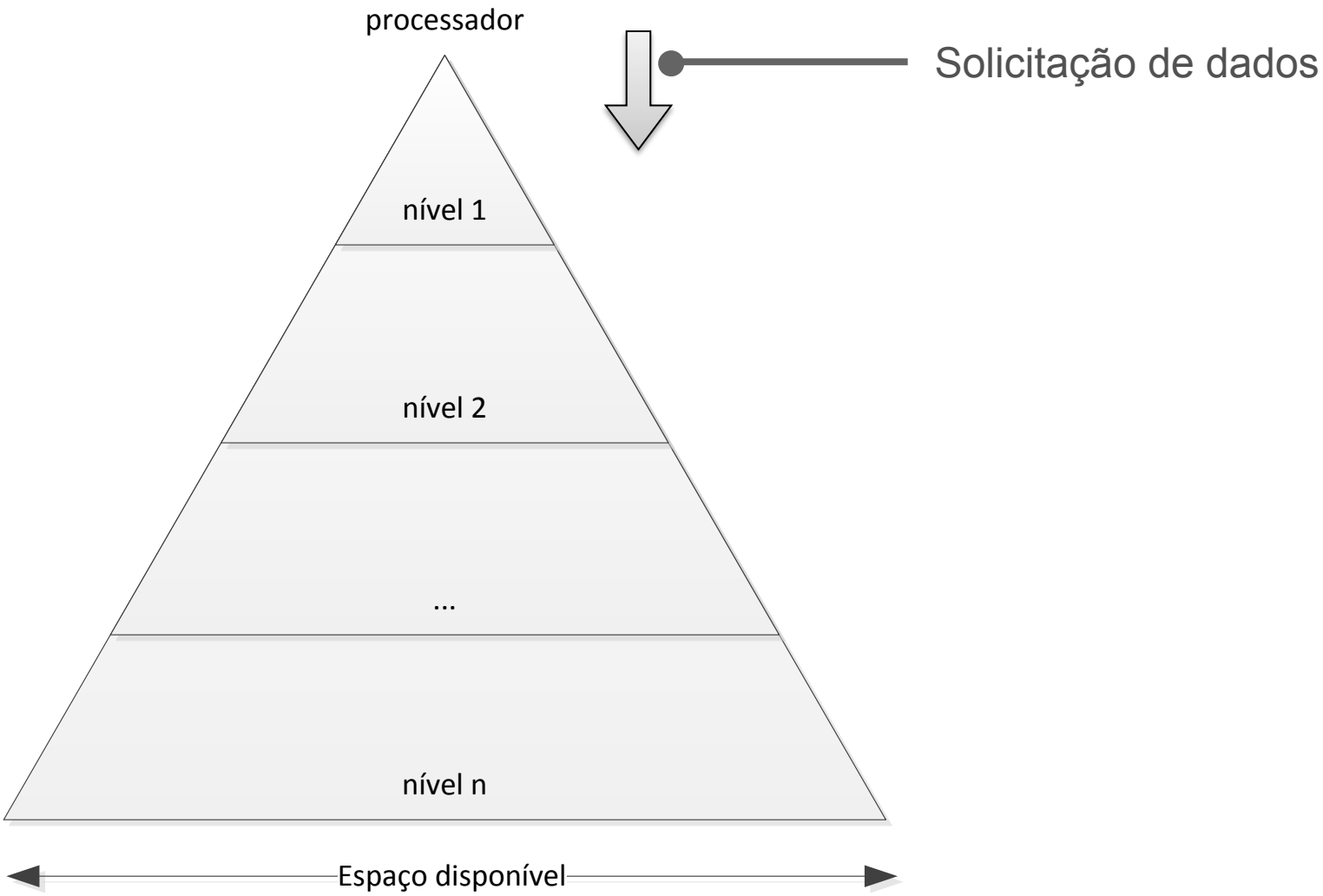
Hierarquia de Memória



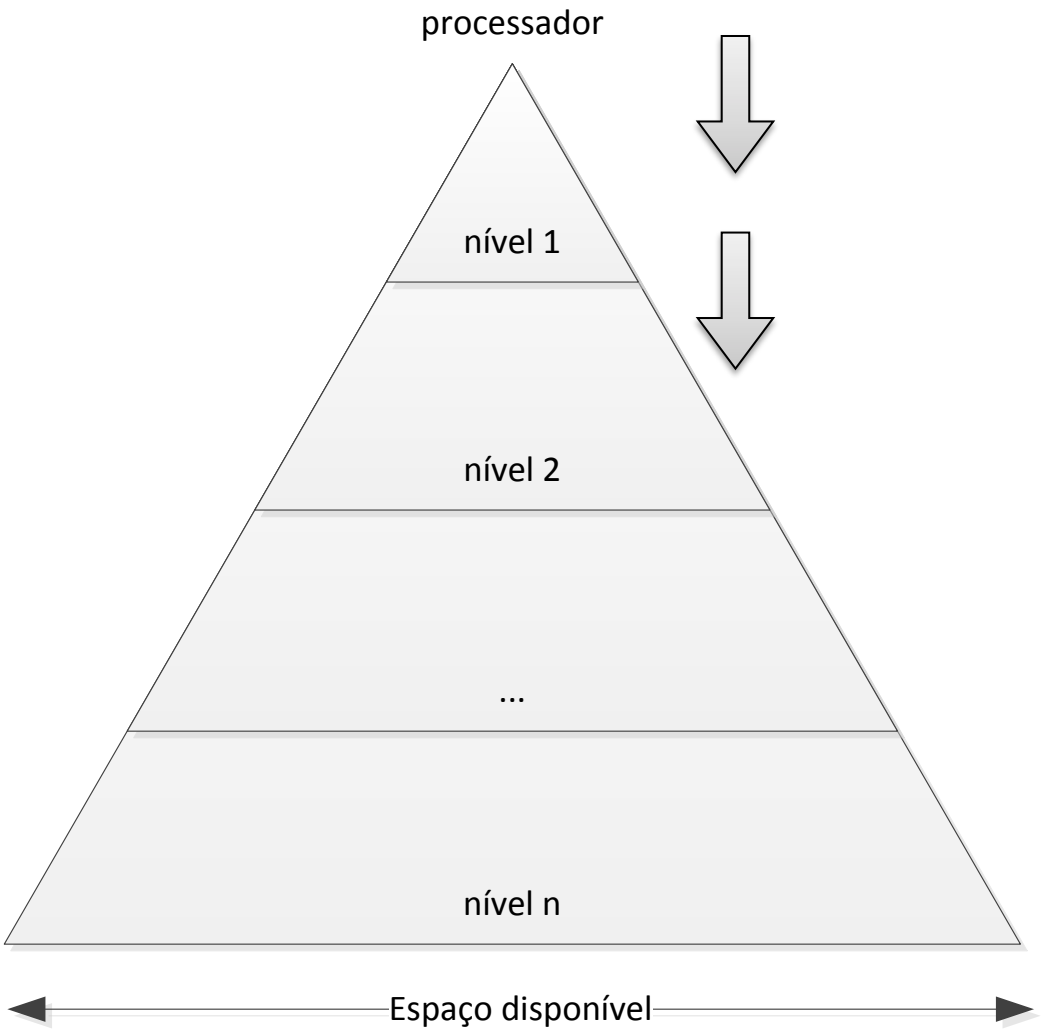
...Hierarquia de Memória



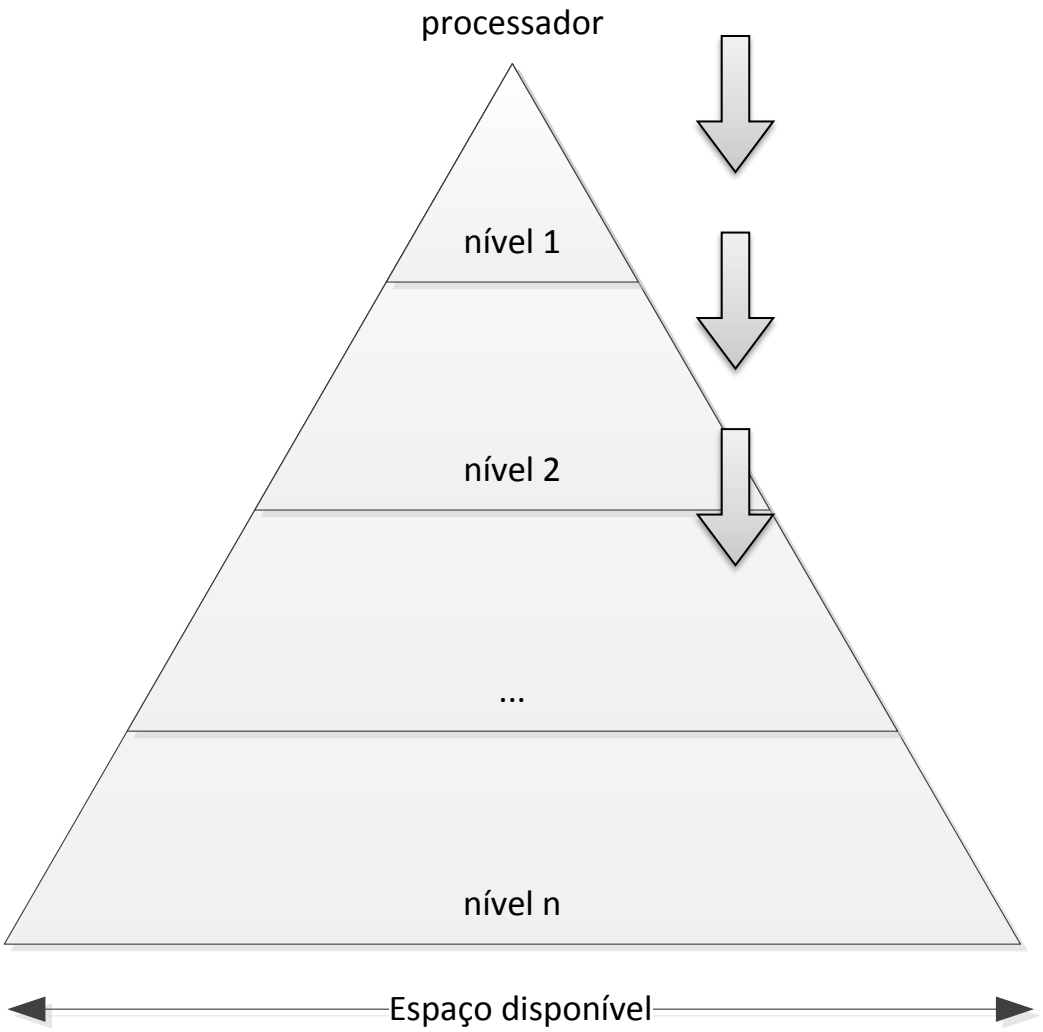
...Hierarquia de Memória



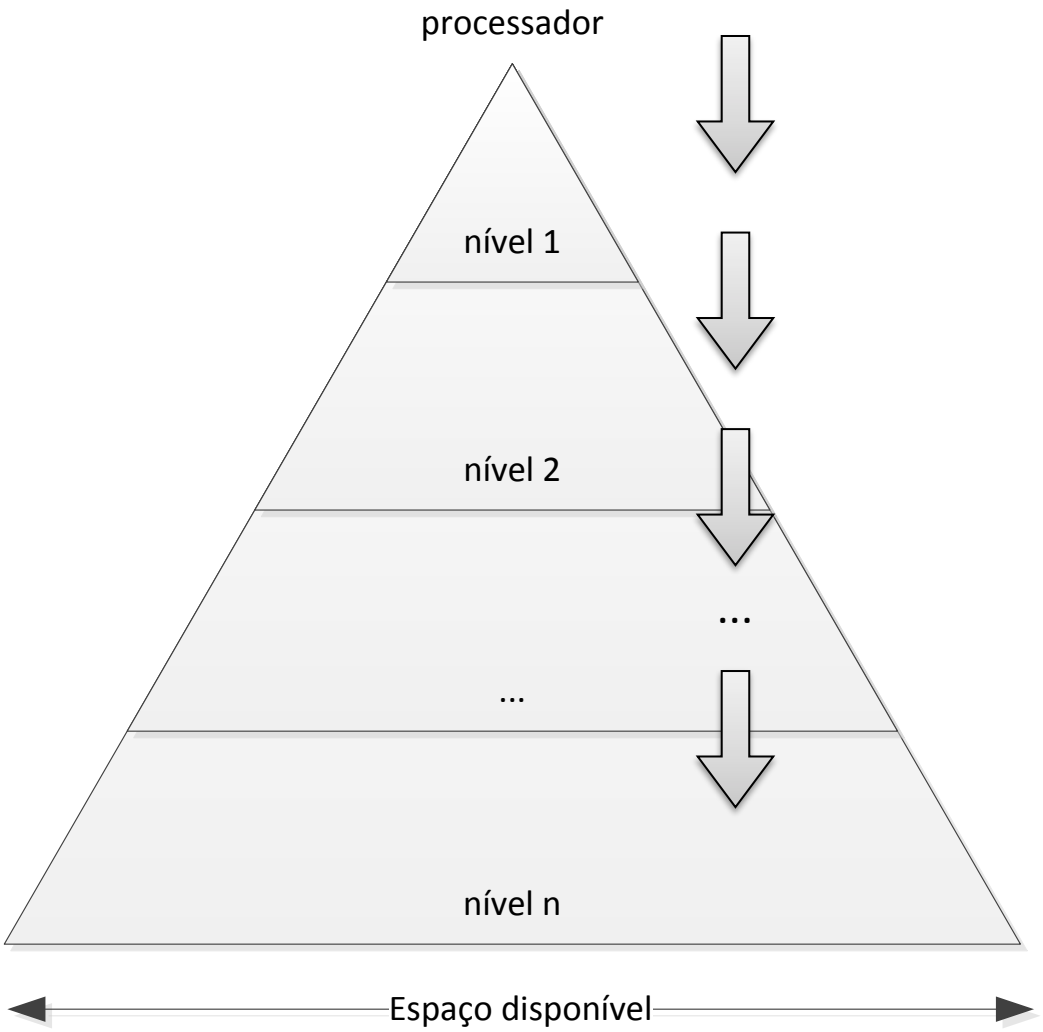
...Hierarquia de Memória



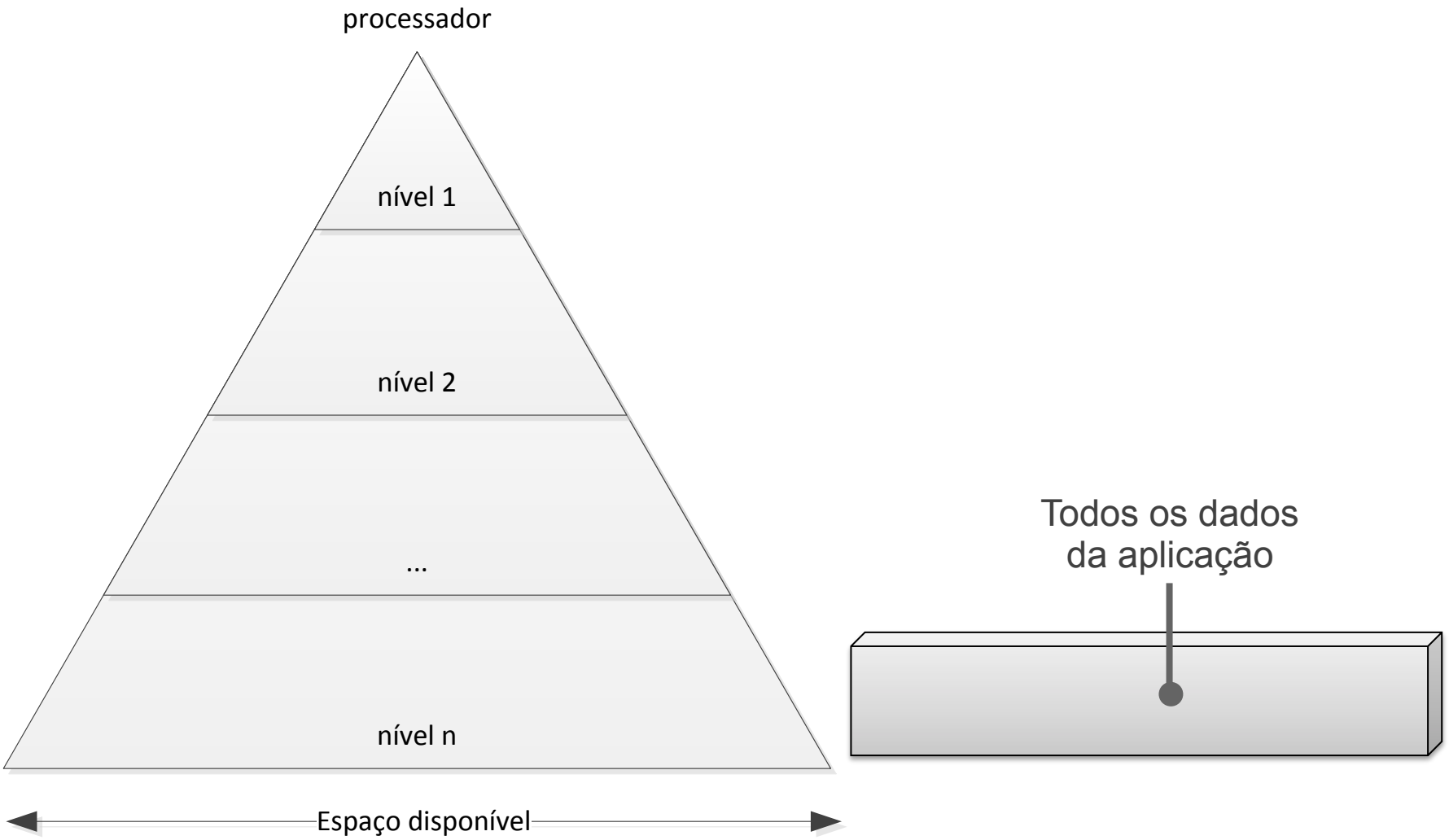
...Hierarquia de Memória



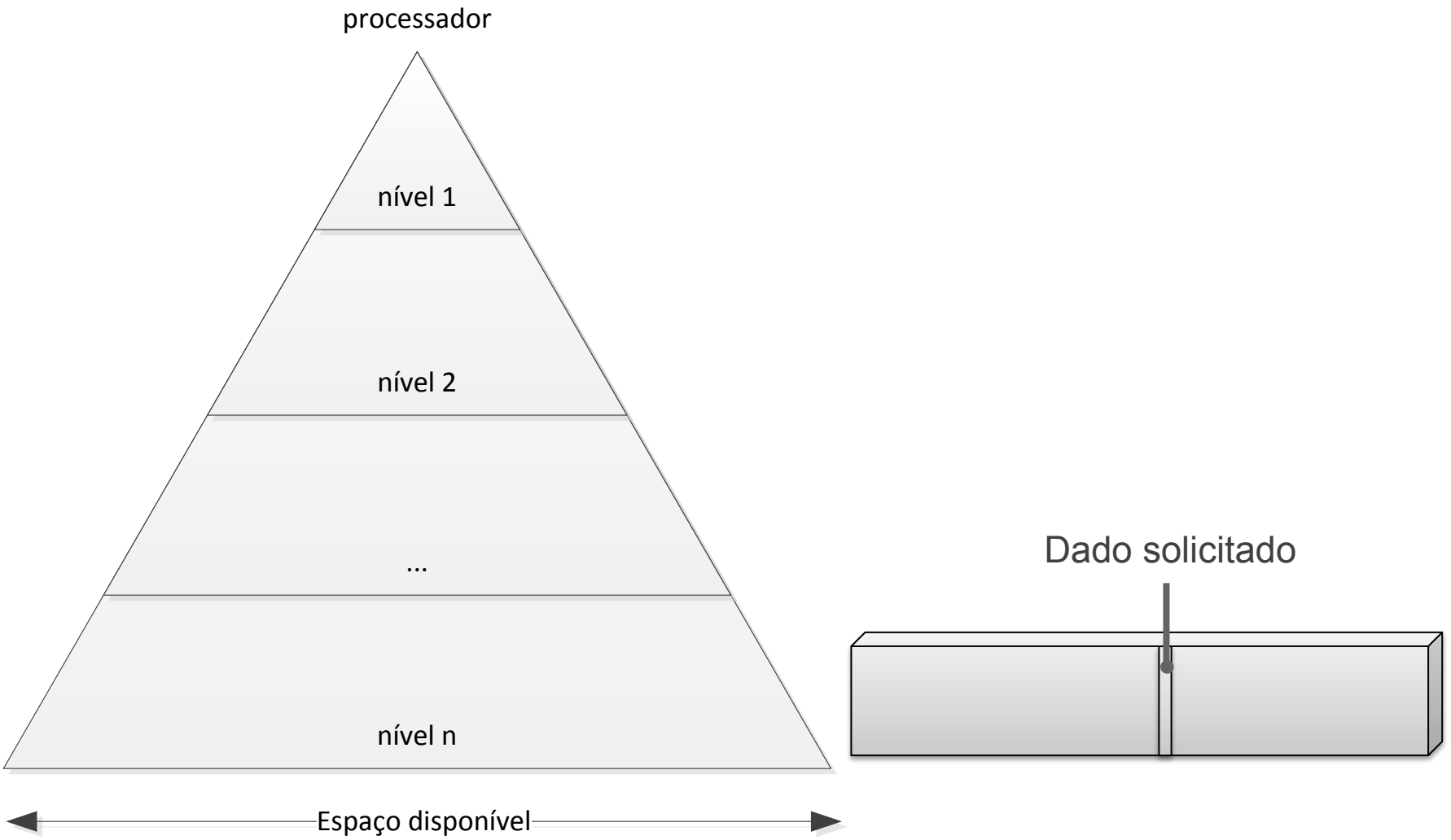
...Hierarquia de Memória



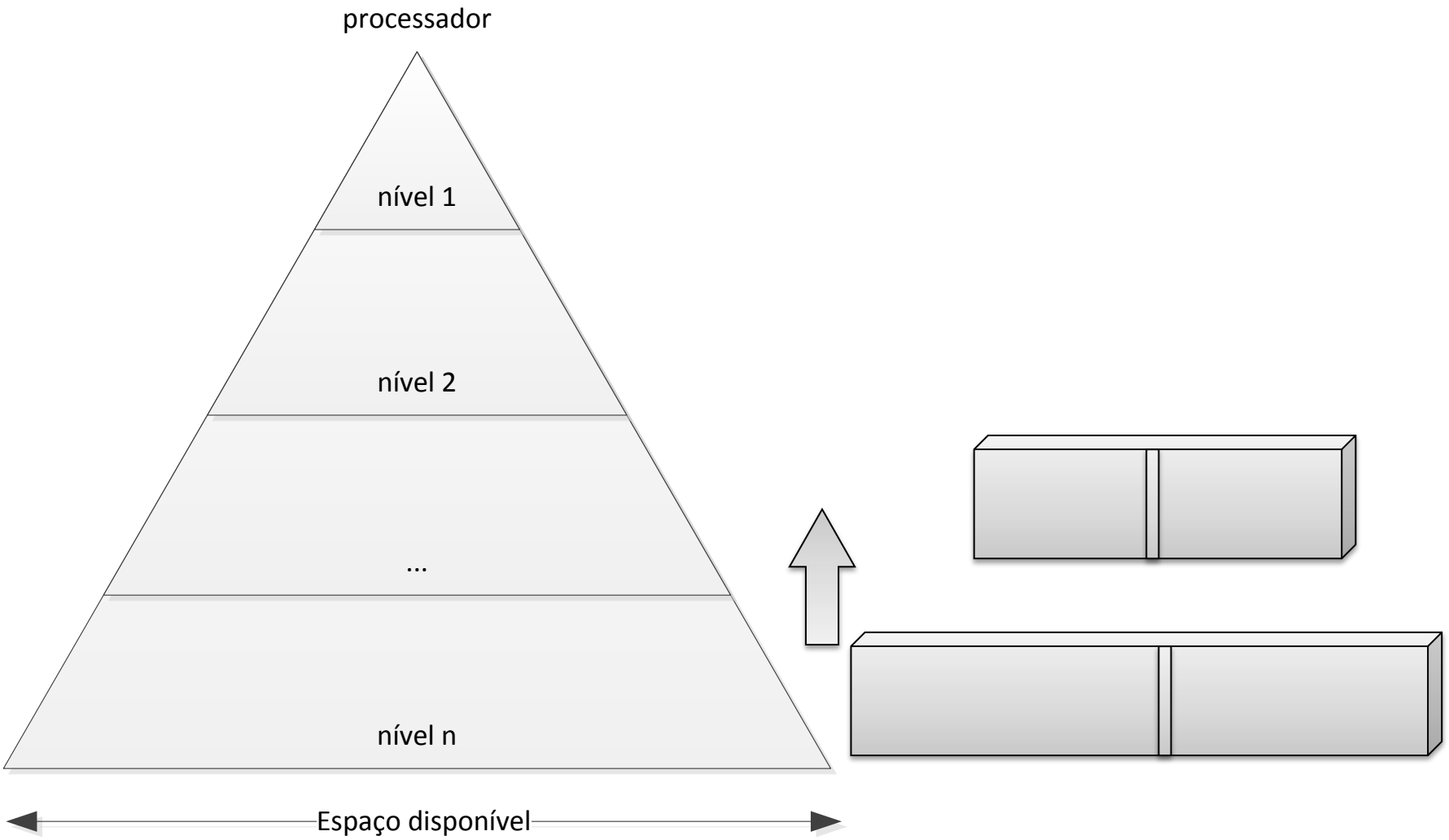
...Hierarquia de Memória



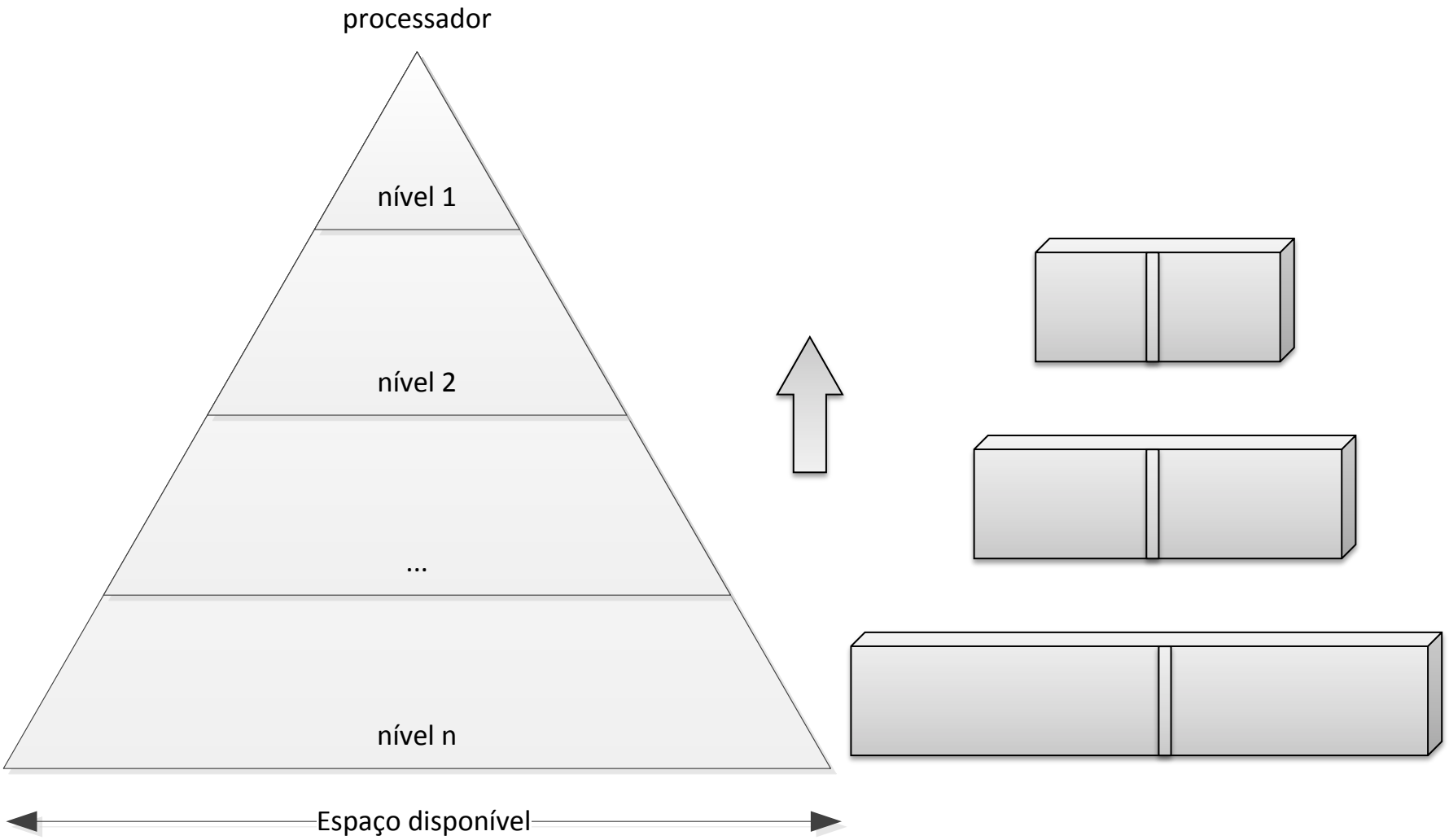
...Hierarquia de Memória



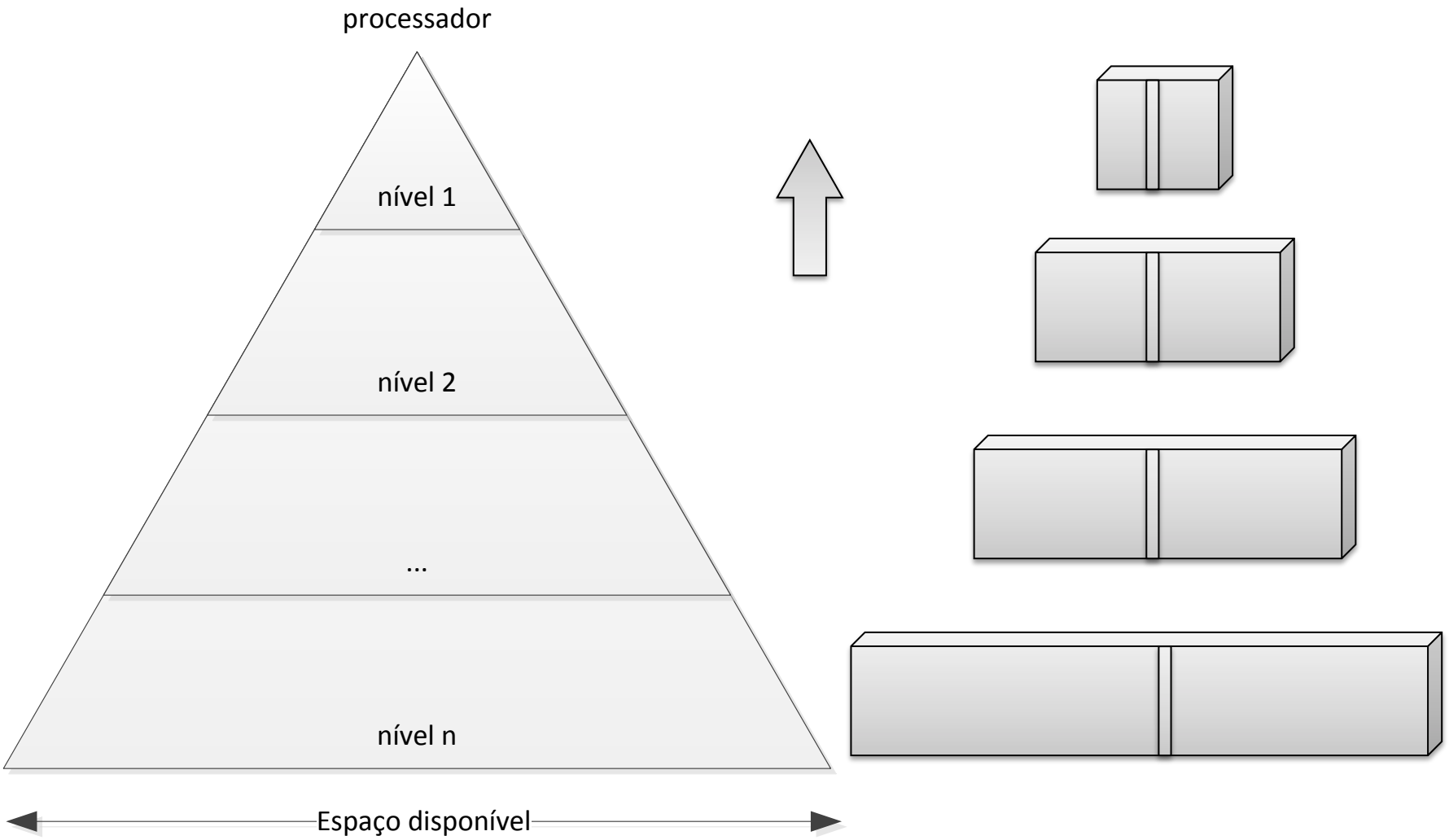
...Hierarquia de Memória



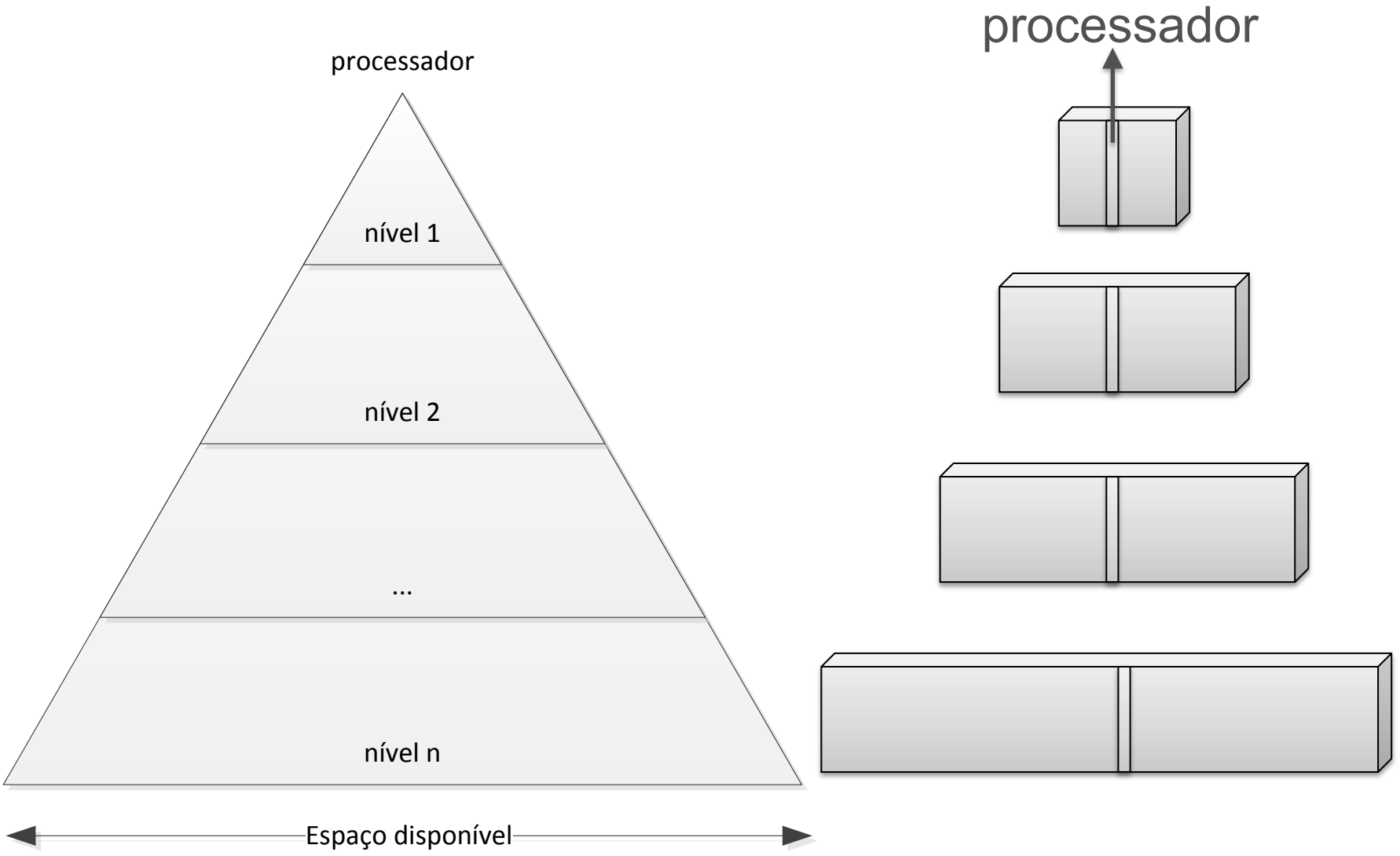
...Hierarquia de Memória



...Hierarquia de Memória



...Hierarquia de Memória

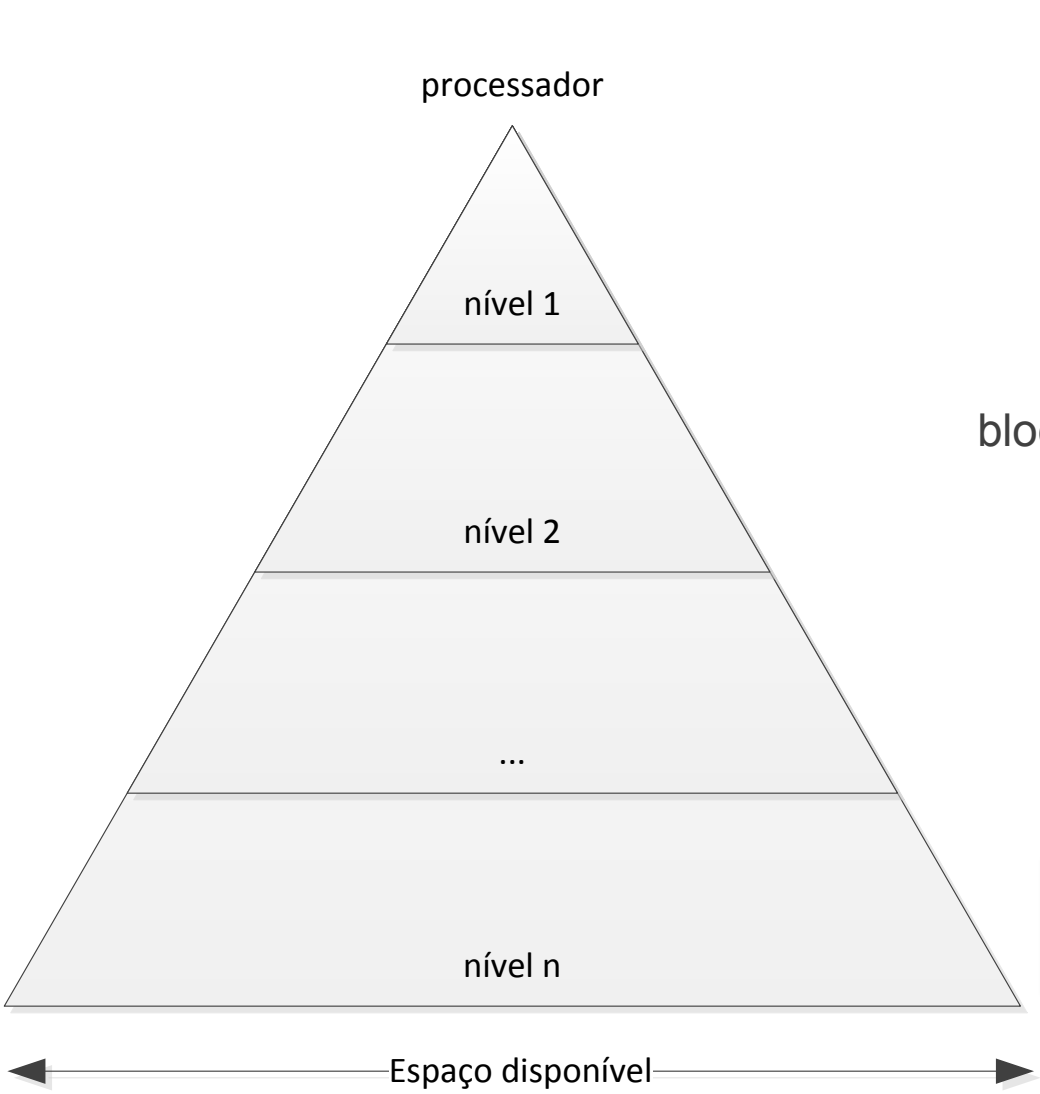


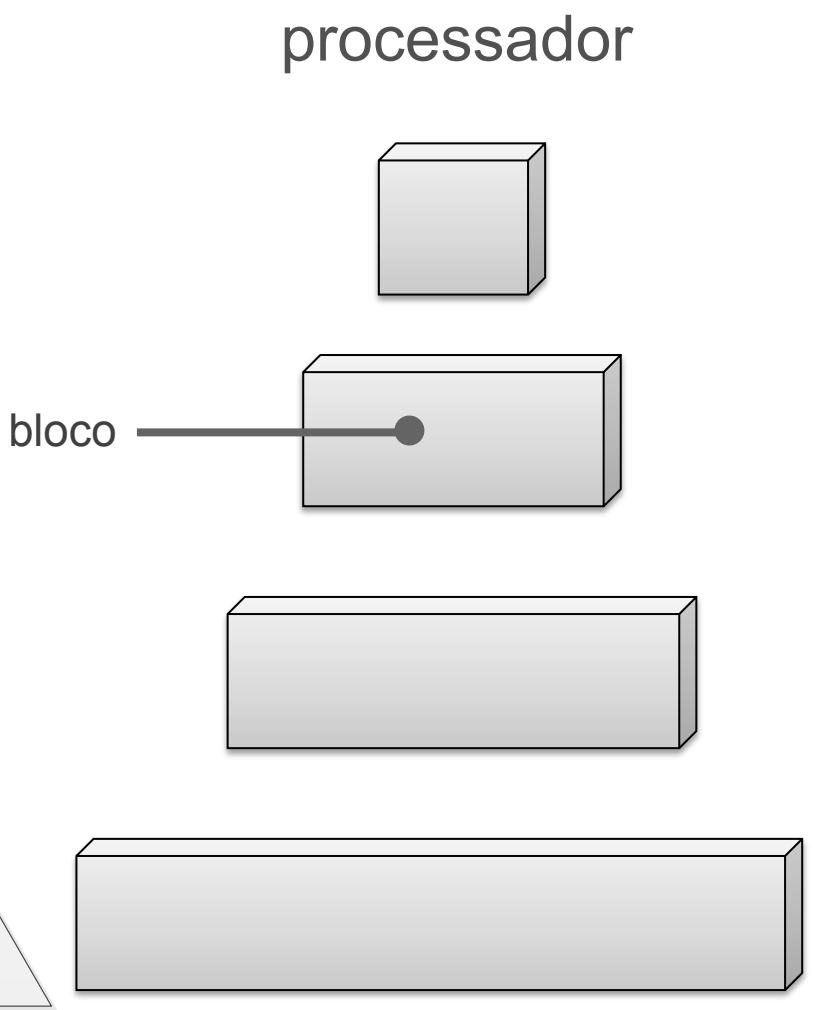
Arquiteturas de Propósito Geral

Em arquiteturas de propósito geral, tradicionalmente toda essa gestão é realizada pelo *hardware*.

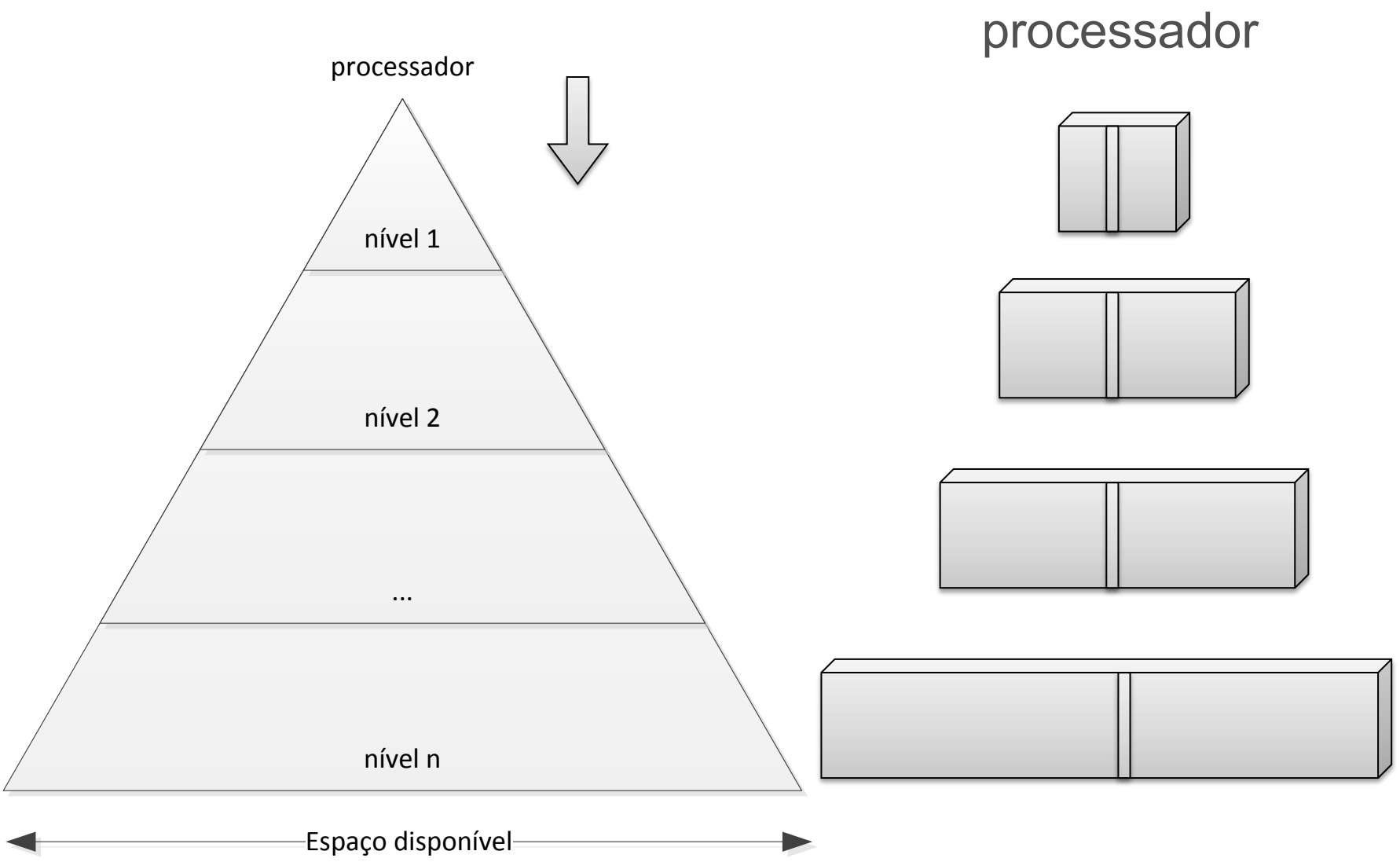
- **Simplifica** o processo de desenvolvimento de software;
- **Eficiente** para o caso geral.

Blocos

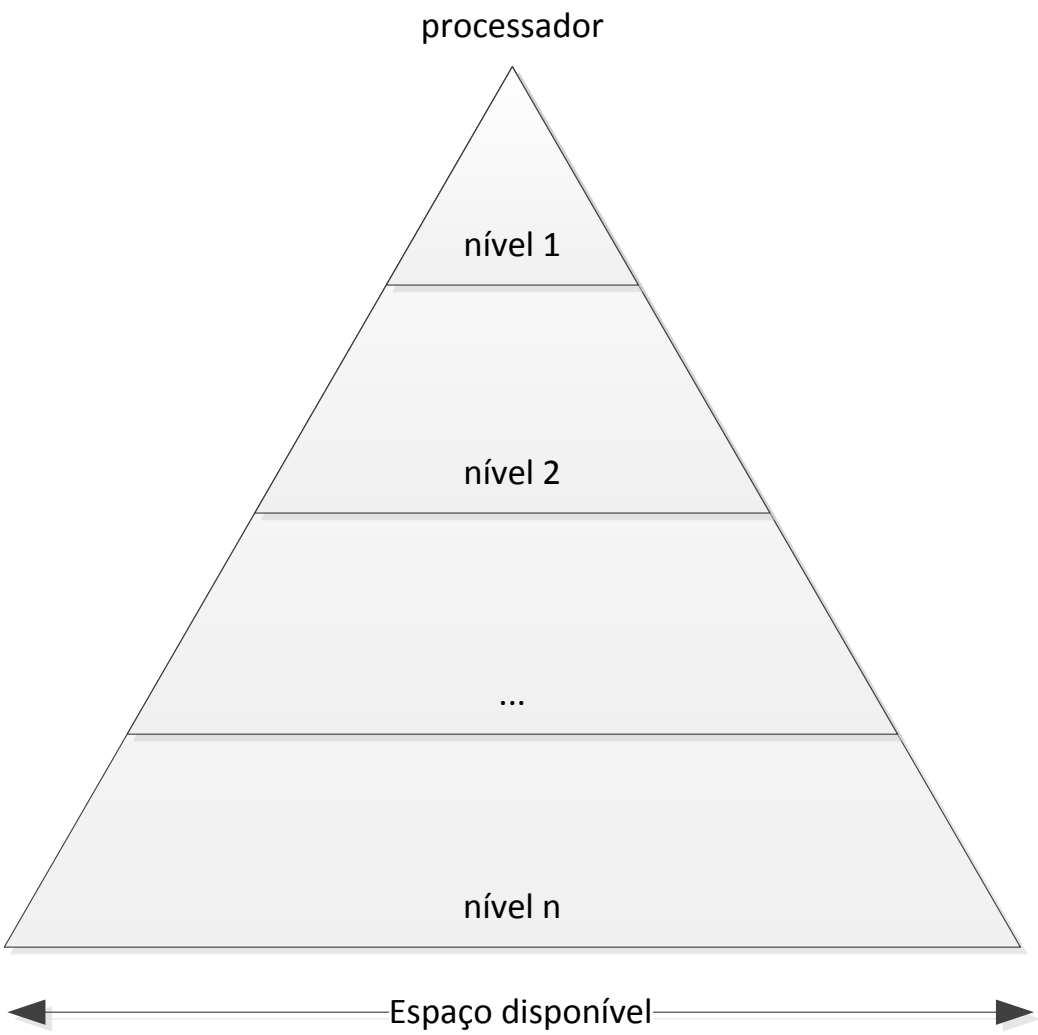

processador
nível 1
nível 2
...
nível n
Espaço disponível

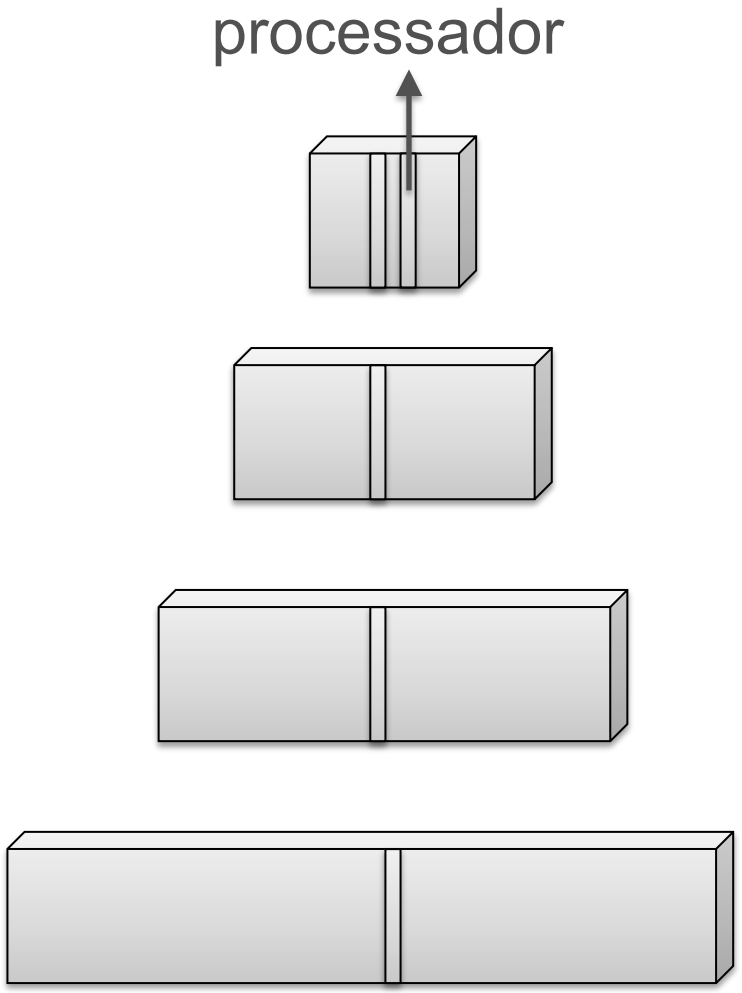

processador
bloco

Solicitando outro dado

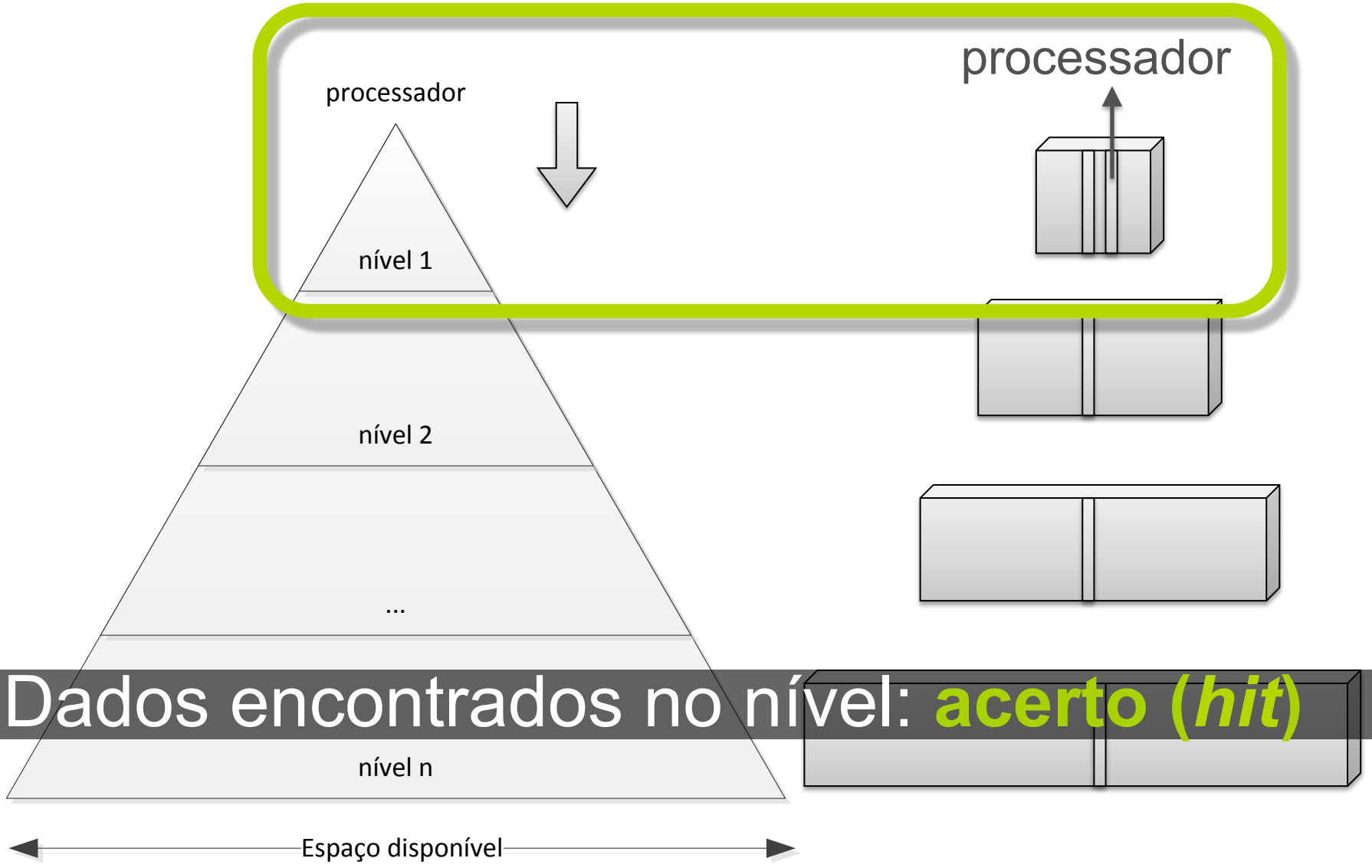


...Solicitando outro dado





...Solicitando outro dado

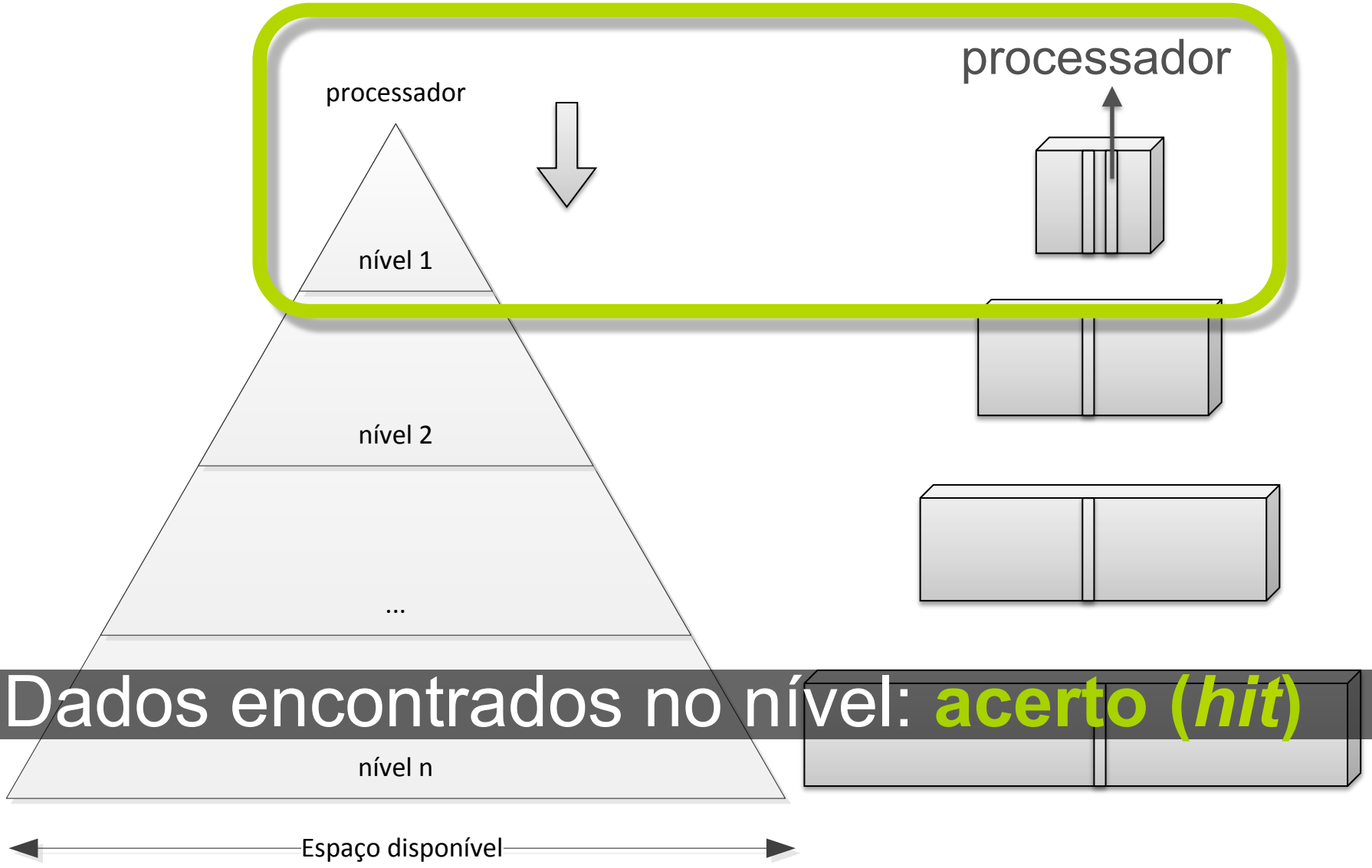


Dados encontrados no nível: **acerto (*hit*)**

...Solicitando outro dado

Dados não encontrados no nível:
falha (*miss*)

...Solicitando outro dado



Dados encontrados no nível: **acerto (*hit*)**

Hierarquia de memória

***Prefetching*:: tentar disponibilizar os blocos nos níveis mais altos antes que eles sejam solicitados.**

Cache

Quando a gestão da hierarquia é feita pelo *hardware*, no nível mais rápido temos a ***cache***.

Hierarquia Gerida por *Hardware*

Baseado nos princípios da localidade:

Localidade Espacial:: os dados próximos a um dado utilizado recentemente serão usados no futuro.

Localidade Temporal:: os dados utilizados em recentemente serão usados no futuro.

...Cache

Mapeamento direto:: um bloco só pode ocupar uma única posição na cache.

Mapeamento por conjunto:: um bloco pode ocupar uma posição de um conjunto de posições da cache.

...Hierarquia Gerida por *Hardware*

Os princípios de localidade são **heurísticas**: sempre levam a uma **solução válida**, mas podem distanciar-se da solução ótima.

Arquiteturas de Propósito Geral

Principais requisitos:

- **eficiência**;
- **generalidade**;
- **retrocompatibilidade**.

Arquiteturas Embarcadas

Os **requisitos** mais importantes costumam **diferir** para **arquiteturas embarcadas**.

Embora as ***caches*** sejam a **"bala de prata"** para **arquiteturas de propósito geral**, elas **não são um consenso em arquiteturas embarcadas**.

...Arquiteturas Embarcadas

Alguns requisitos:

- Minimizar o consumo de energia;
- Reduzir a área *on-chip* ocupada;
- Garantias de tempo real;
- Permitir acessos concorrentes com número menor de conflitos;
- Melhorar a eficiência para casos específicos.

...Arquiteturas Embarcadas

A memória *scratchpad* é uma alternativa que permite acolher parte desses requisitos.

Memória *Scratchpad*

Memória *Scratchpad*:: quando a gestão do nível mais rápido da hierarquia de memória é realizada por *software, chamamos esse nível de* Memória *Scratchpad*.

...Memória *Scratchpad*

Os dados são **alocados e liberados** na memória *on-chip* **explicitamente pelo *software***;

Os dados são **enviados e trazidos** da memória *on-chip* para o próximo nível da hierarquia **explicitamente pelo *software***.

Motivação

Motivação

Algumas aplicações não são bem atendidas pela *cache*.

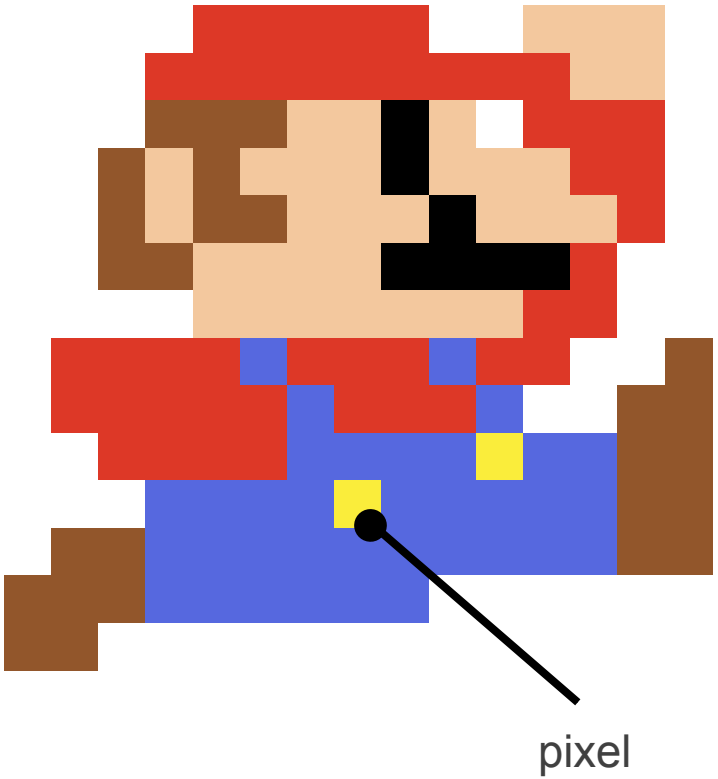
Um padrão de acesso complexo, por exemplo, pode levar a um **desempenho ruim** utilizando as heurísticas de localidade.

...Motivação

Um histograma é um bom exemplo.

Histograma:: é uma representação gráfica em que as medidas ou observações são agrupadas em uma escala horizontal e verticalmente marcam-se as alturas iguais ao número de vezes que cada uma dessas classes ocorre (FREUND, 2004).

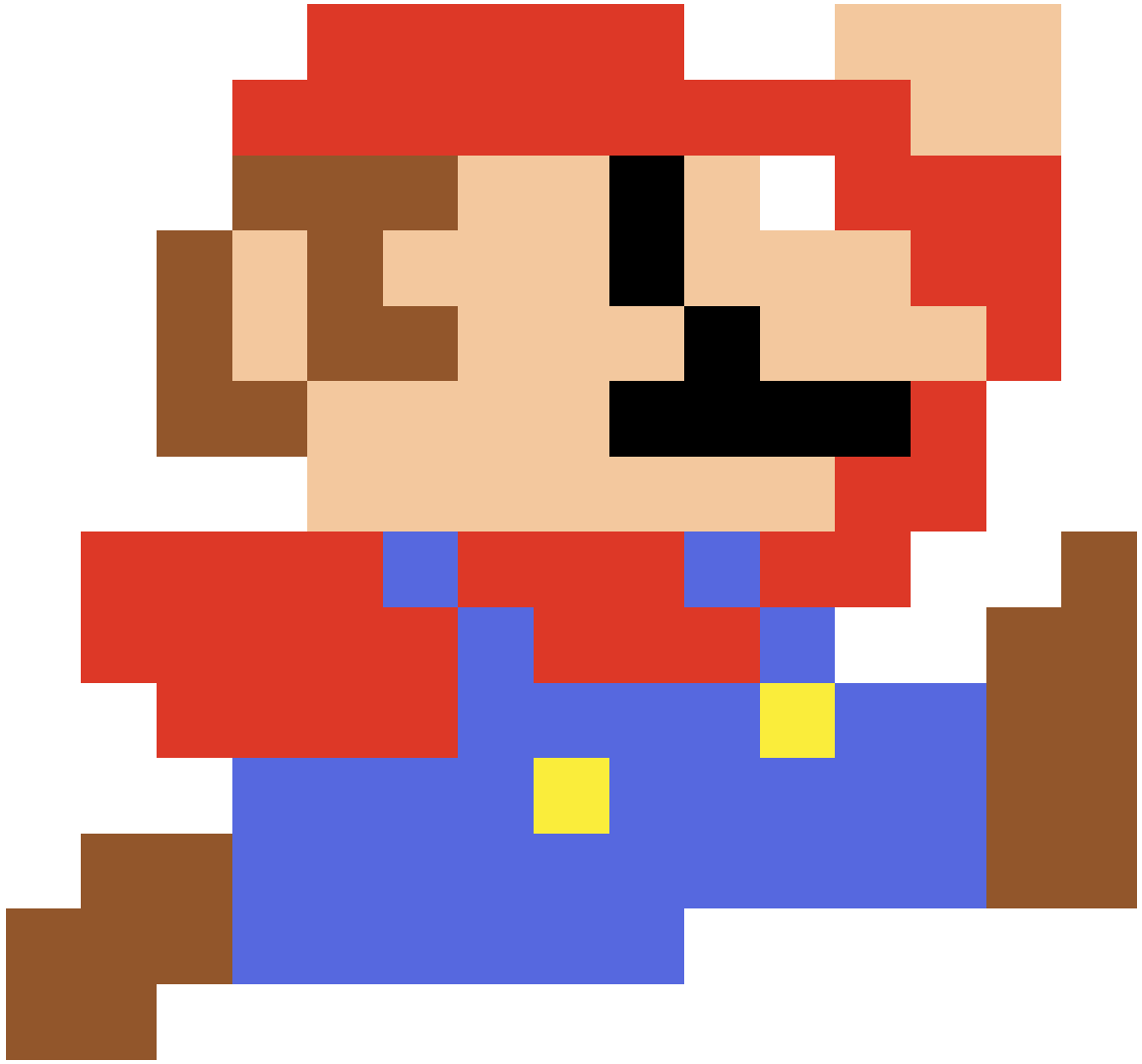
...Motivação



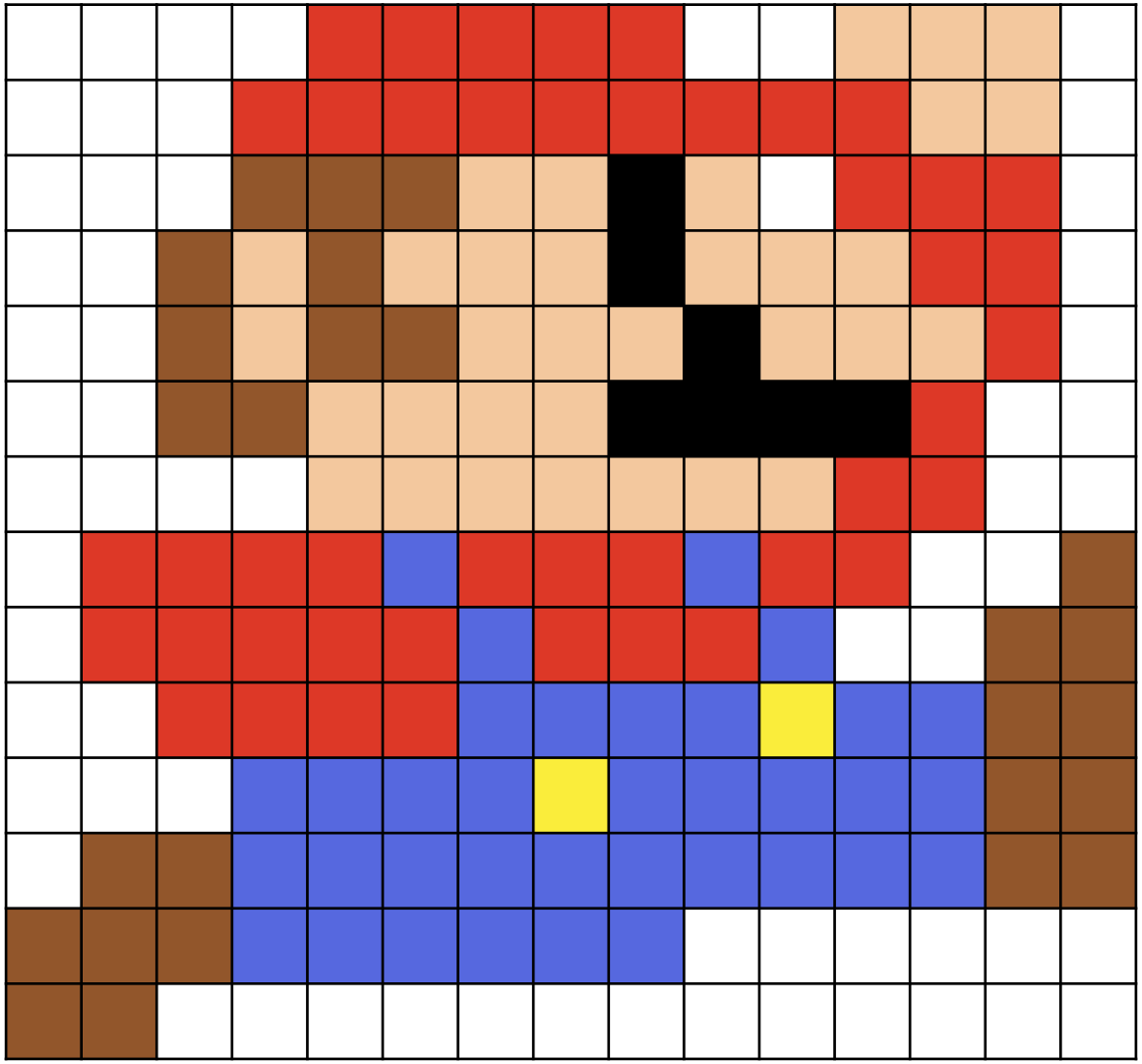
As imagens no computador são compostas por pixels.

Se considerarmos que a cor de cada pixel pode variar de 0 a 255, podemos plotar um histograma.

...Motivação



...Motivação



...Motivação

255	255	255	255	120	120	120	120	120	255	255	190	190	190	255
255	255	255	120	120	120	120	120	120	120	120	120	190	190	255
255	255	255	80	80	80	190	190		190	255	120	120	120	255
255	255	80	190	80	190	190	190		190	190	190	120	120	255
255	255	80	190	80	80	190	190	190		190	190	190	120	255
255	255	80	80	190	190	190	190					120	255	255
255	255	255	255	190	190	190	190	190	190	190	120	120	255	255
255	120	120	120	120	142	120	120	120	142	120	120	255	255	80
255	120	120	120	120	120	142	120	120	120	142	255	255	80	80
255	255	120	120	120	120	142	142	142	142	147	142	142	80	80
255	255	255	142	142	142	142	147	142	142	142	142	142	80	80
255	80	80	142	142	142	142	142	142	142	142	142	142	80	80
80	80	80	142	142	142	142	142	142	255	255	255	255	255	255
80	80	255	255	255	255	255	255	255	255	255	255	255	255	255

...Motivação

255	255	255	255	120	120	120	120	120	255	255	190	190	190	255
255	255	255	120	120	120	120	120	120	120	120	120	190	190	255
255	255	255	80	80	80	190	190	0	190	255	120	120	120	255
255	255	80	190	80	190	190	190	0	190	190	190	120	120	255
255	255	80	190	80	80	190	190	190	0	190	190	190	120	255
255	255	80	80	190	190	190	190	0	0	0	0	120	255	255
255	255	255	255	190	190	190	190	190	190	190	120	120	255	255
255	120	120	120	120	142	120	120	120	142	120	120	255	255	80
255	120	120	120	120	120	142	120	120	120	142	255	255	80	80
255	255	120	120	120	120	142	142	142	142	147	142	142	80	80
255	255	255	142	142	142	142	147	142	142	142	142	142	80	80
255	80	80	142	142	142	142	142	142	142	142	142	142	80	80
80	80	80	142	142	142	142	142	142	255	255	255	255	255	255
80	80	255	255	255	255	255	255	255	255	255	255	255	255	255

...Motivação

Se a matriz *M* representa uma imagem com 16x16 pixels,

$$M_{16,16} = \begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ 67 & 56 & 220 & \dots & 225 & 226 & 49 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

...Motivação

o código a seguir pode ser utilizado para calcular o histograma:

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
   for (int j = 0; j < n; j++){
      p = M[i][j];
      histograma[p] = histograma[p] + 1;
   }
```

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
   for (int j = 0; j < n; j++){
      p = M[i][j];
      histograma[p] = histograma[p] + 1;
   }
```

O valor lido do *pixel* da matriz *M* em *p* é utilizado para indexar o vetor *histograma*.

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
   for (int j = 0; j < n; j++){
      p = M[i][j];
      histograma[p] = histograma[p] + 1;
   }
```

Isso **invialibiliza** que os mecanismos de localidade e *prefetching* da *cache* consigam **antecipar** a necessidade das **posições** do arranjo histograma.

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ 67 & 56 & 220 & \dots & 225 & 226 & 49 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

Primeira iteração...

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ \cdot & 56 & 220 & \dots & 225 & 226 & 49 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

```
p = M[0][0];
histograma[p] = histograma[p] + 1;
```

Primeira iteração...

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ \cdot & 56 & 220 & \dots & 225 & 226 & 49 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

```
p = 254;
histograma[p] = histograma[p] + 1;
```

Primeira iteração...

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ 67 & 56 & 220 & \dots & 225 & 226 & 49 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

```
p = 254;
histograma[254] = histograma[254] + 1;
```

Primeira iteração: a penúltima posição do arranjo histograma é necessária. **Ocorre uma falha,** os dados não estão na cache.

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ 67 & 56 & 220 & \dots & 225 & 226 & 49 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

Segunda iteração...

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ 67 & 56 & 220 & \dots & 225 & 226 & 49 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

```
p = M[0][1];
histograma[p] = histograma[p] + 1;
```

Segunda iteração...

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ 67 & 56 & 220 & \dots & 225 & 226 & 49 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

```
p = 1;
histograma[p] = histograma[p] + 1;
```

Segunda iteração...

...Motivação

```
int histograma[256];
int p;
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++){
        p = M[i][j];
        histograma[p] = histograma[p] + 1;
    }
```

$$\begin{pmatrix} 254 & 1 & 183 & \dots & 122 & 47 & 14 \\ 67 & 56 & 220 & \dots & 225 & 226 & 49 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 10 & 56 & 88 & \dots & 232 & 248 & 184 \\ 210 & 15 & 71 & \dots & 53 & 53 & 147 \\ 87 & 74 & 18 & \dots & 255 & 95 & 214 \end{pmatrix}$$

```
p = 1;
histograma[1] = histograma[1] + 1;
```

Segunda iteração: a segunda posição do arranjo histograma é necessária. **Ocorre uma falha,** os dados não estão na cache.

...Motivação

Como a posição a ser acessada depende dos valores de *M*, é muito **difícil** prever quais posições do vetor serão necessárias.

...Motivação

Se todo o vetor fosse alocado na memória *scratchpad*, **nenhuma falha ocorreria**.

...Motivação

O exemplo explicita um problema possível e não um caso ruim.

Apesar de ilustrativo, a *cache* é uma **boa** abordagem para o problema do histograma de imagens: o **acesso a *M* é previsível** e o vetor ***histograma* é pequeno**.

Um Modelo Transparente de Memória Scratchpad para Arquiteturas de Propósito Geral

Objetivo

Nos casos em que a *cache* não é uma boa opção, é necessário conhecer o **domínio do problema** para fornecer um **acesso eficiente** aos dados.

Solução: gestão da memória por software, memória *scratchpad*. Permite realizar a transferência dos dados conforme a necessidade da aplicação.

Um Modelo Transparente de Memória Scratchpad para Arquiteturas de Propósito Geral

Propor uma arquitetura de propósito geral que ofereça a ***scratchpad*** e a ***cache*** ao mesmo tempo.

A introdução da ***scratchpad*** deve ser **transparente**: **preservar a funcionalidade** do **código legado**, permitindo que **novas aplicações a utilizem quando ela for útil**, podendo **ignorá-la quando ela não o for**.

...Um Modelo Transparente de Memória Scratchpad para Arquiteturas de Propósito Geral

A ideia é definir uma organização que **permita** ao ***software*** explicitamente **alocar**, **liberar** e **transferir** dados para regiões da memória *on-chip*.

Solução proposta

Hardware

Solução Proposta

Adicionar um bit de controle (*S*) a cada posição dos conjuntos da *cache*, para identificar se trata-se de um bloco da memória *scratchpad.*

...Hardware

Blocos assinalados por *S* não são submetidos as políticas de substituição da *cache.*

...Hardware

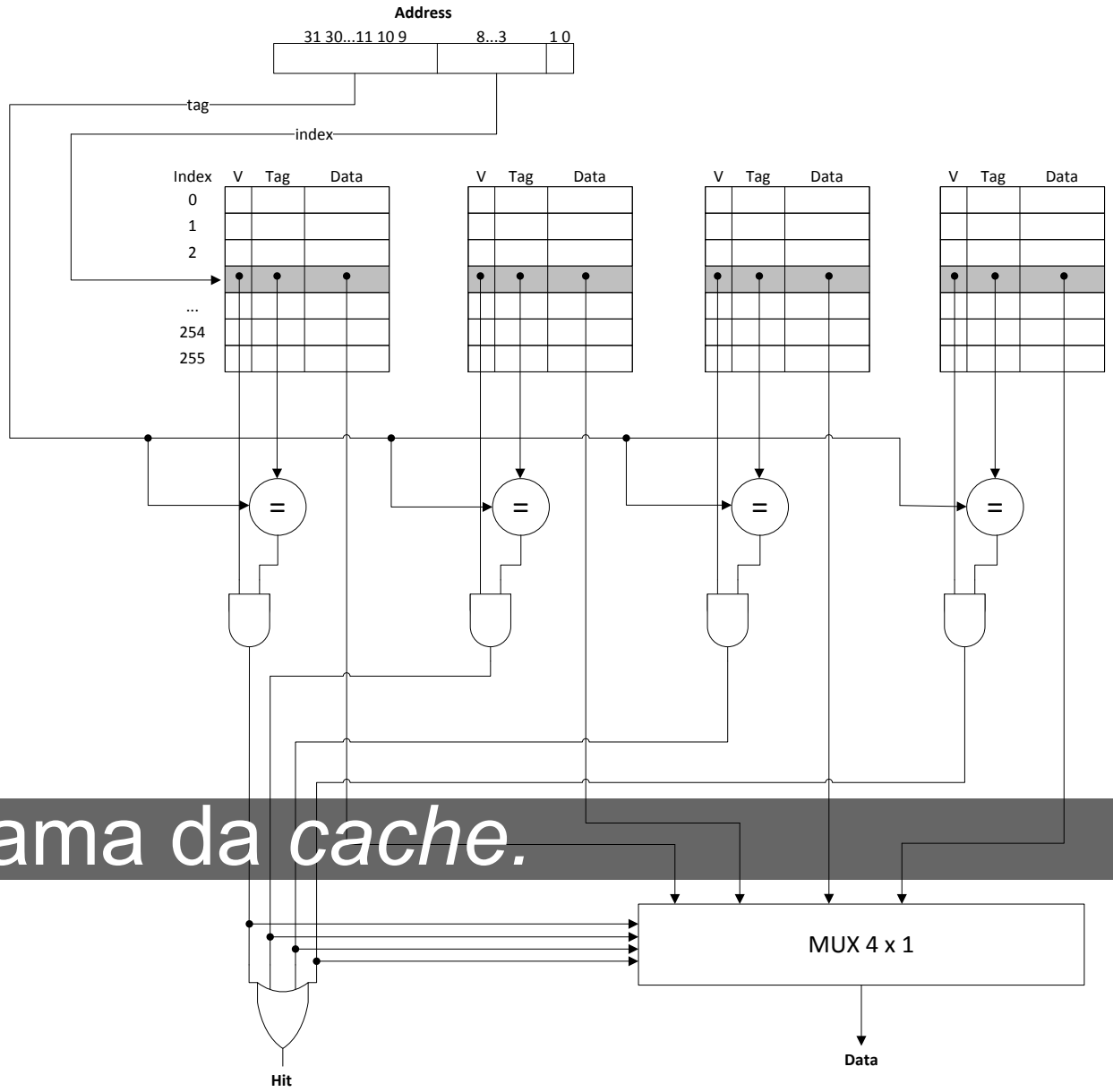


Diagrama da *cache*.

...Hardware

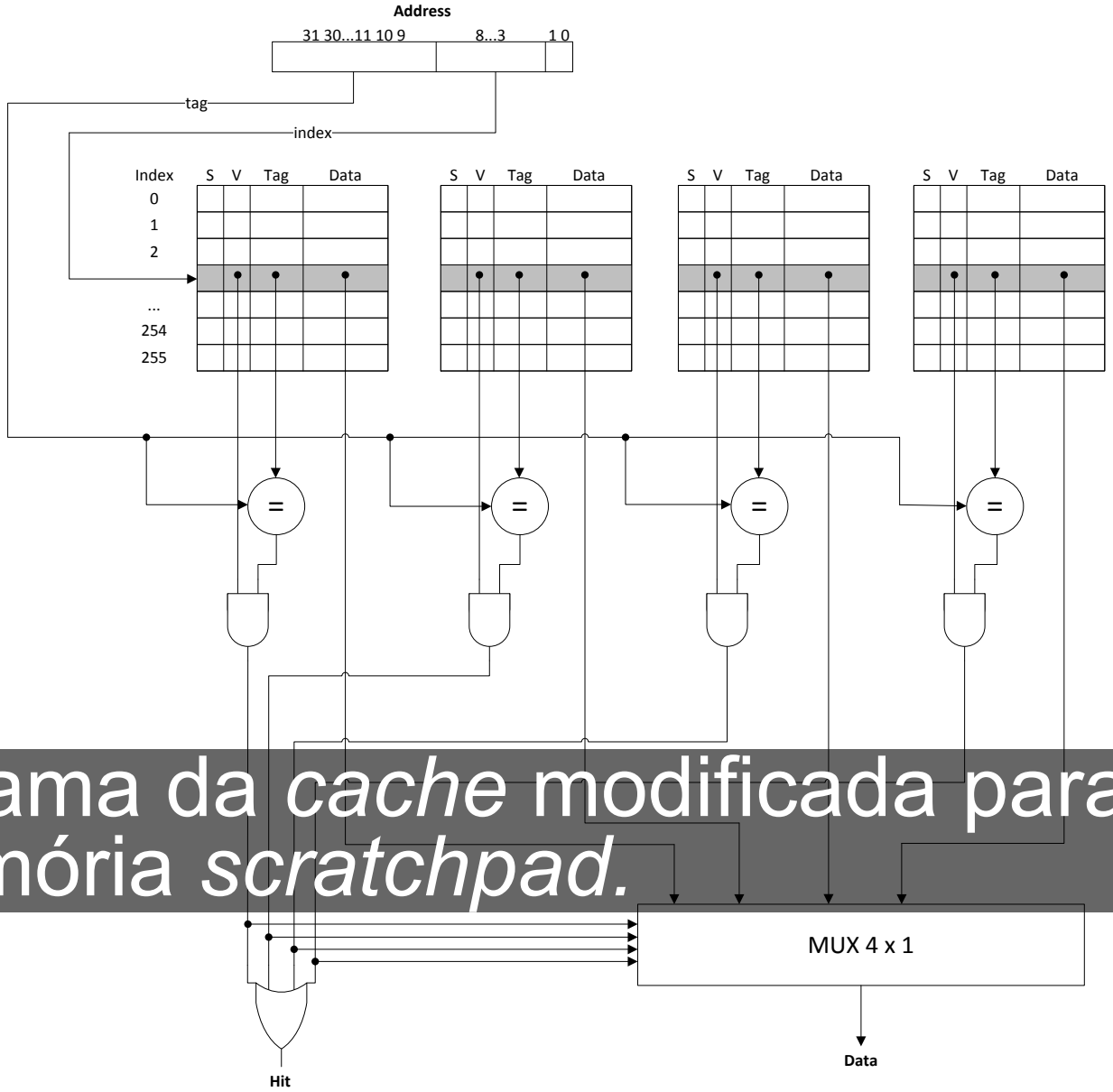
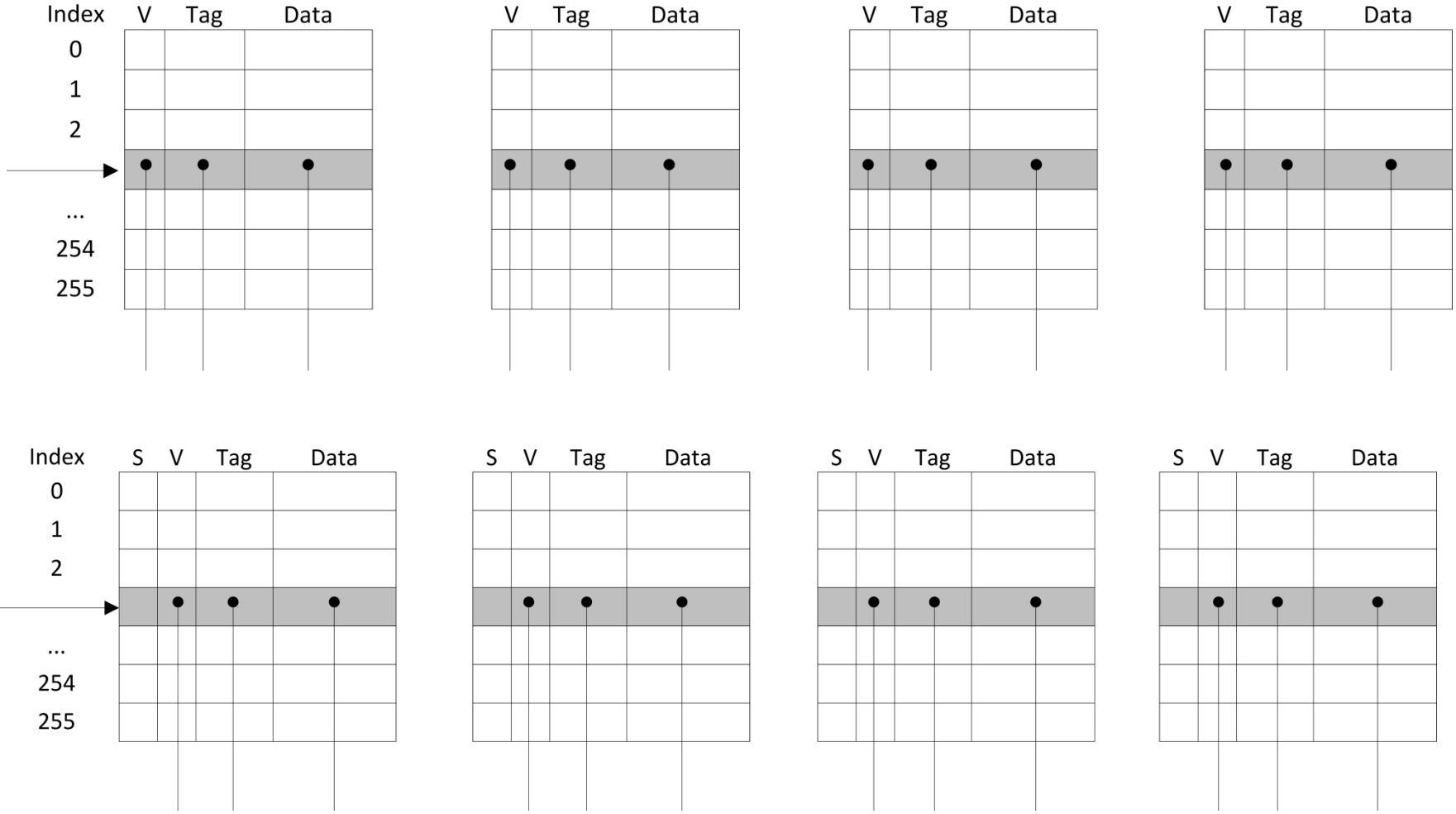


Diagrama da *cache* modificada para incluir a memória *scratchpad*.

...Hardware



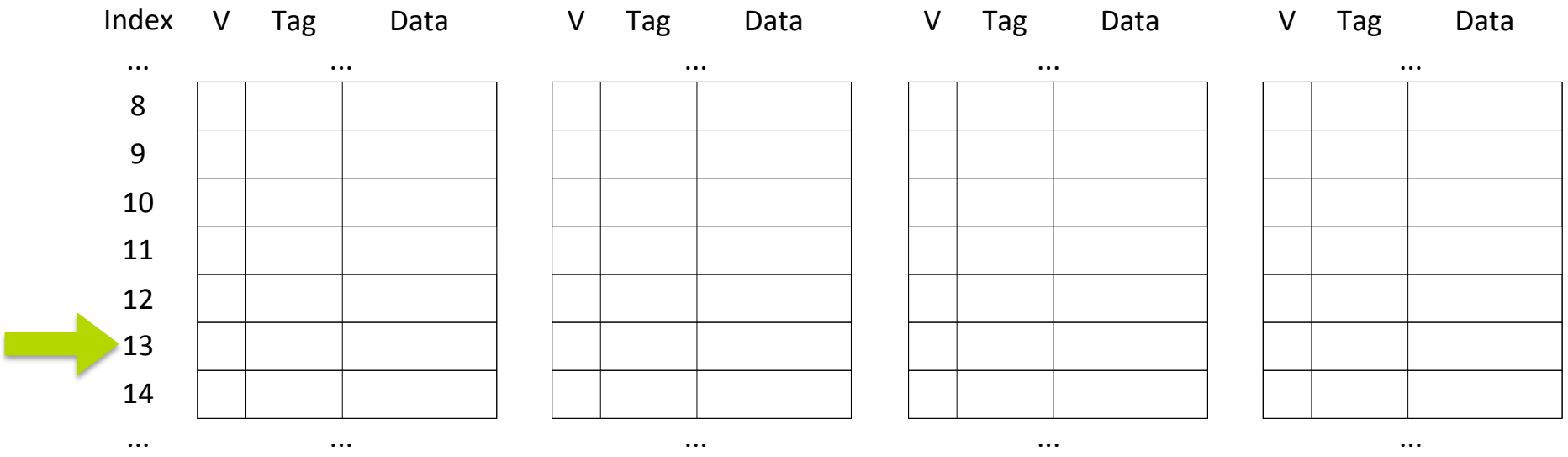
...Hardware

Vantagens

- O funcionamento da *cache* é **preservado**;
- Introduz a *scratchpad* com uma **solução simples**;
- A solução é **transparente** para o código legado;
- Utiliza a **infraestrutura existente**;

- **Perde** as vantagens relacionadas **economia de energia** e **área ocupada**.

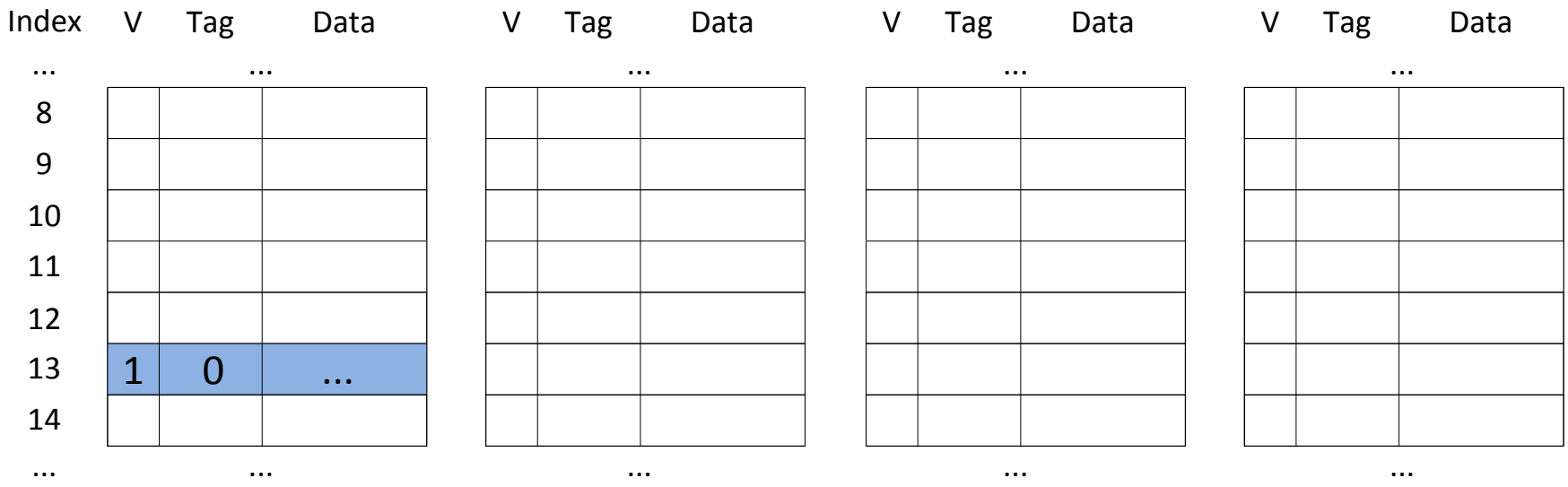
Funcionamento da *cache*



Processador solicita endereço 52

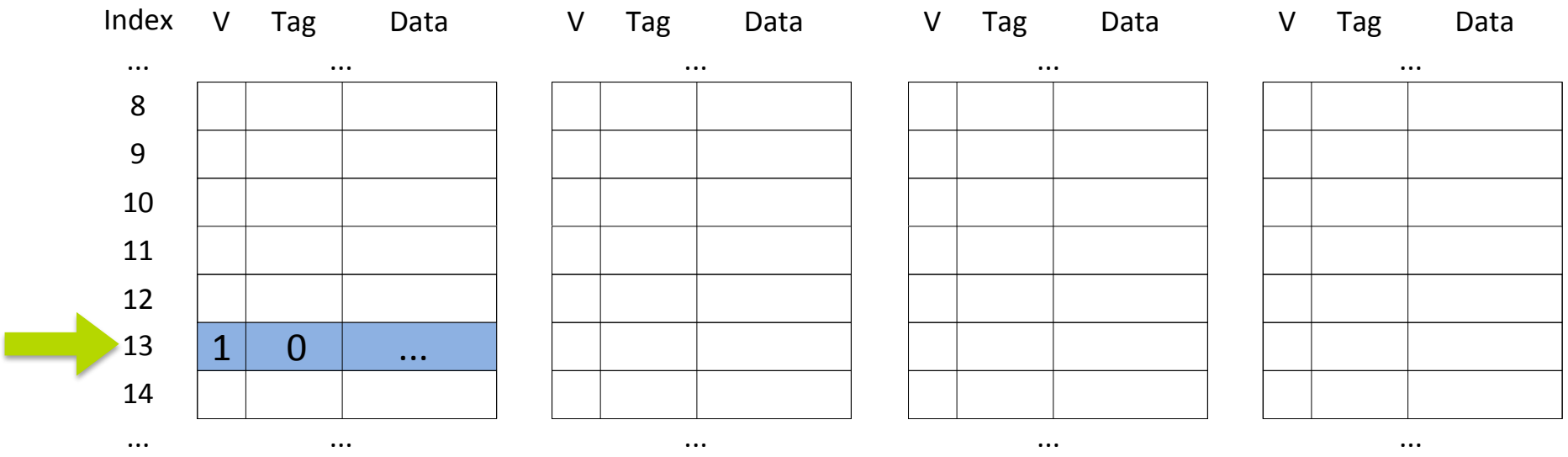
0000 0000 0000 0000 0000 0000 0011 0100

...Funcionamento da *cache*



Todos os elementos estão livres colocar o bloco em um deles...

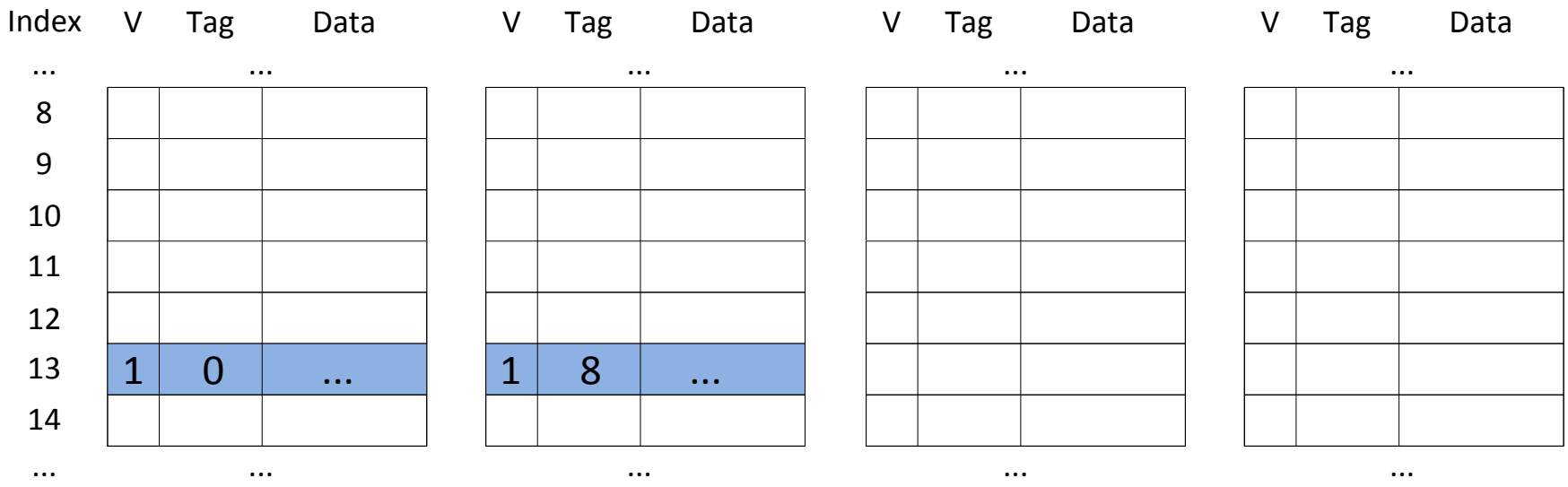
...Funcionamento da *cache*



Processador solicita endereço 2100

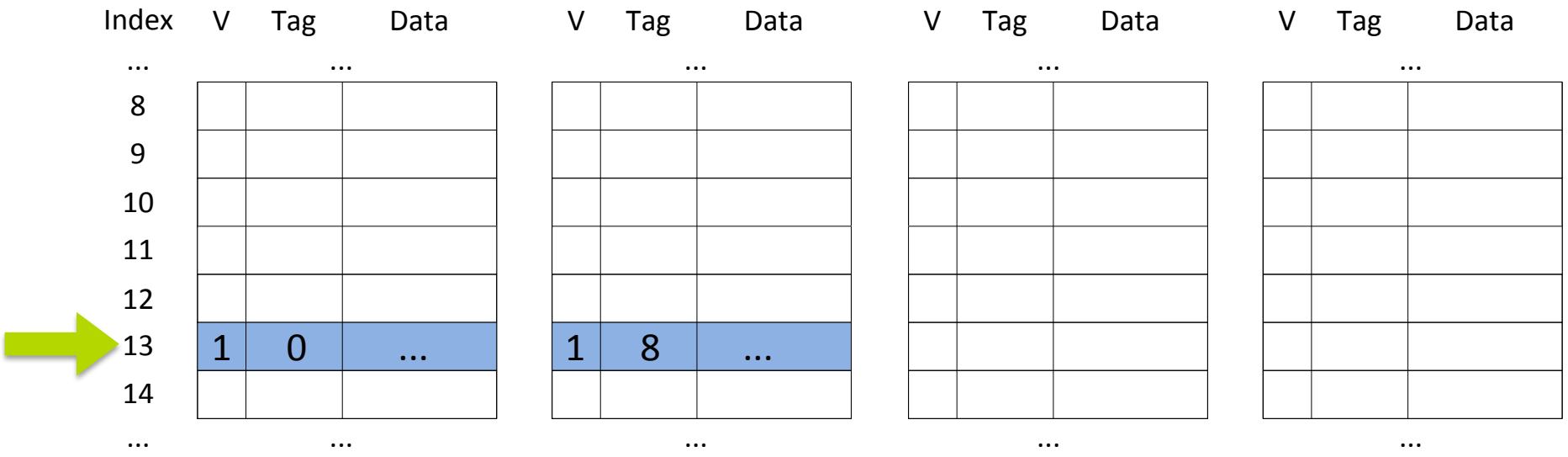
0000 0000 0000 0000 0000 1000 0011 0100

...Funcionamento da *cache*



Ainda há elementos livres, utilizar o próximo...

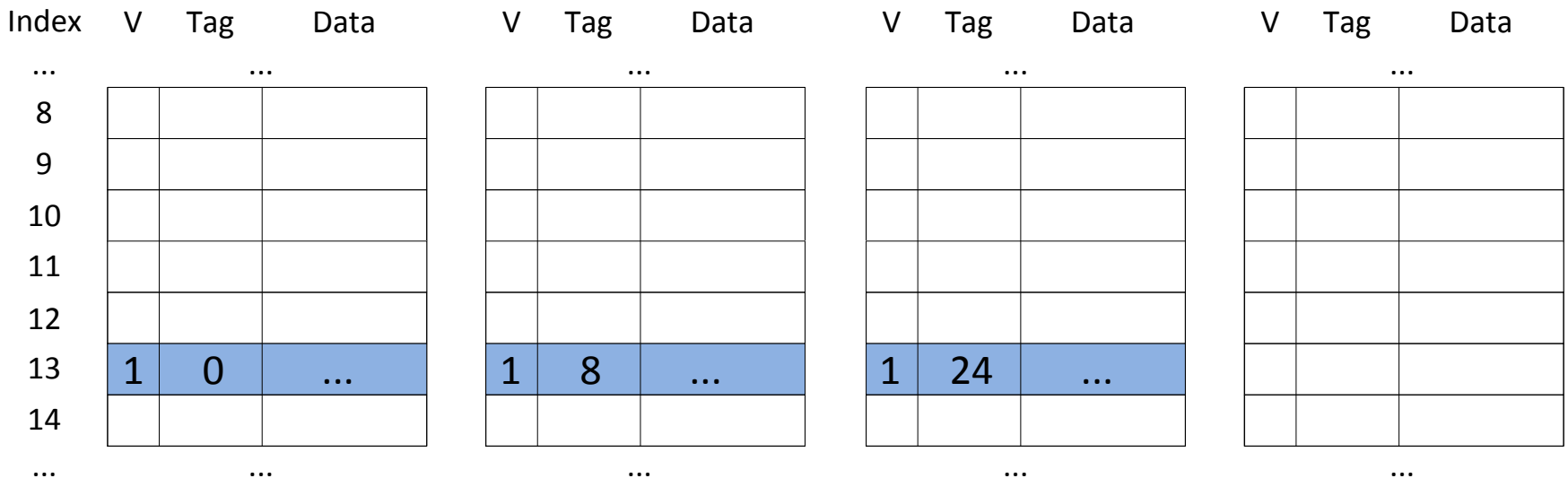
...Funcionamento da *cache*



Processador solicita endereço 6196

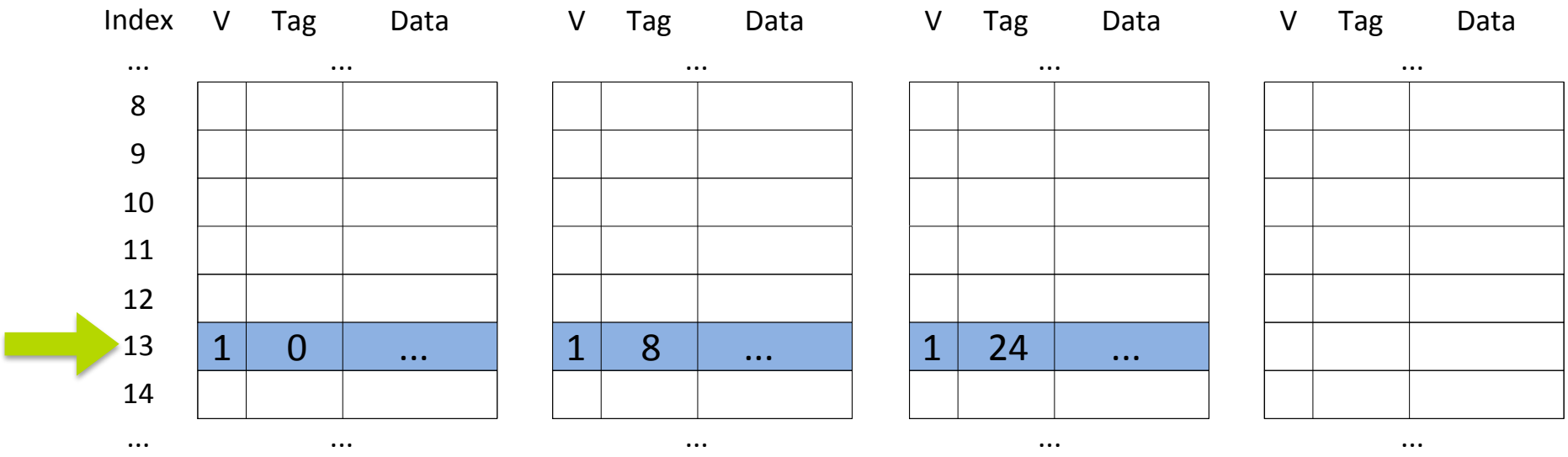
0000 0000 0000 0000 0001 1000 0011 0100

...Funcionamento da *cache*



Ainda há elementos livres, utilizar o próximo...

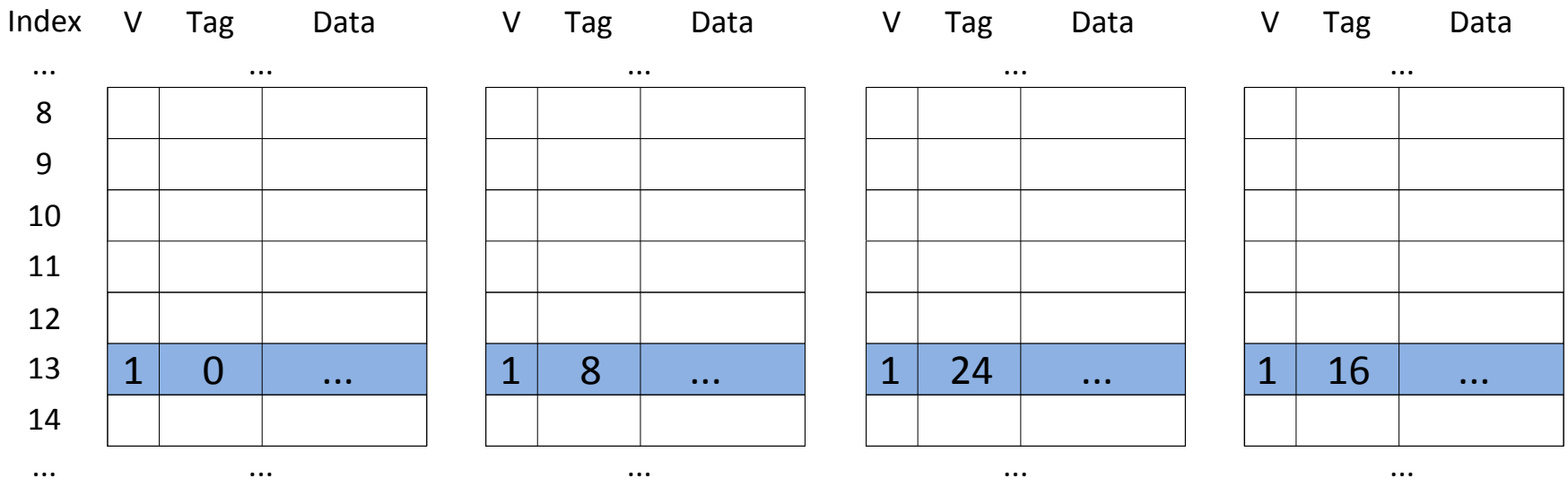
...Funcionamento da *cache*



Processador solicita endereço 4148

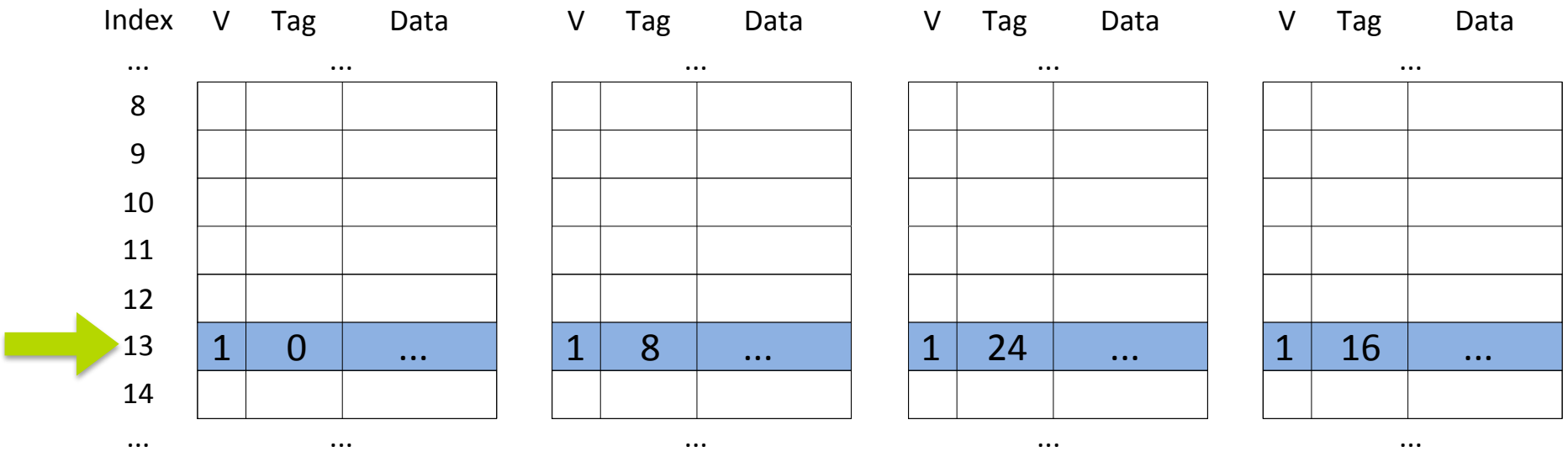
0000 0000 0000 0000 0001 0000 0011 0100

...Funcionamento da *cache*



Ainda há um elemento livre, utilizá-lo...

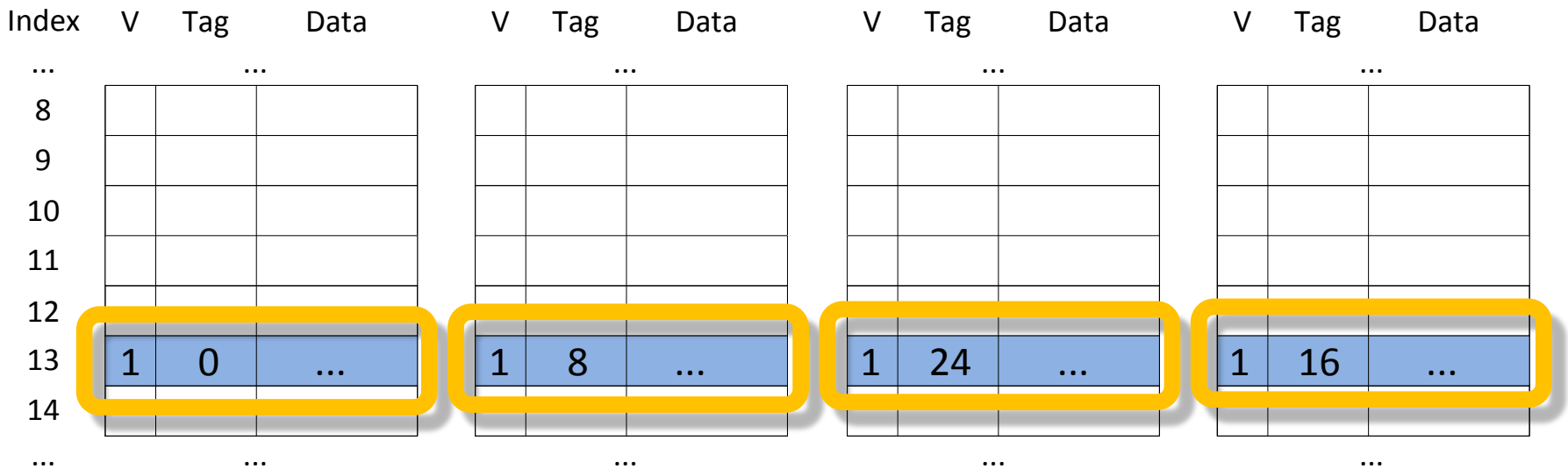
...Funcionamento da *cache*



Processador solicita endereço 14388

0000 0000 0000 0000 0011 1000 0011 0100

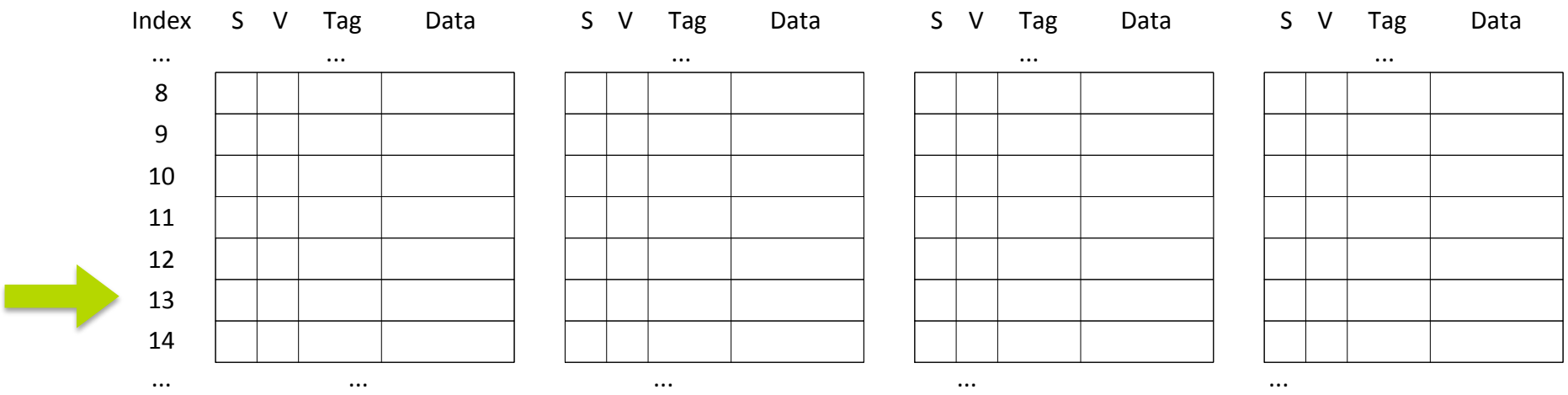
...Funcionamento da *cache*



Conflito, algum bloco terá que ser derramado.

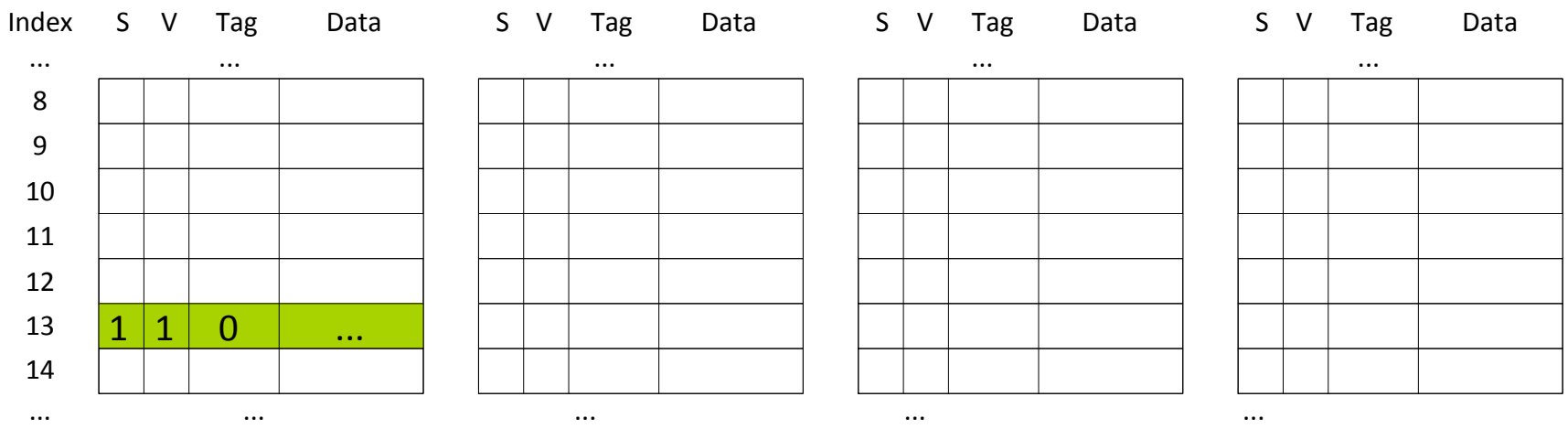
Os quatro elementos são candidatos para a política de substituição.

Funcionamento da *scratchpad*



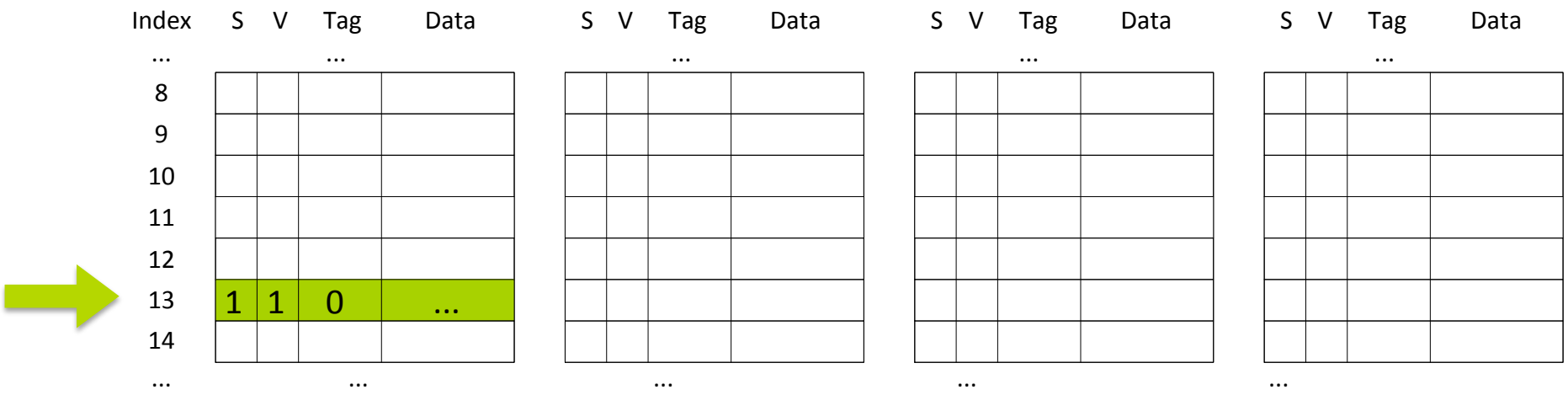
Suponha que o processador aloque o endereço 52 como ***scratchpad***.
0000 0000 0000 0000 0000 0000 0011 0100

...Funcionamento da *scratchpad*



Um elemento do conjunto de índice 13 é assinalado como *scratchpad*.

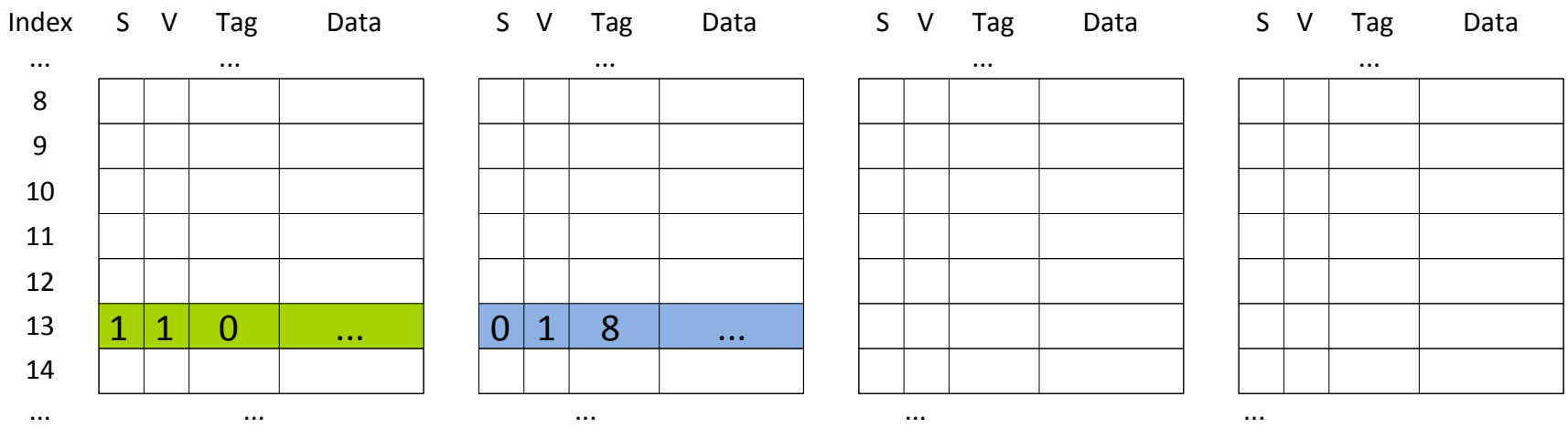
...Funcionamento da *scratchpad*



Processador solicita endereço 2100

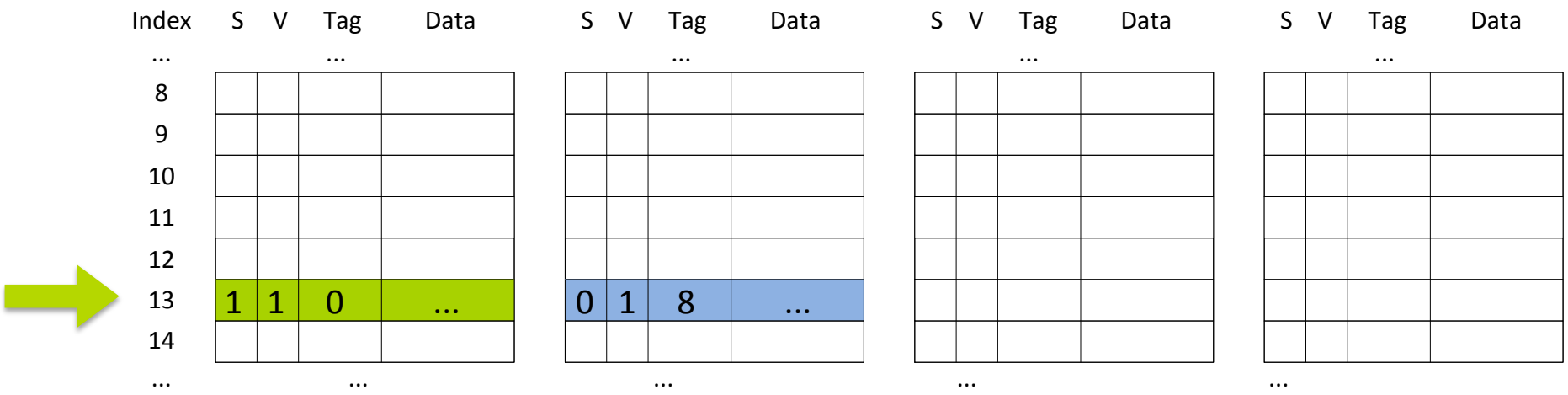
0000 0000 0000 0000 1000 0011 0100

...Funcionamento da *scratchpad*



Utilizar um dos elementos livres...

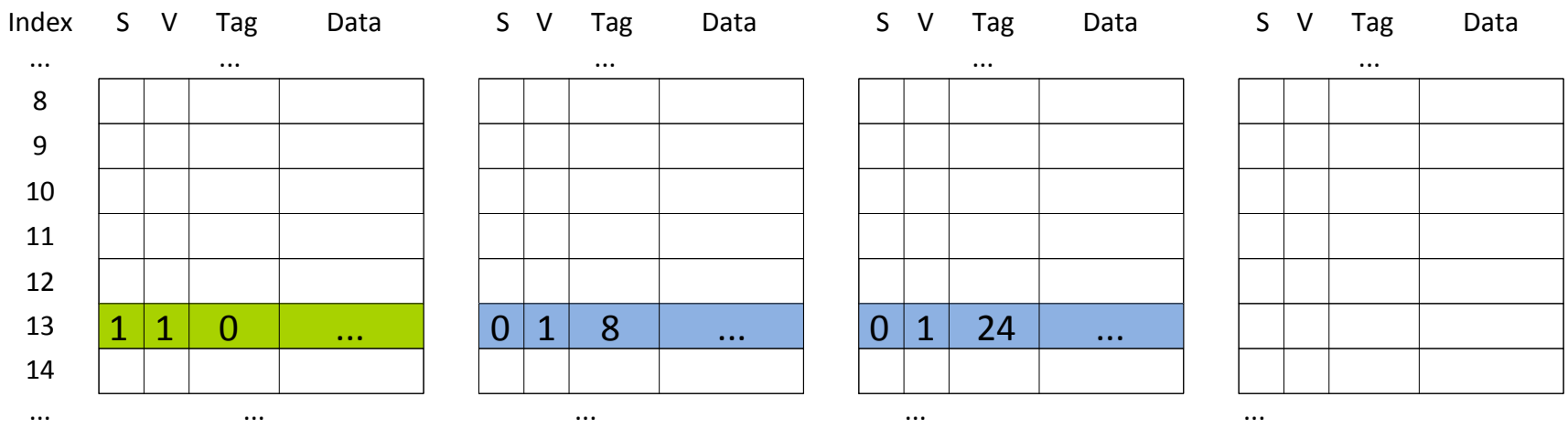
...Funcionamento da *scratchpad*



Processador solicita endereço 6196

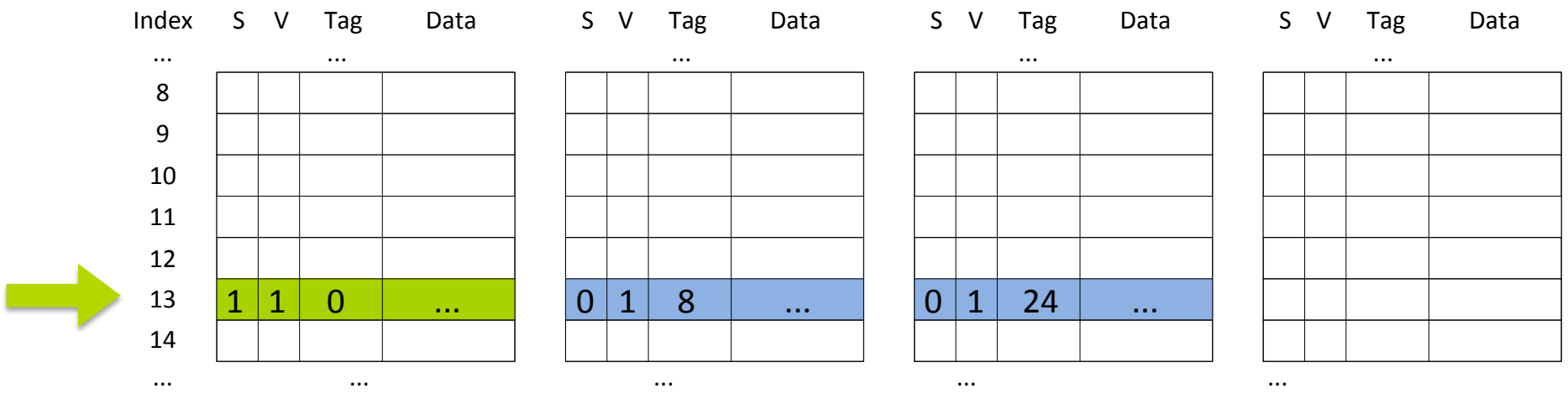
0000 0000 0000 0000 0001 1000 0011 0100

...Funcionamento da *scratchpad*



Utilizar um dos elementos livres...

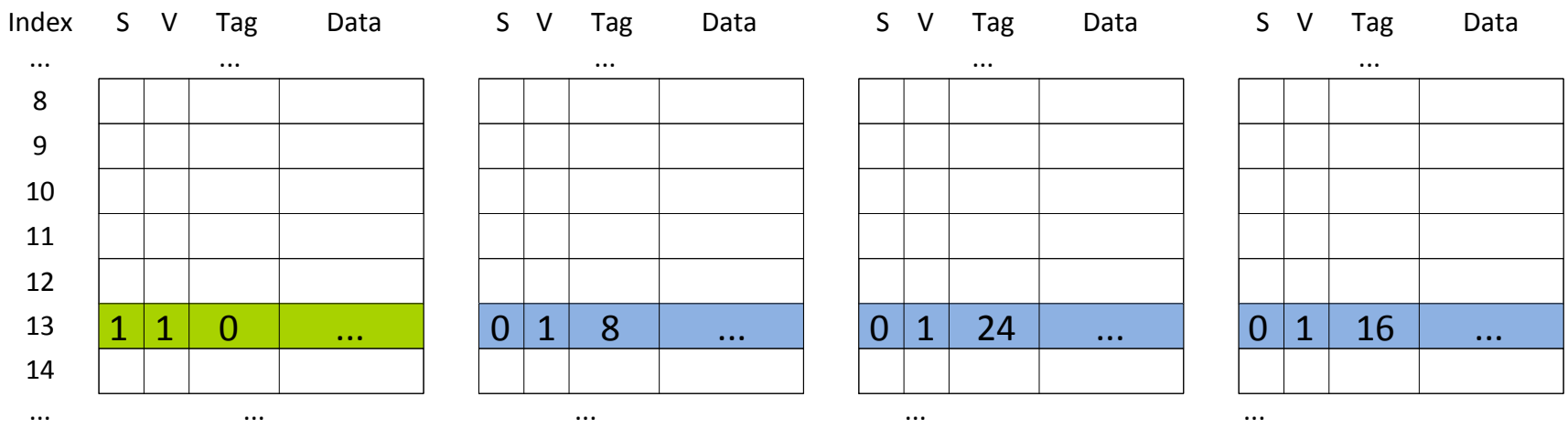
...Funcionamento da *scratchpad*



Processador solicita endereço 4148

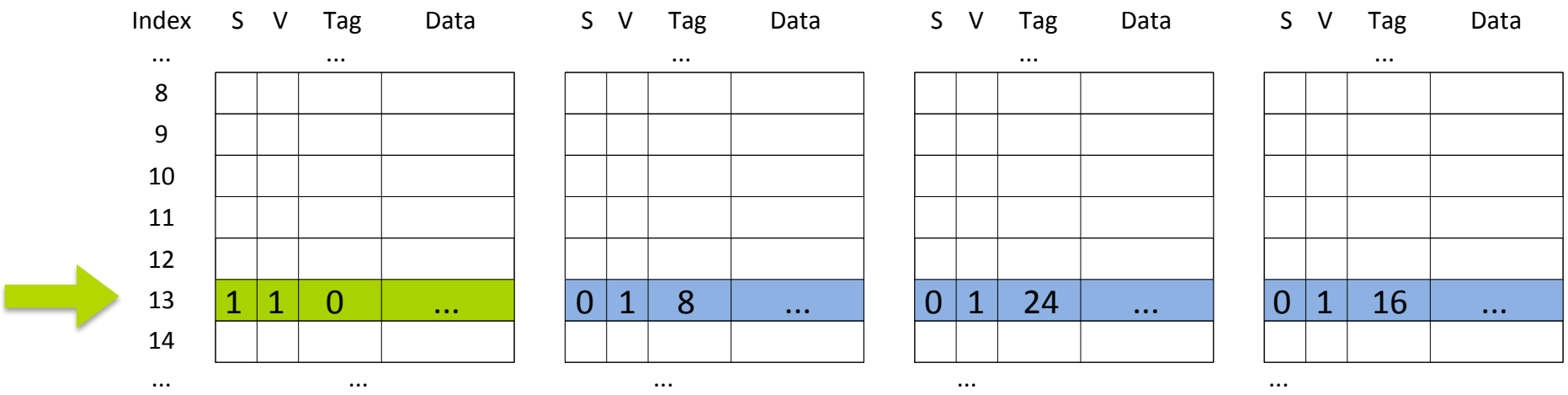
0000 0000 0000 0000 0001 0000 0011 0100

...Funcionamento da *scratchpad*



Utilizar o último elemento livre...

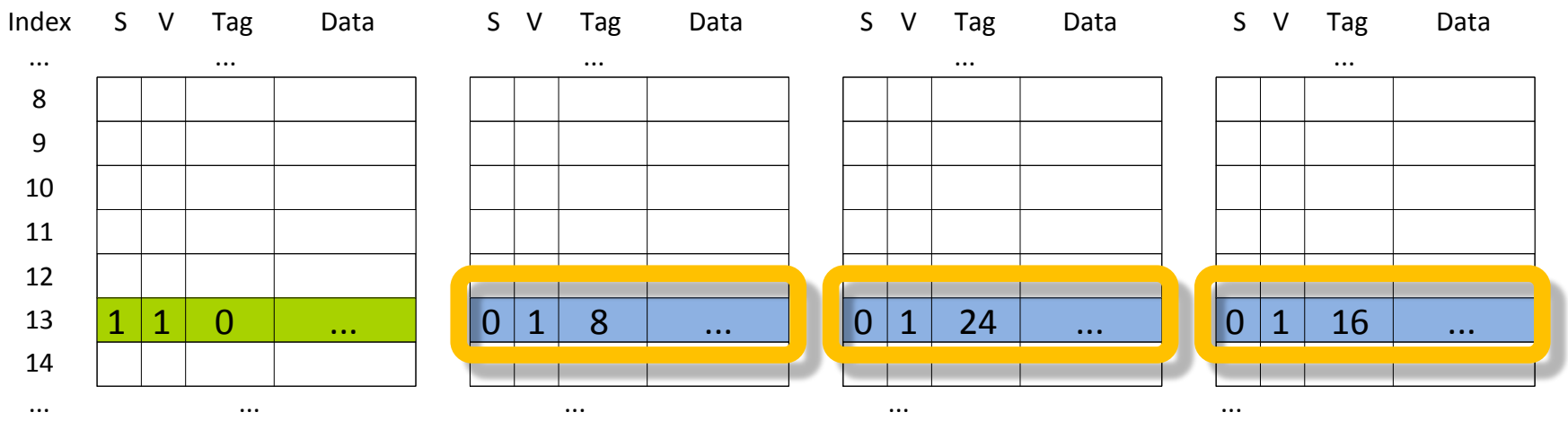
...Funcionamento da *scratchpad*



Processador solicita endereço 14388

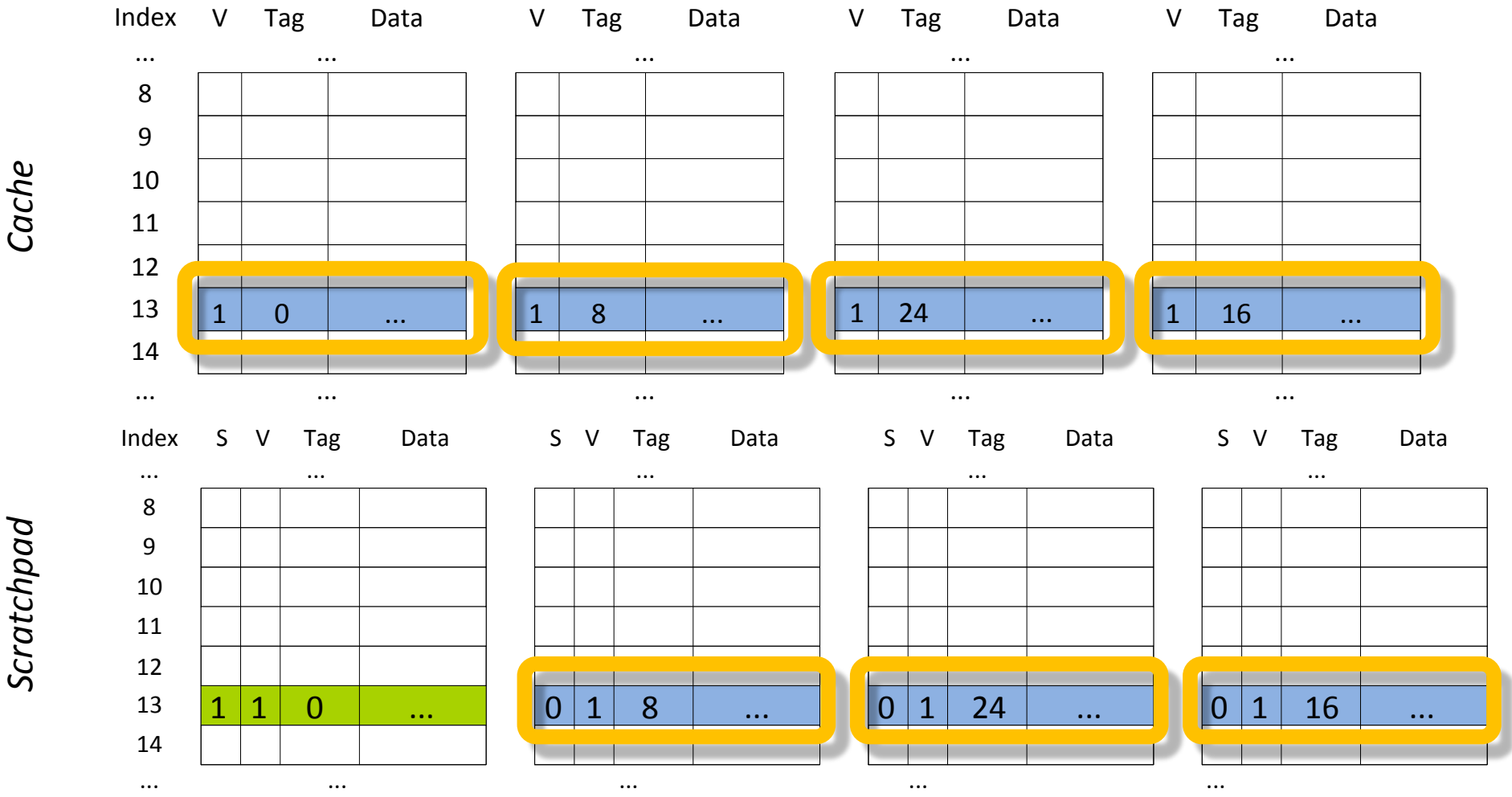
0000 0000 0000 0000 0011 1000 0011 0100

...Funcionamento da *scratchpad*



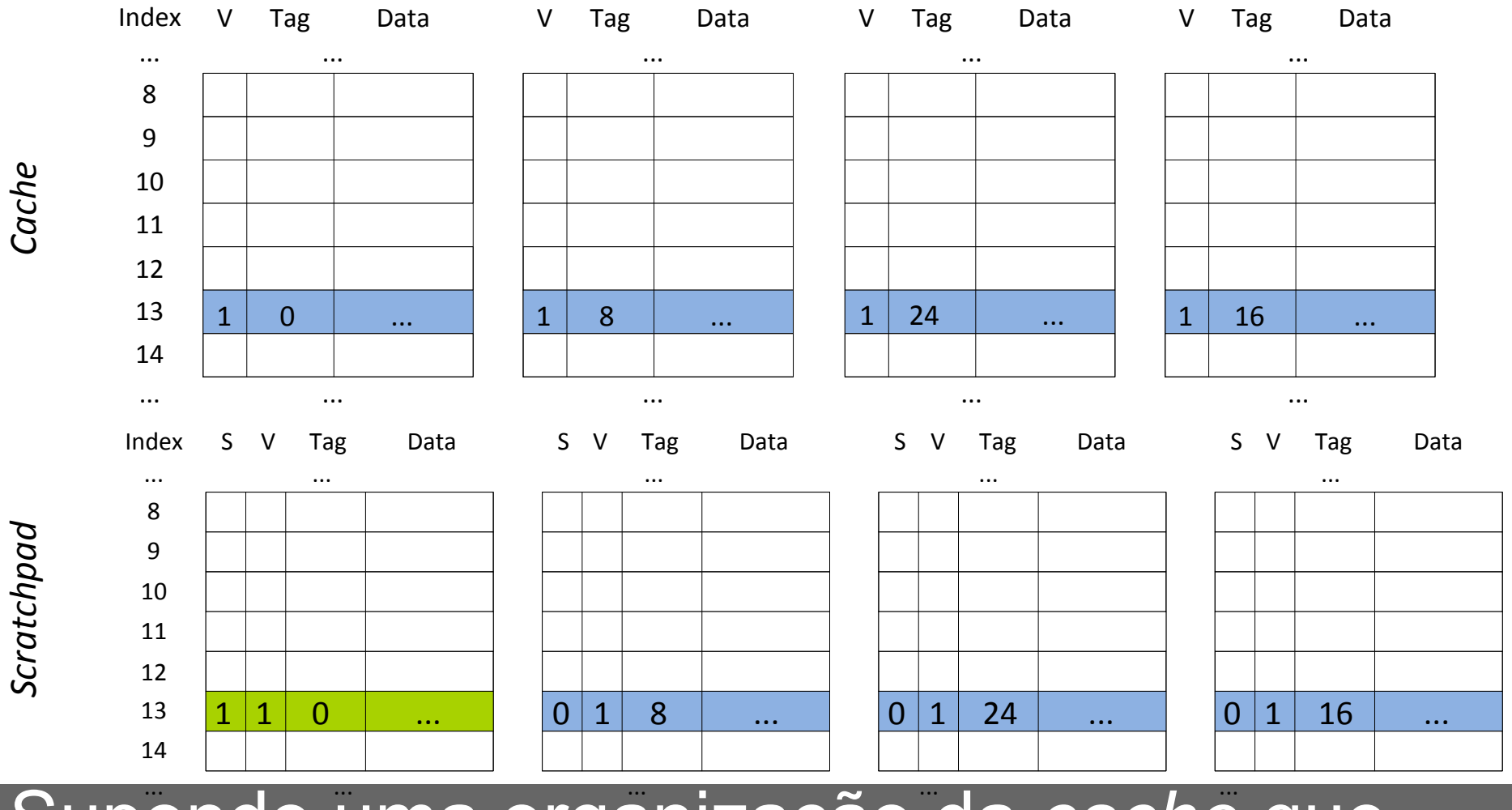
Conflito.
Somente os elementos que **não** são *scratchpad* devem ser candidatos para a política de substituição.

...Funcionamento da *scratchpad*



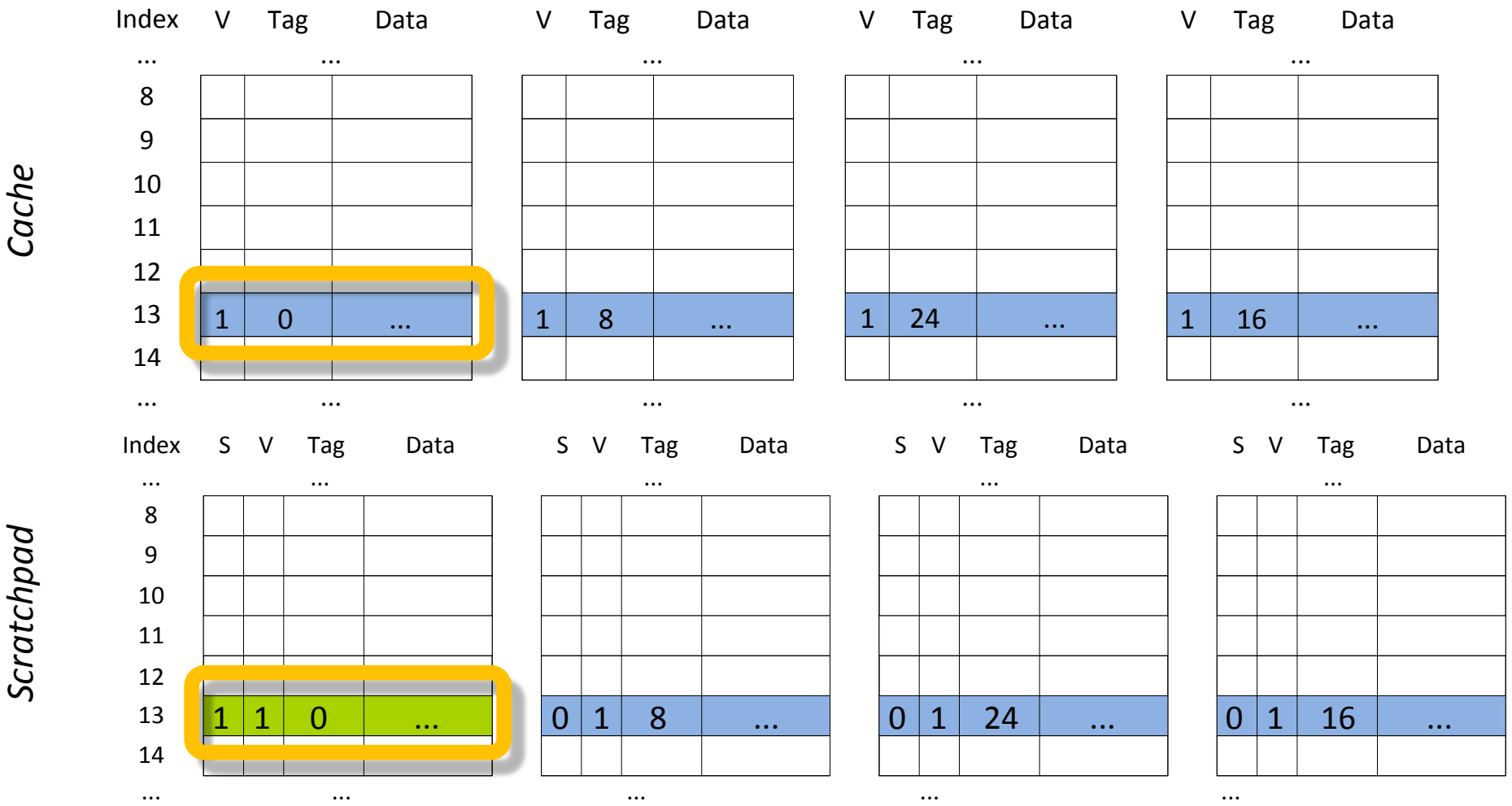
Candidatos a substituição

...Funcionamento da *scratchpad*



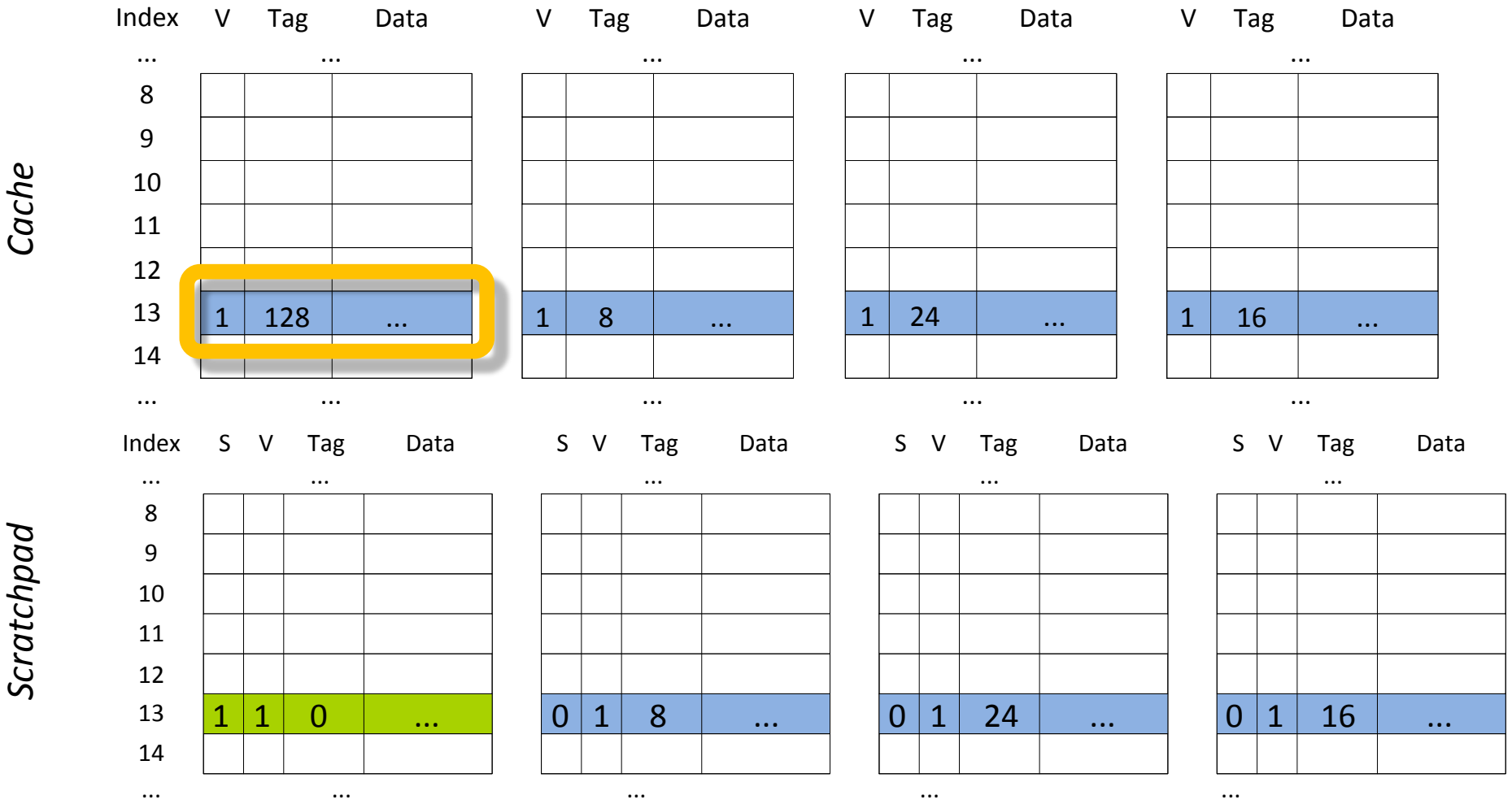
Supondo uma organização da *cache* que derrama os elementos menos recentemente utilizados...

...Funcionamento da *scratchpad*



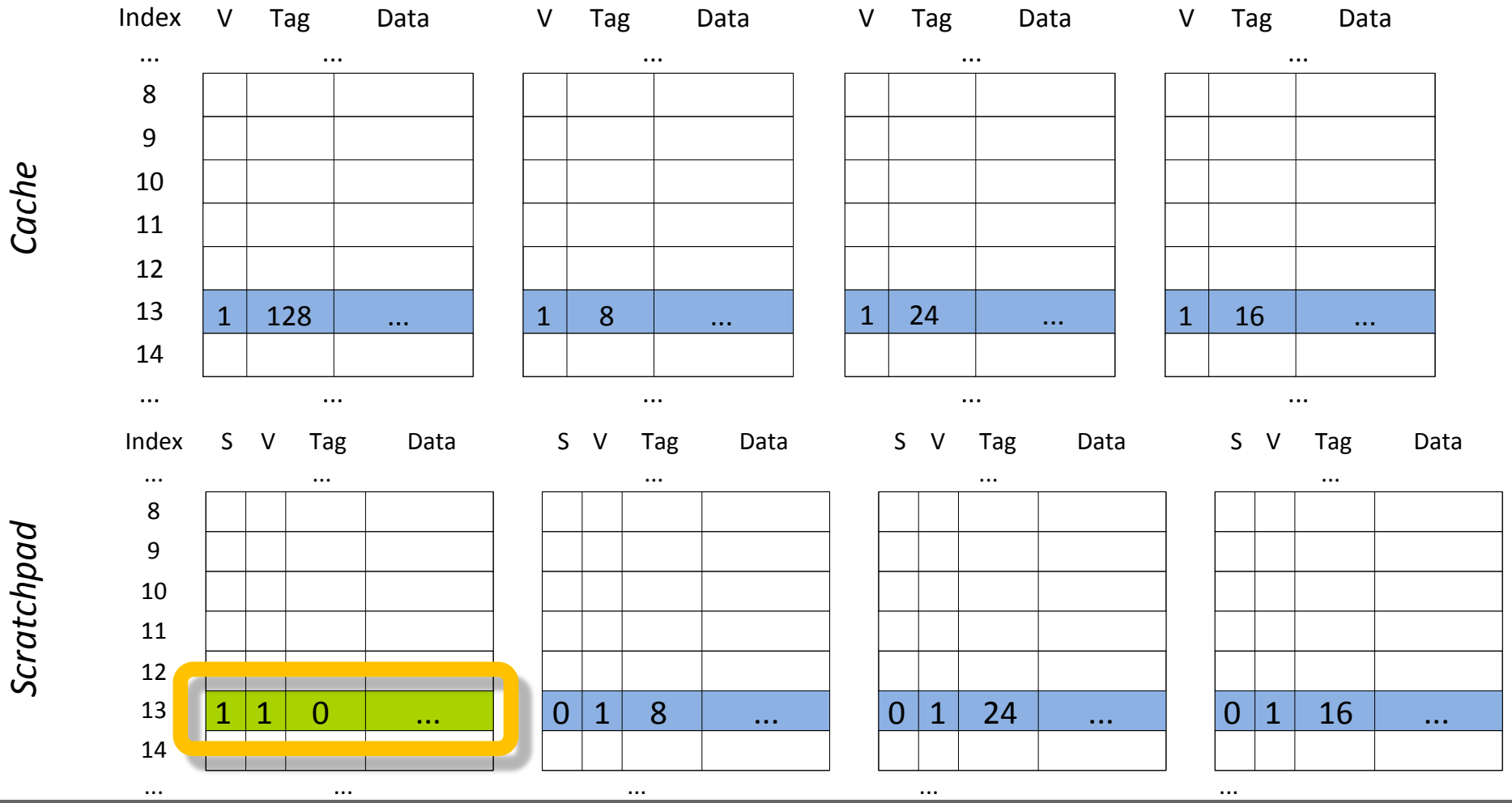
Se os elementos assinalados são os **menos recentemente** utilizados, teríamos...

...Funcionamento da *scratchpad*



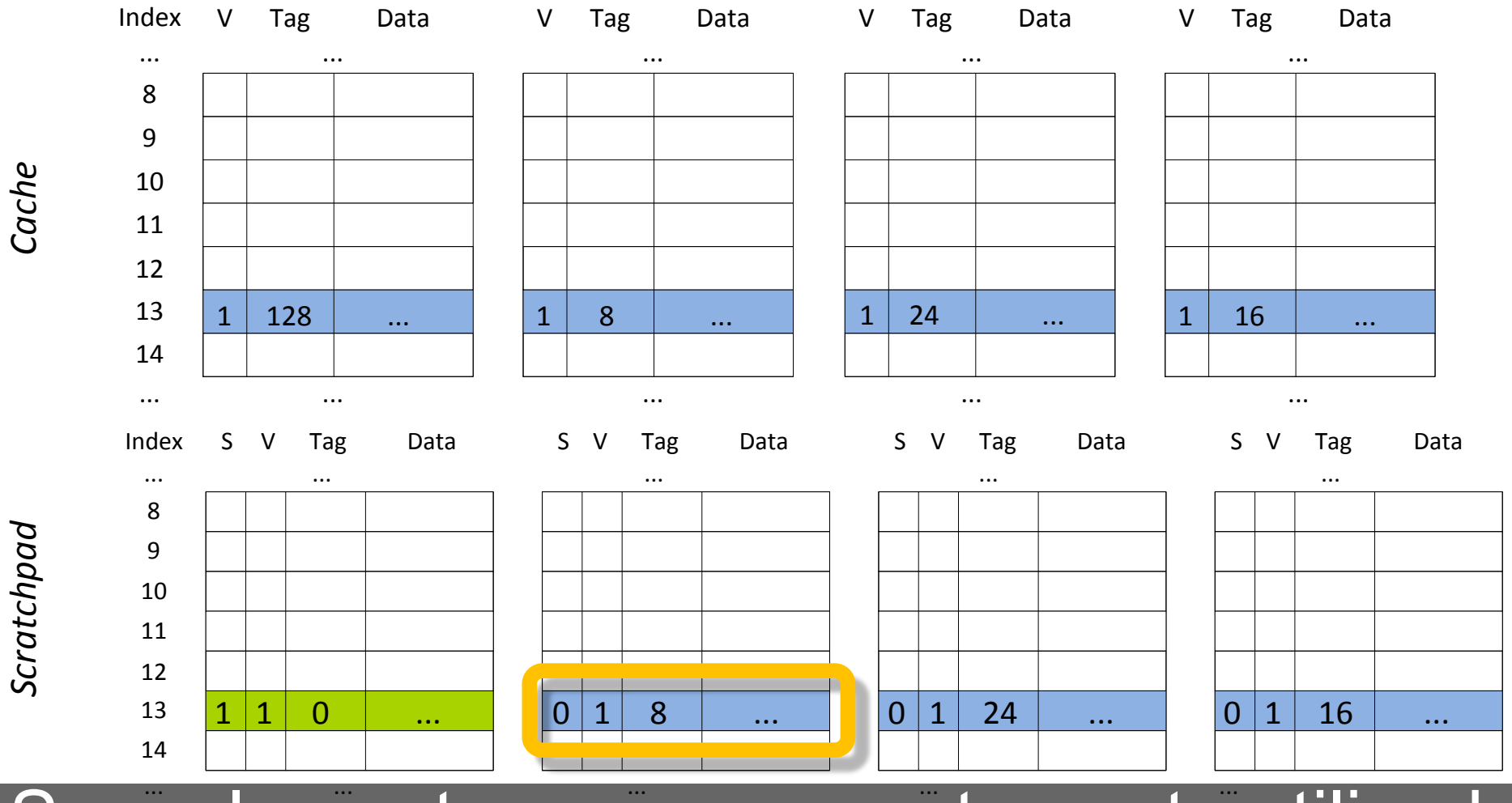
A *cache* substituiria o elemento...

...Funcionamento da *scratchpad*



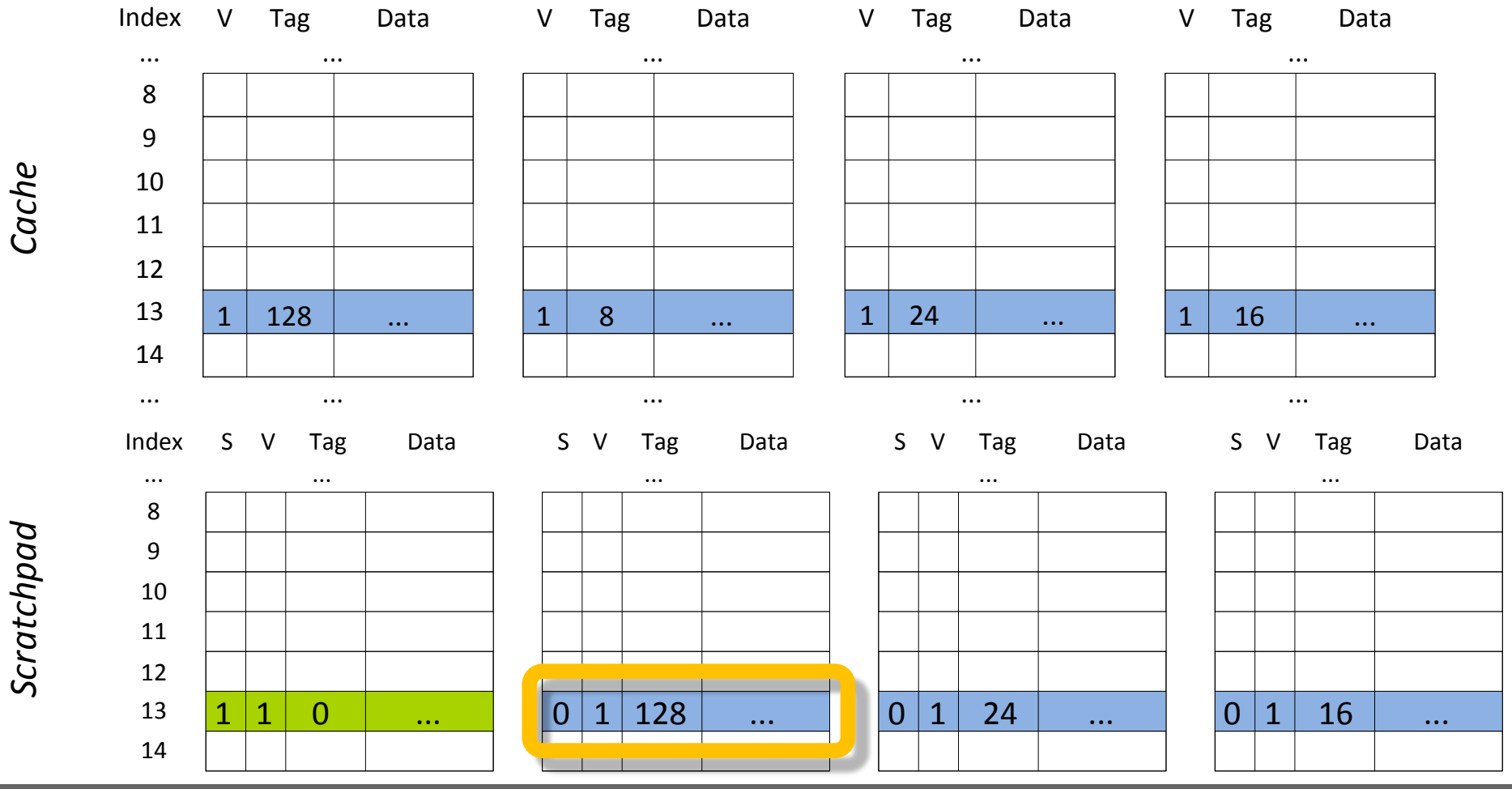
O menos recentemente utilizado compõe a *scratchpad*, nesse caso a *cache* precisa eleger outro candidato preservando sua política de substituição...

...Funcionamento da *scratchpad*



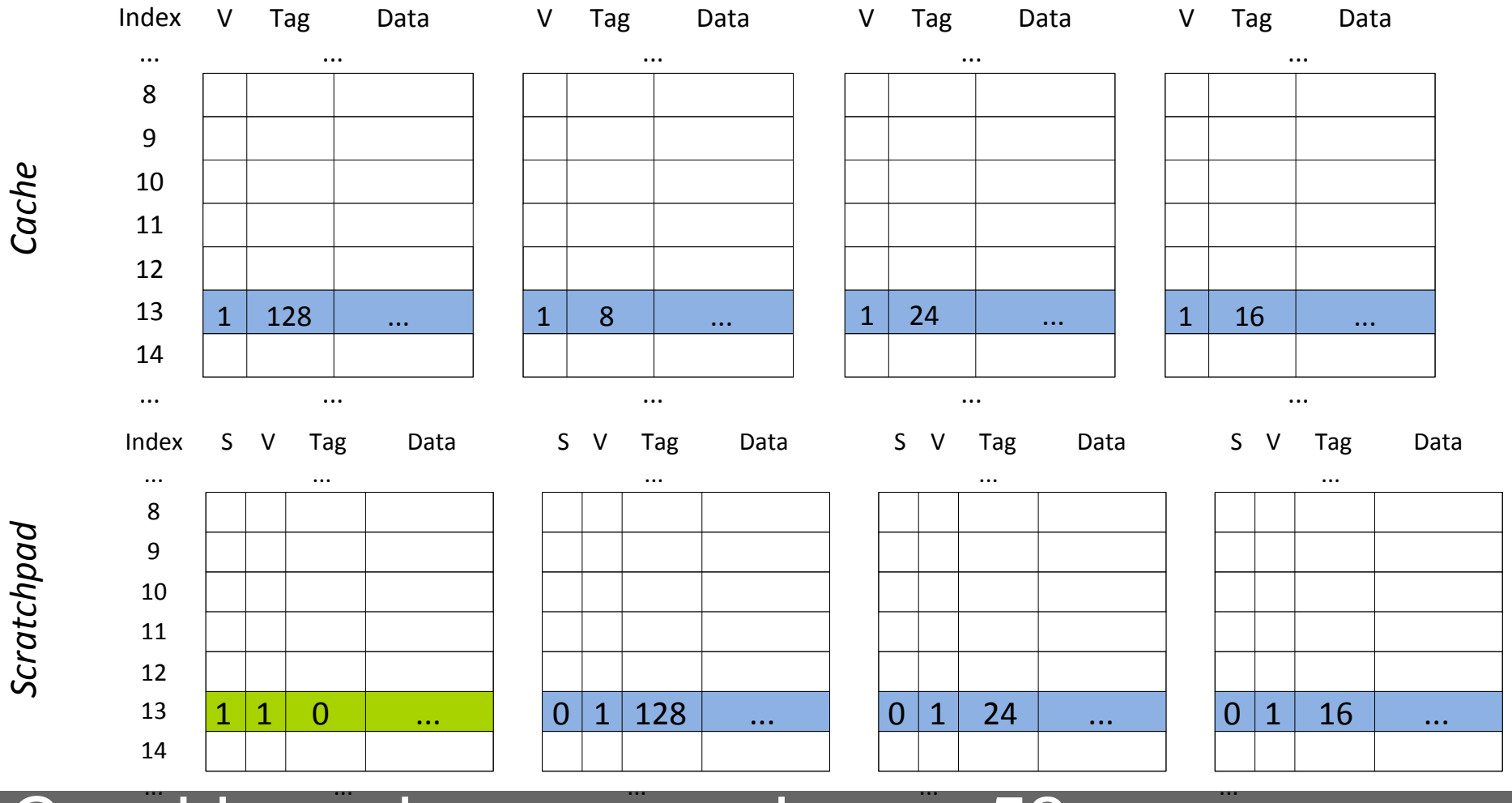
Se o elemento menos recentemente utilizado que não compõe a *scratchpad* for o assinalado, ele seria substituído...

...Funcionamento da *scratchpad*



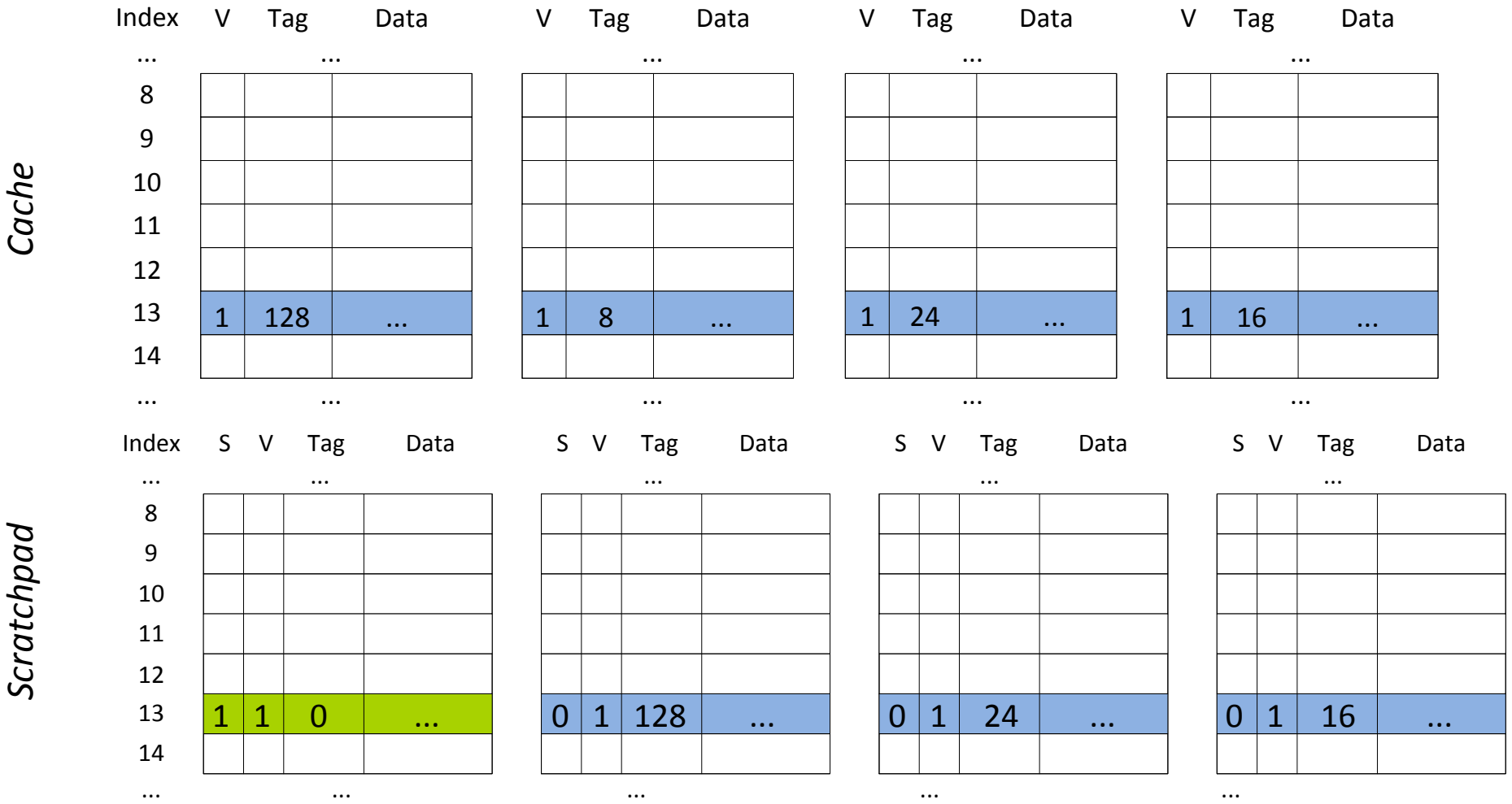
Se o elemento menos recentemente utilizado que não compõe a *scratchpad* for o assinalado, ele seria substituído...

...Funcionamento da *scratchpad*



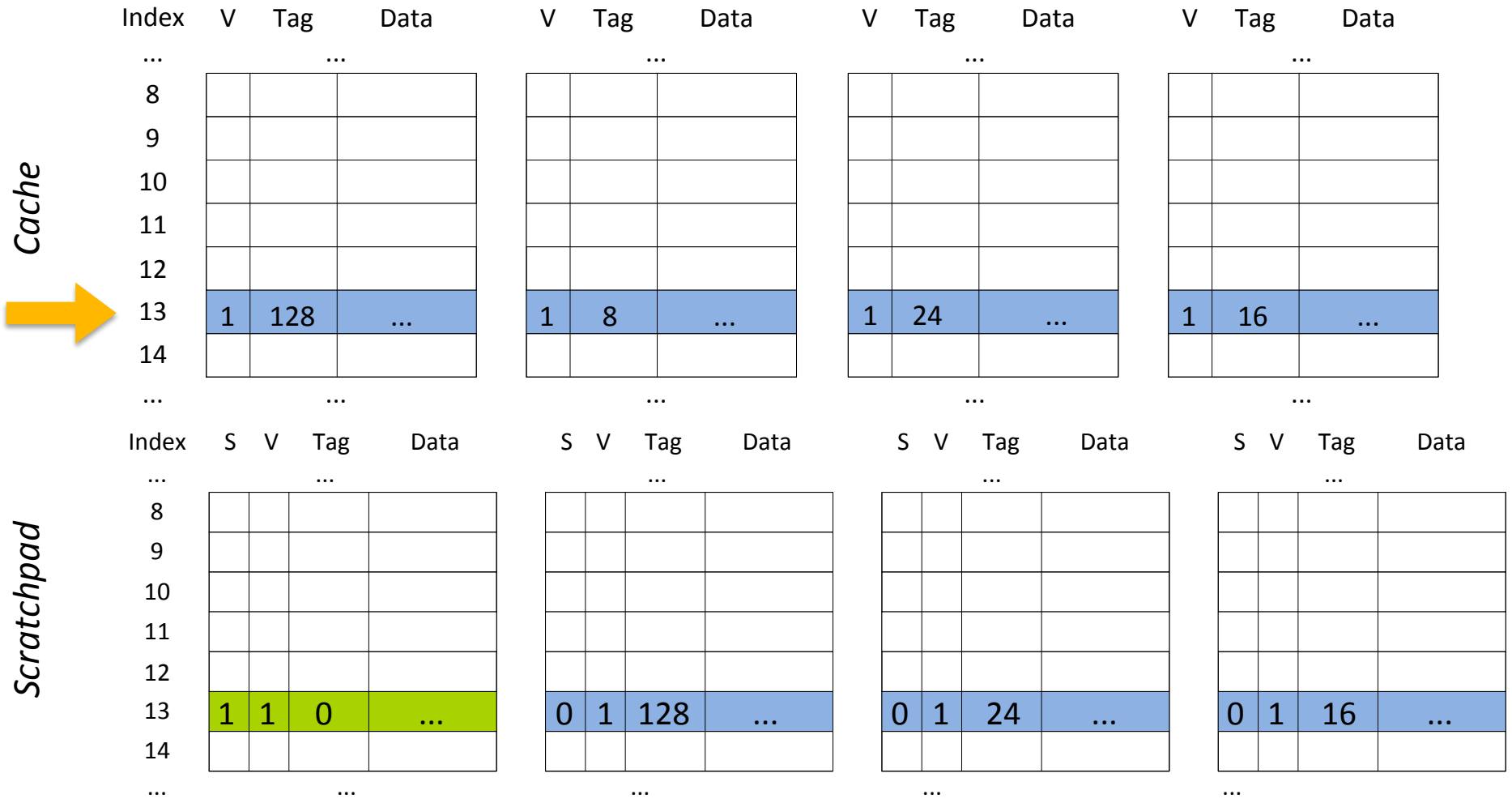
Considerando que o endereço 52 era importante e por isso foi atribuído a *scratchpad*...

...Funcionamento da *scratchpad*



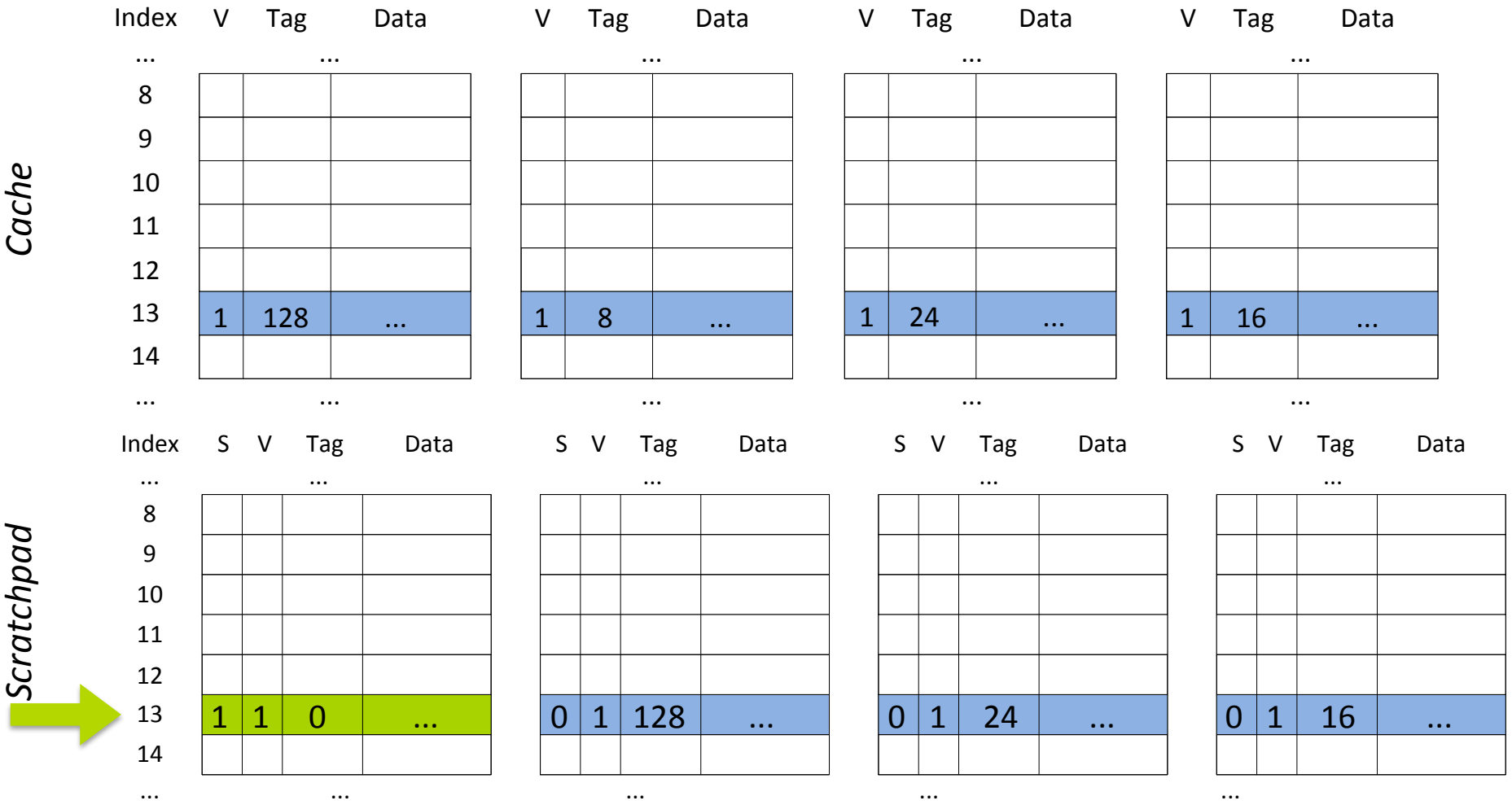
Ele novamente é solicitado pelo processador...

...Funcionamento da *scratchpad*



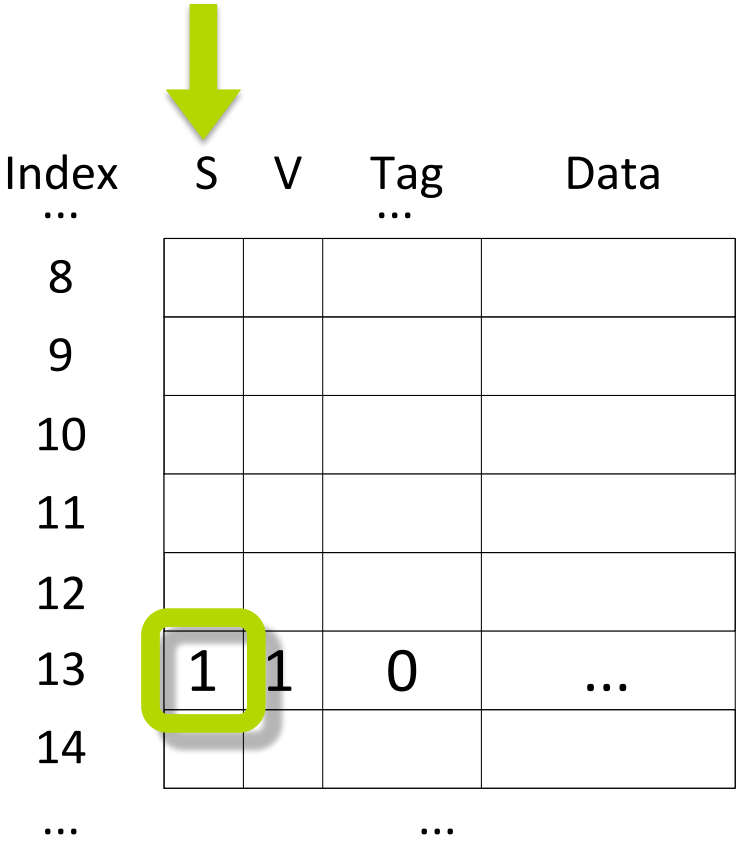
A *cache* o derramou: **falha!**

...Funcionamento da *scratchpad*



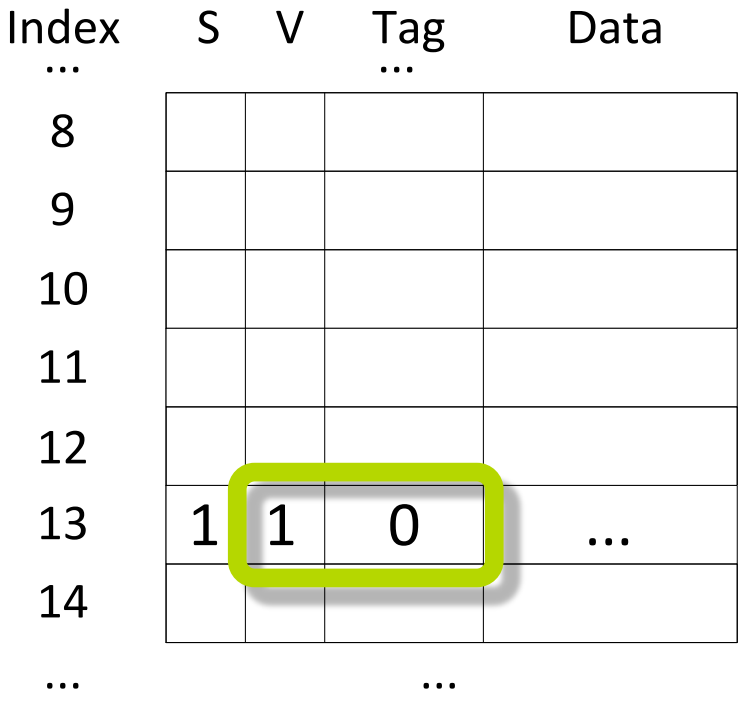
Por ser **importante** foi atribuído a *scratchpad*, **não foi** e **nem será derramado**: **acerto!**

...Funcionamento da *scratchpad*



Elemento assinalado como ***scratchpad***.

...Funcionamento da *scratchpad*



Bit de validade (V) e a *Tag* são consistentes. Isso permite um **tratamento uniforme** utilizando as instruções de *load* e *store*.

Unidade de alocação

Uma palavra: **excesso** de chamadas para a instrução de alocação;

Muitas palavras: **desperdício** ao alocar poucos dados;

Solução: um bloco da cache – expande a **compatibilidade** e **facilita mensurar** os resultados.

Alocação

Para **melhorar a eficiência** da solução, o processador solicita a transferência dos dados e uma unidade de DMA a realiza **o liberando** para a computação.

Transparência

A memória *scratchpad* é provida com uma metodologia **transparente**, seus endereços são tratados de modo **uniforme** sem que qualquer abstração adicional seja necessária.

Implicações

- O código legado **não é afetado**;
- O desempenho pode **melhorar** ou **piorar** de acordo com as decisões do desenvolvedor;

...Implicações

O *hardware* ou o sistema operacional precisa definir uma abordagem para limitar o uso da memória *scratchpad*.

Troca de contexto

Em um ambiente multiprogramado quando uma troca de contexto ocorrer:

- Se todos os elementos de uma linha forem alocados como *scratchpad* um **endereço poderia tornar-se inacessível**;
- Um mapeamento associativo por conjunto poderia ser reduzido a um **mapeamento direto**.

...Troca de contexto

Soluções:

- Limitar as aplicações que podem utilizar a *scratchpad*;
- Limitar a quantidade de *scratchpad* alocada pelas aplicações;

...Troca de contexto

Soluções:

• Os dados da *scratchpad* não são derramados durante sua execução, mas na troca de contexto são tratados como dados comuns.

- **Desempenho afetado**;
- Semântica da *scratchpad* seria **modificada**;
- **Falha da *scratchpad***;
- Uma aplicação **não interferiria** na execução das demais aplicações;

Endereçamento

Há duas opções:

- Os endereços podem ser abstraídos pelo compilador, como feito pela função *malloc* da linguagem C;
- Os endereços podem ser administrados diretamente pelo desenvolvedor.
  - **Simplifica** o projeto das APIs;
  - **Aumenta a complexidade** do desenvolvimento de aplicações;
  - Precisa **conhecer os endereços** utilizados pelo compilador antes de compilar ou o compilador lidar com isso.

Abordagem empregada

- Não há restrições quanto ao uso da memória *scratchpad*;
- Os endereços ficaram sob administração do desenvolvedor:
  - Uso de endereços com deslocamento grande o suficiente para evitar colisões;
  - Separação de memória de dados e instruções evitando que dados sejam sobrescritos.

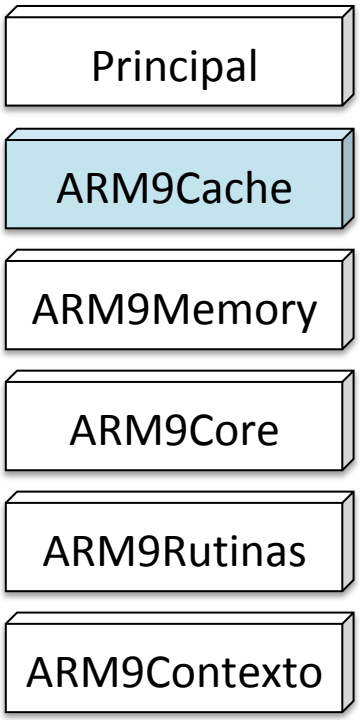
Simulador

- Processador **RISC**;
- Arquitetura ARM 9;
- Implementação utilizando FPGA
  - **Falta de documentação**;
  - **Complexidade**;
- Simulador.

...Simulador

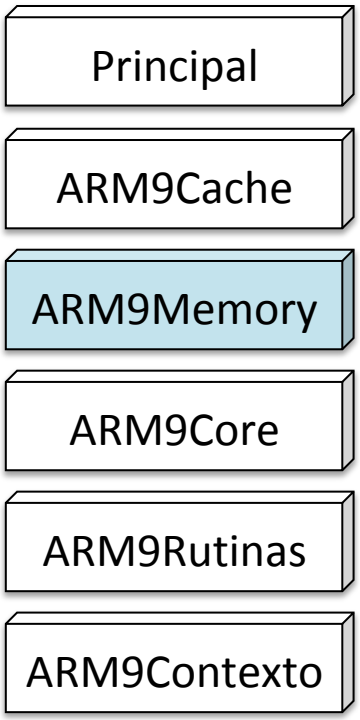
- Executa as instruções a partir do endereço 0 da memória;
- Sem sistema operacional:
  - Instruções para entrada e saída de dados;
  - Instrução de parada;
  - Instrução de log.
- Instruções da *scratchpad.*

...Simulador



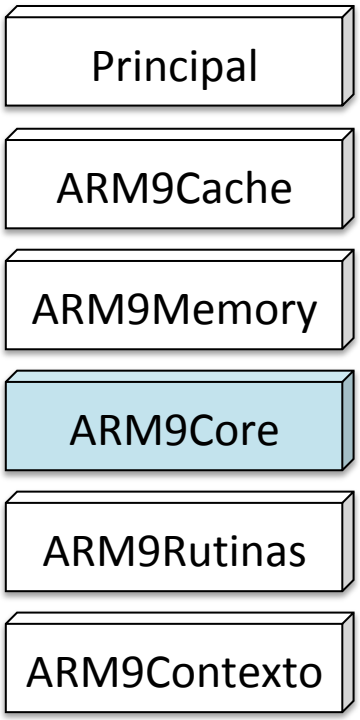
Implementa a ***cache***. Inclui a ***scratchpad***.

...Simulador



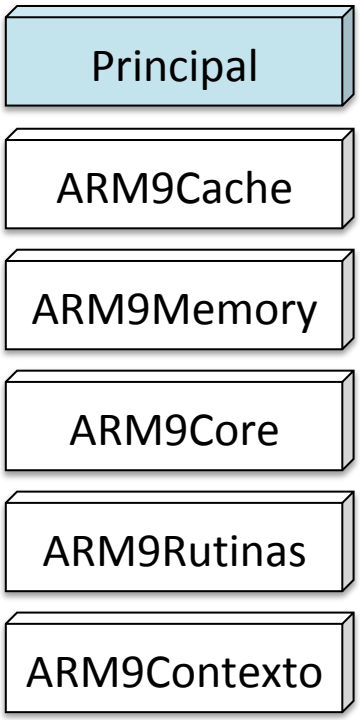
Subclasse da *cache* que implementa a memória.

...Simulador



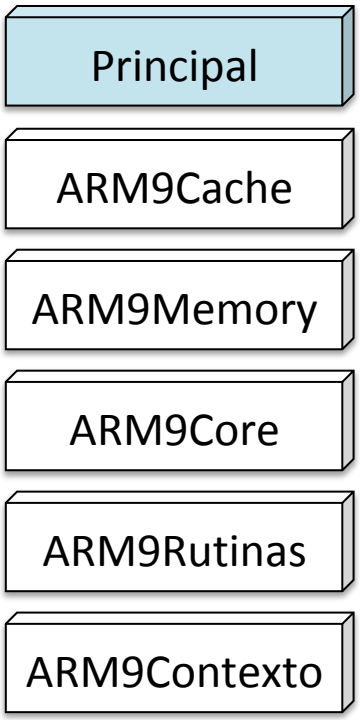
Representa uma unidade de processamento.

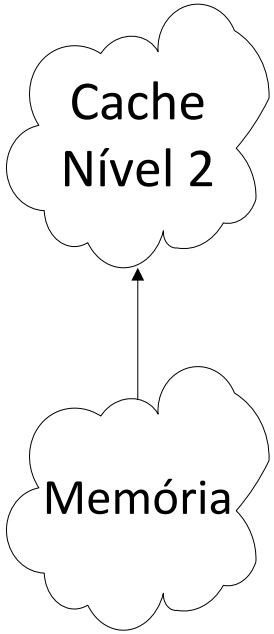
...Simulador



Cria uma ocorrência da memória principal.

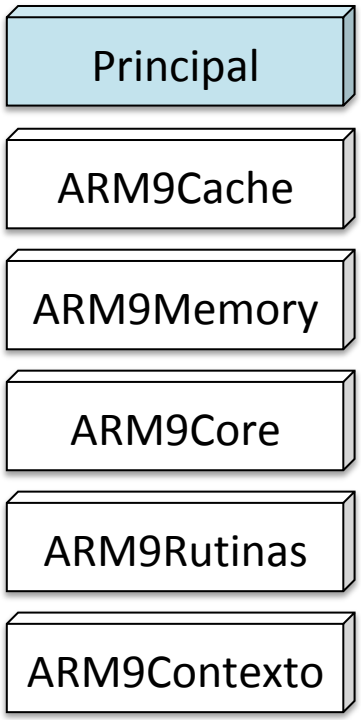
...Simulador

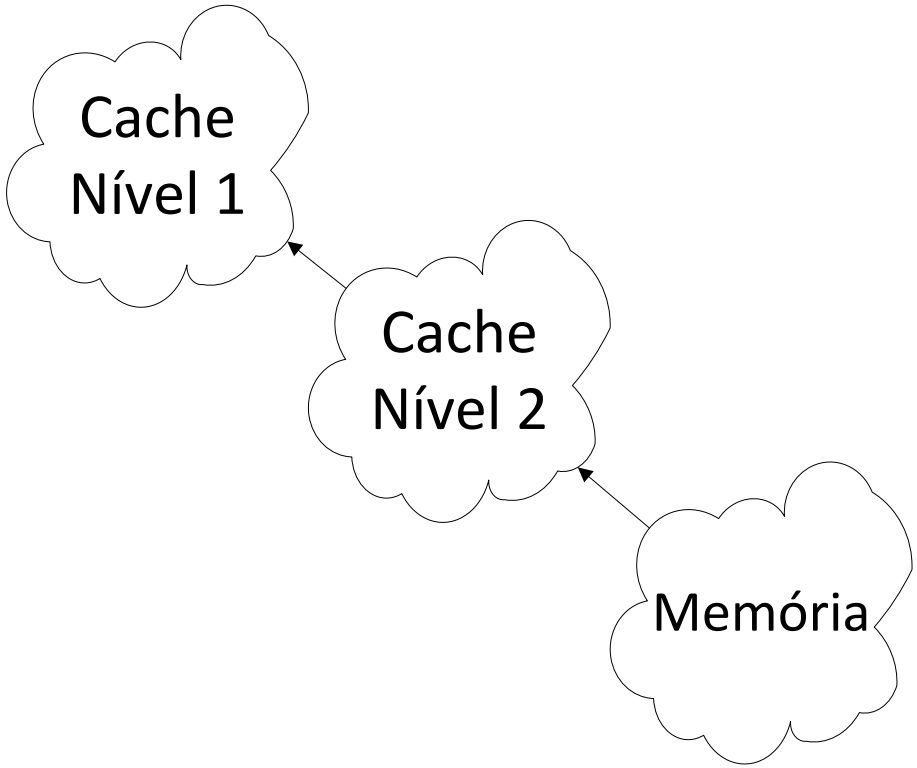




Cria uma ocorrência da *cache* e a configura como *cache* de nível 2. A memória é configurada como próximo nível da hierarquia.

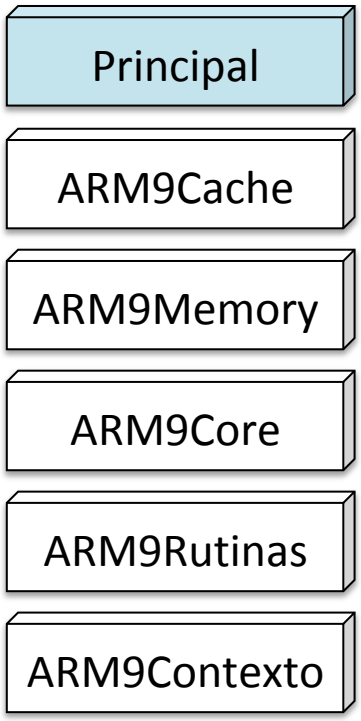
...Simulador

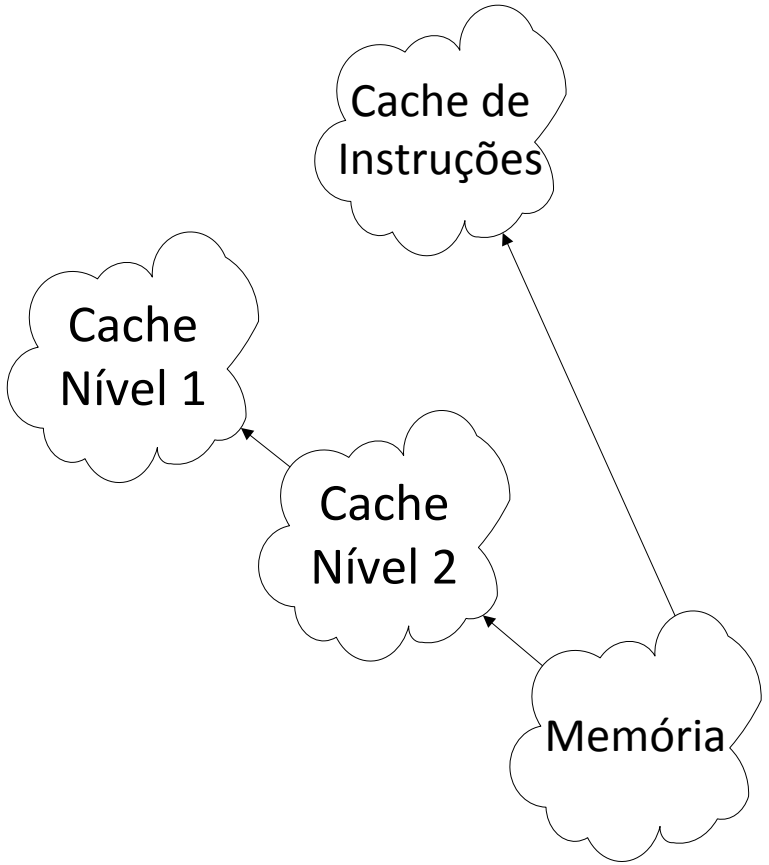




Cria uma ocorrência da *cache* e a configura como *cache* de nível 1. A *cache* de nível 2 é configurada como próximo nível da hierarquia.

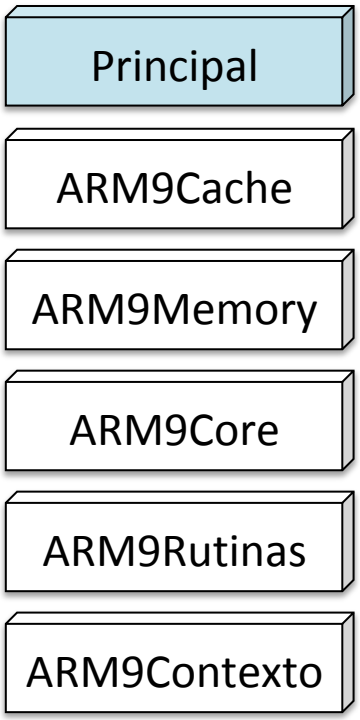
...Simulador

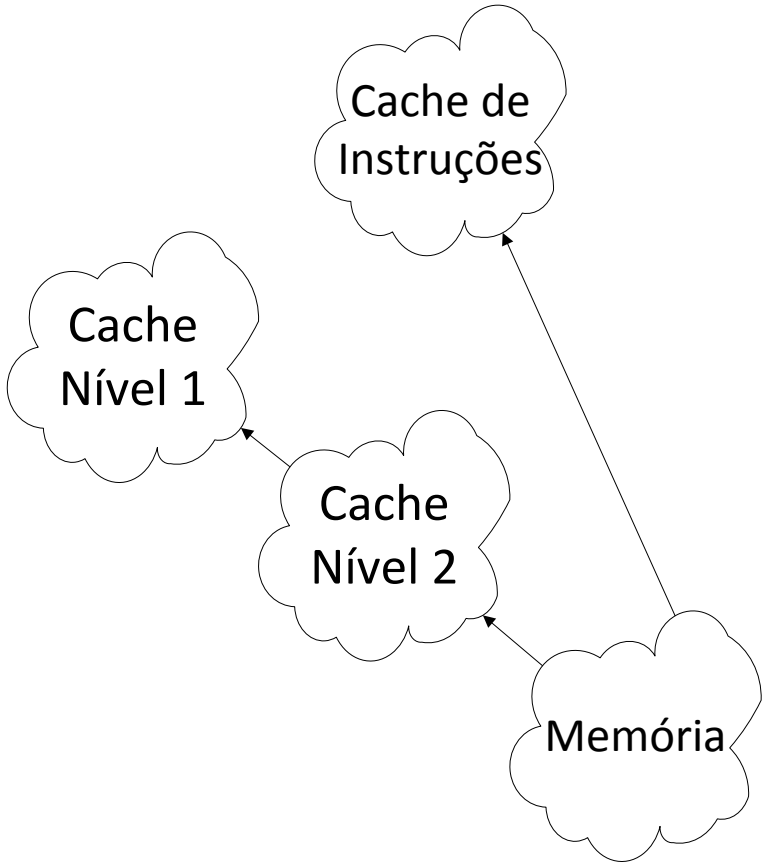




Cria a *cache* de instruções.

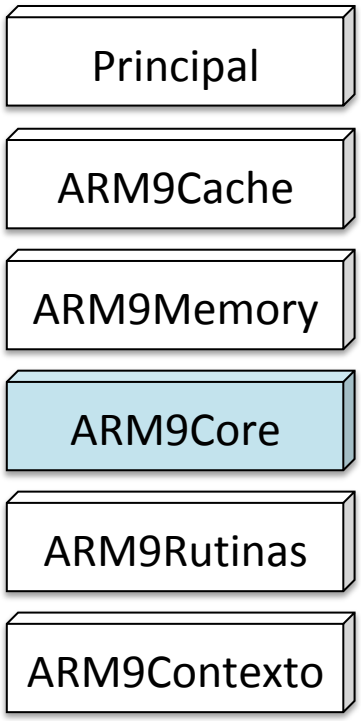
...Simulador

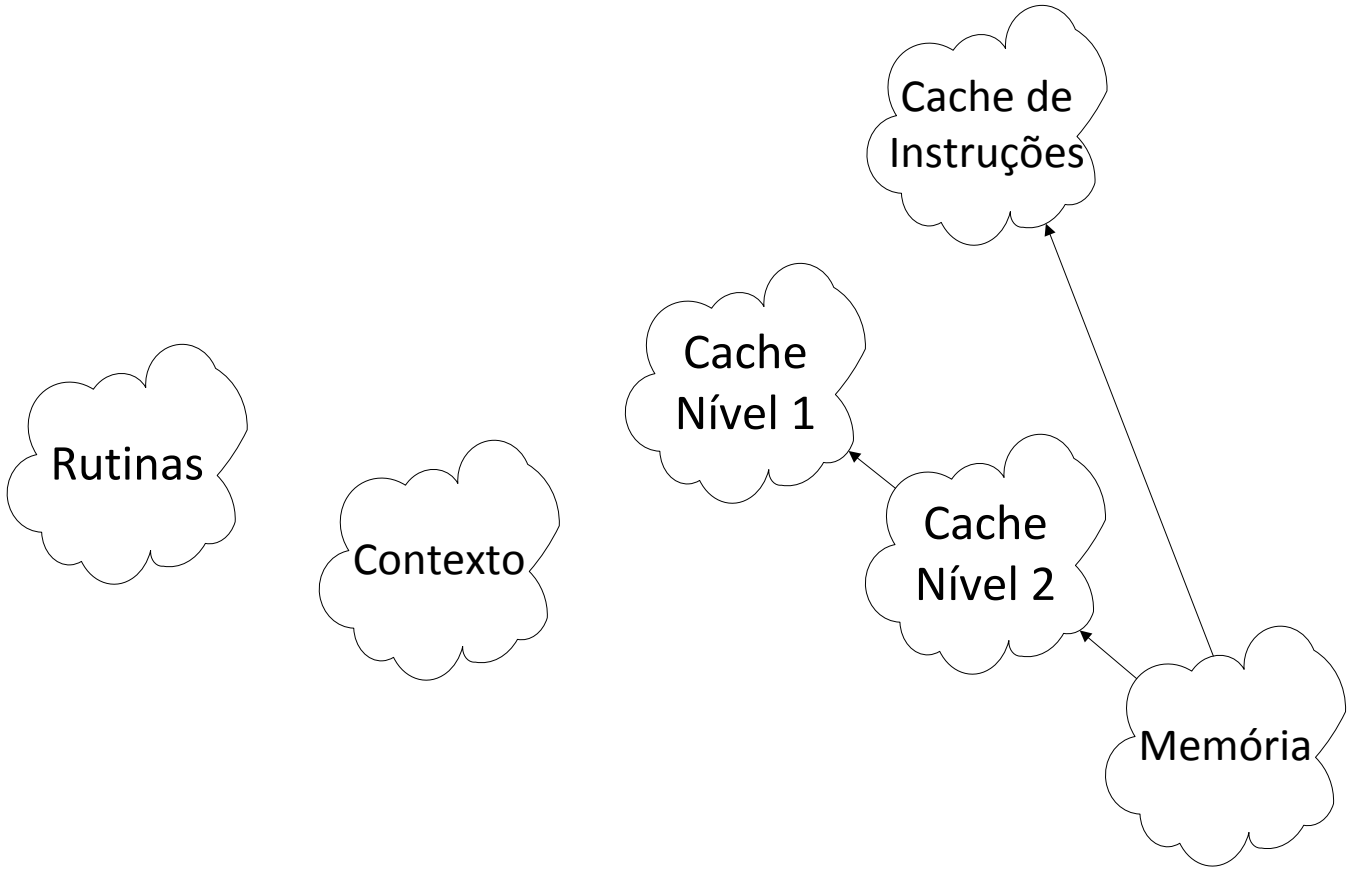




Solicita a criação de uma ocorrência da unidade de processamento.

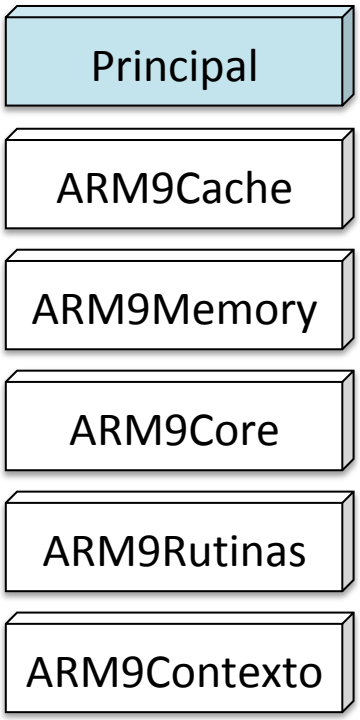
...Simulador

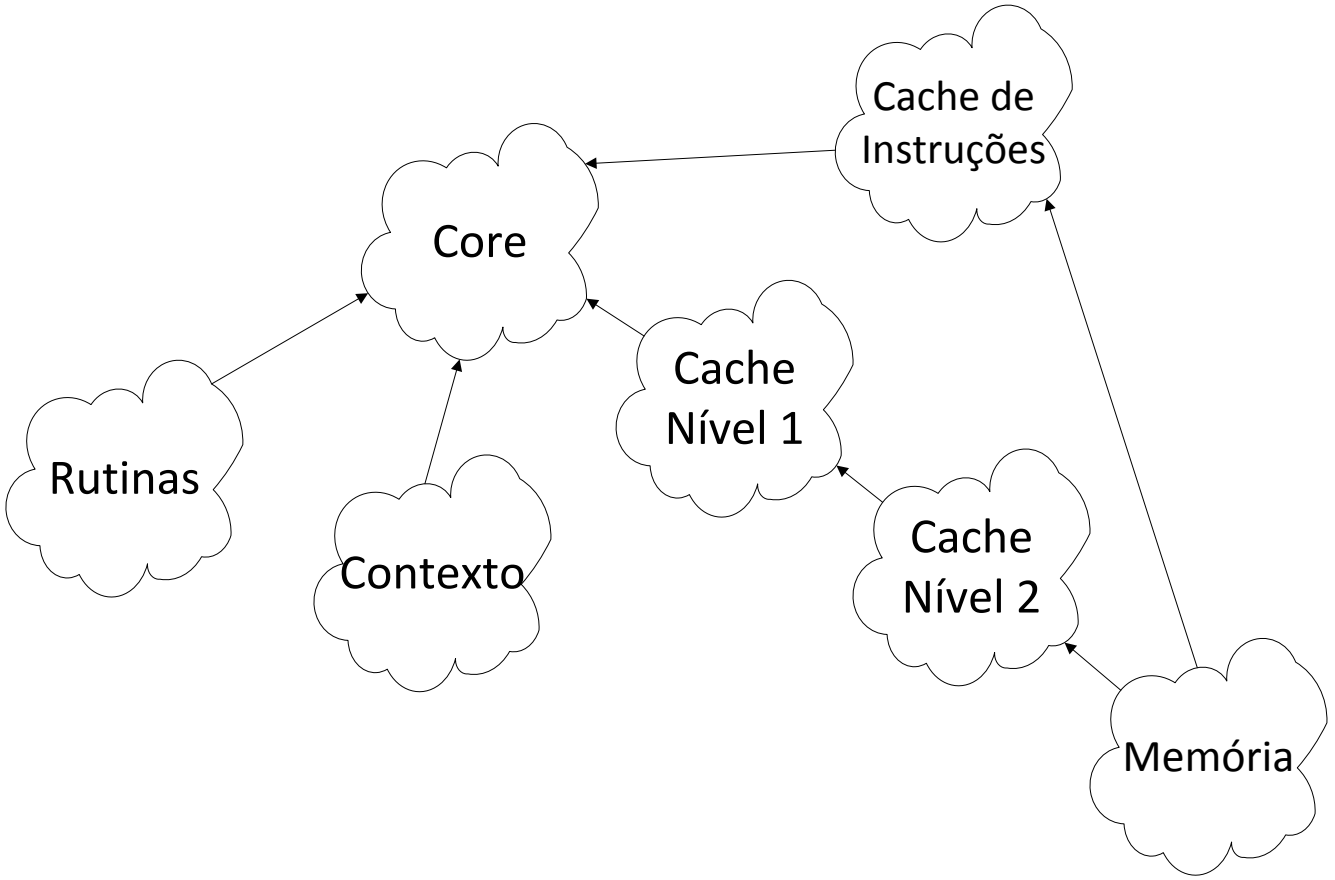




O construtor de *Core* cria uma ocorrência de *Rutinas* e de *Contexto*.

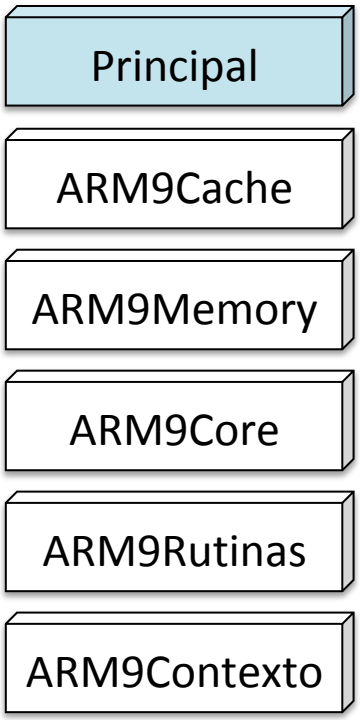
...Simulador

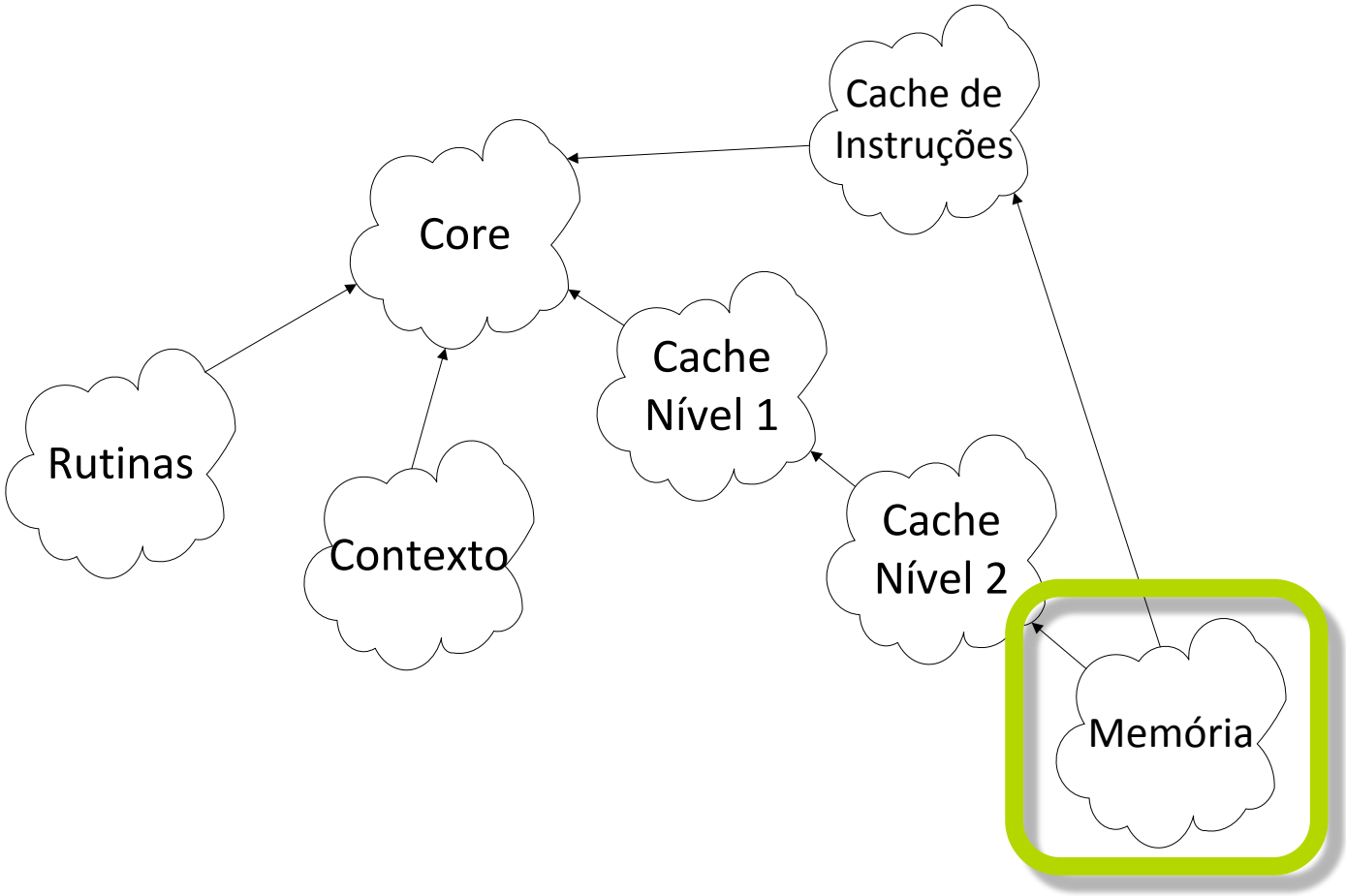




Uma ocorrência da unidade de processamento é criada ligando-a com a hierarquia de memória.

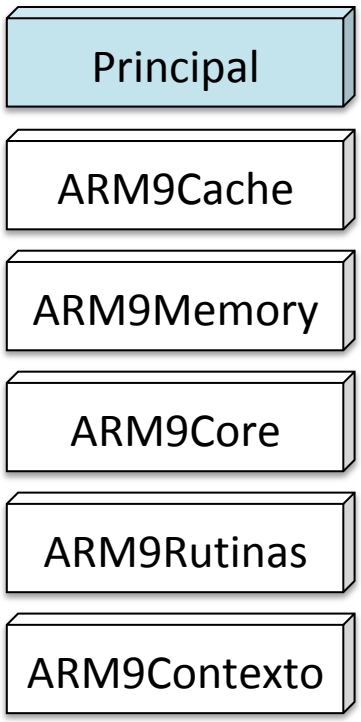
...Simulador

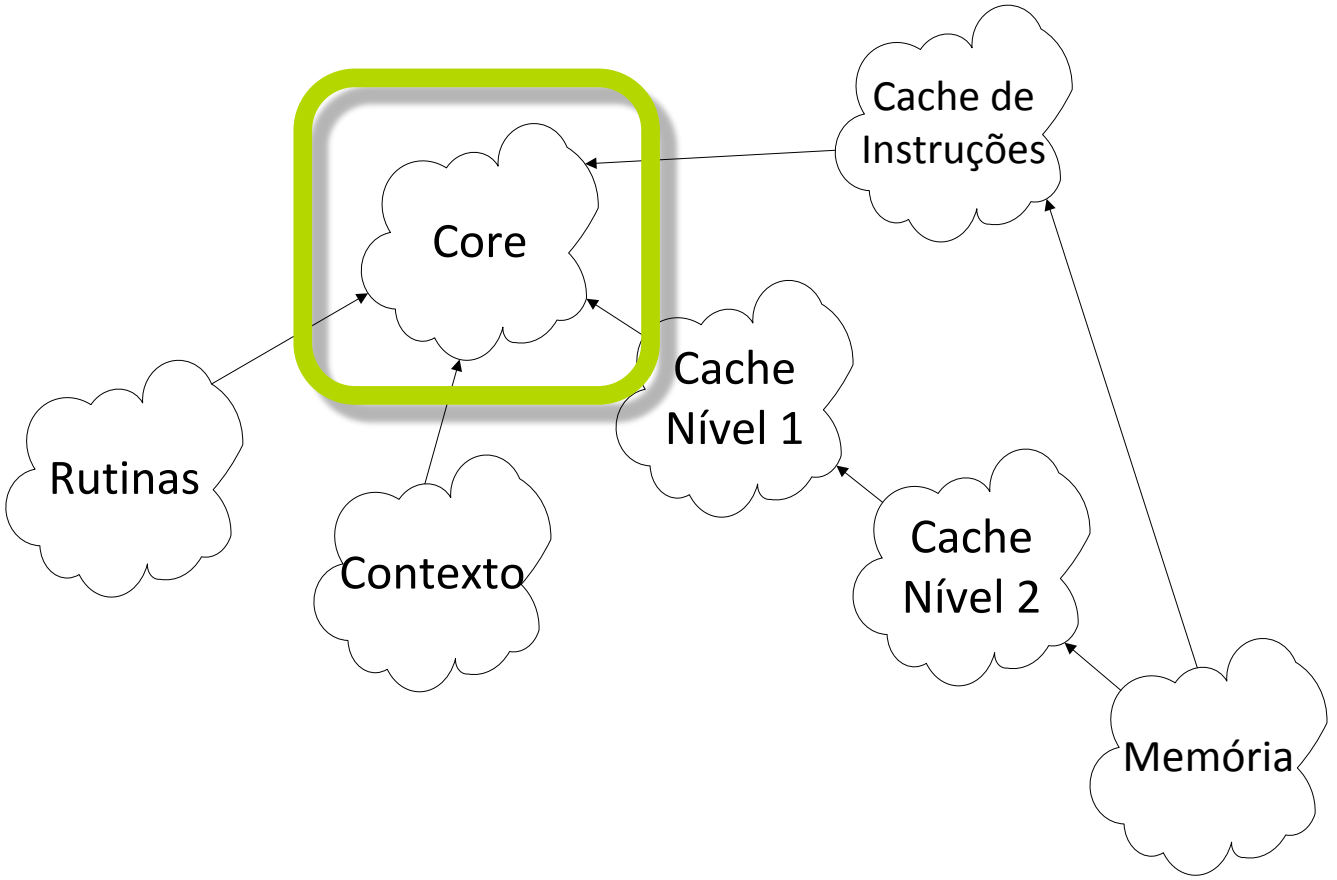




Os dados da aplicação são carregados na **Mémoria** principal.

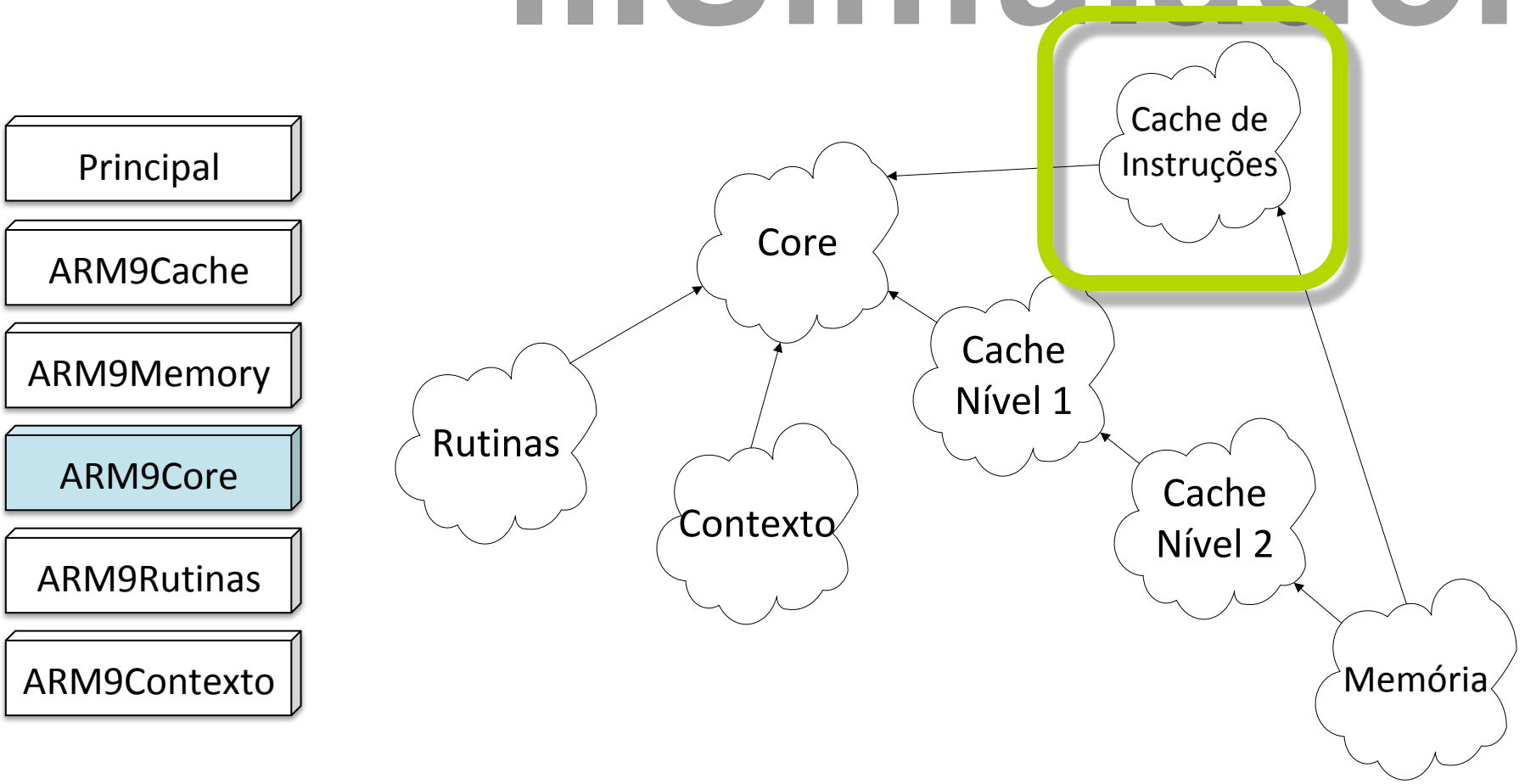
...Simulador





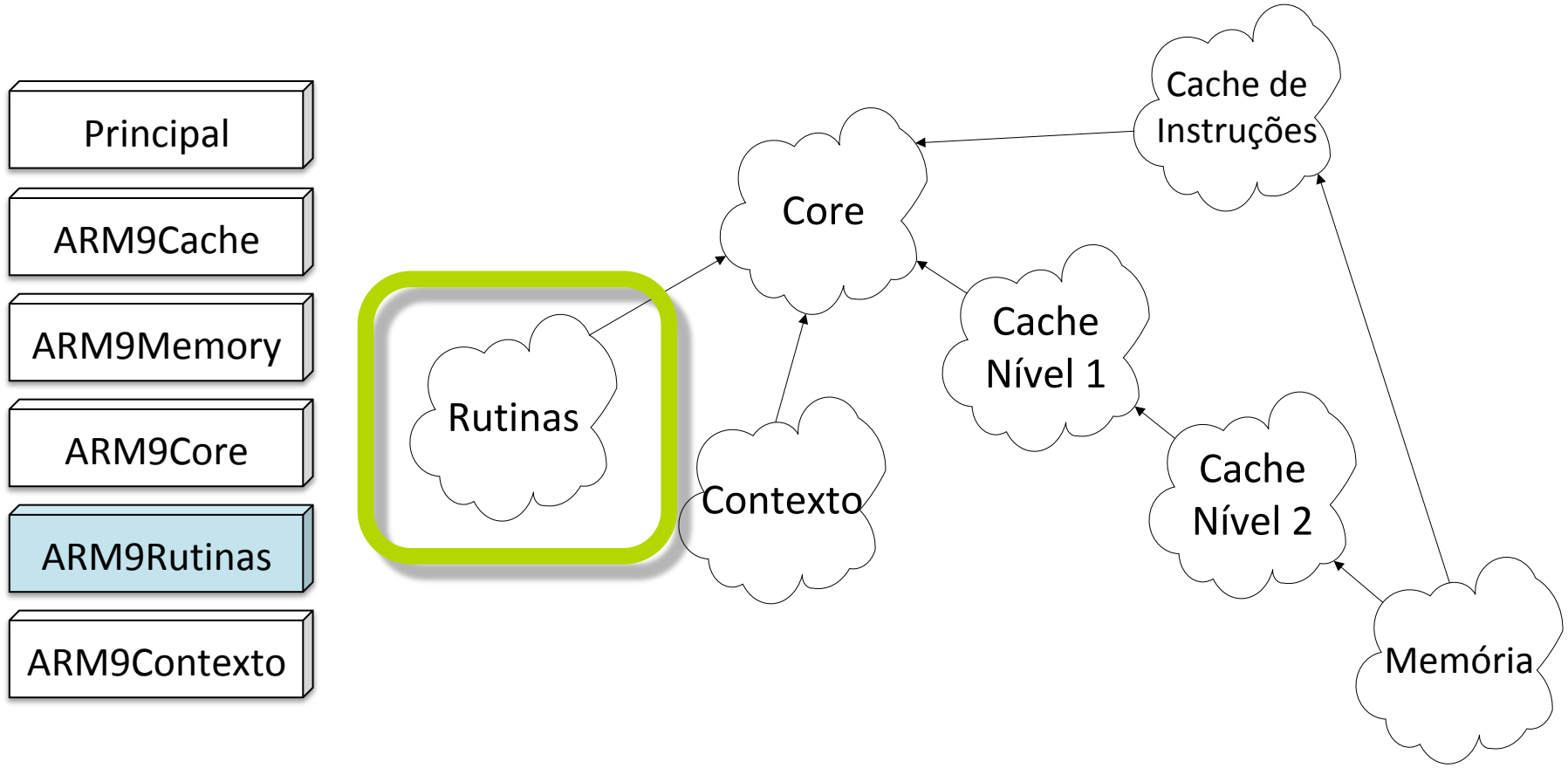
O método *run* de **Core** é invocado e a execução se inicia.

...Simulador



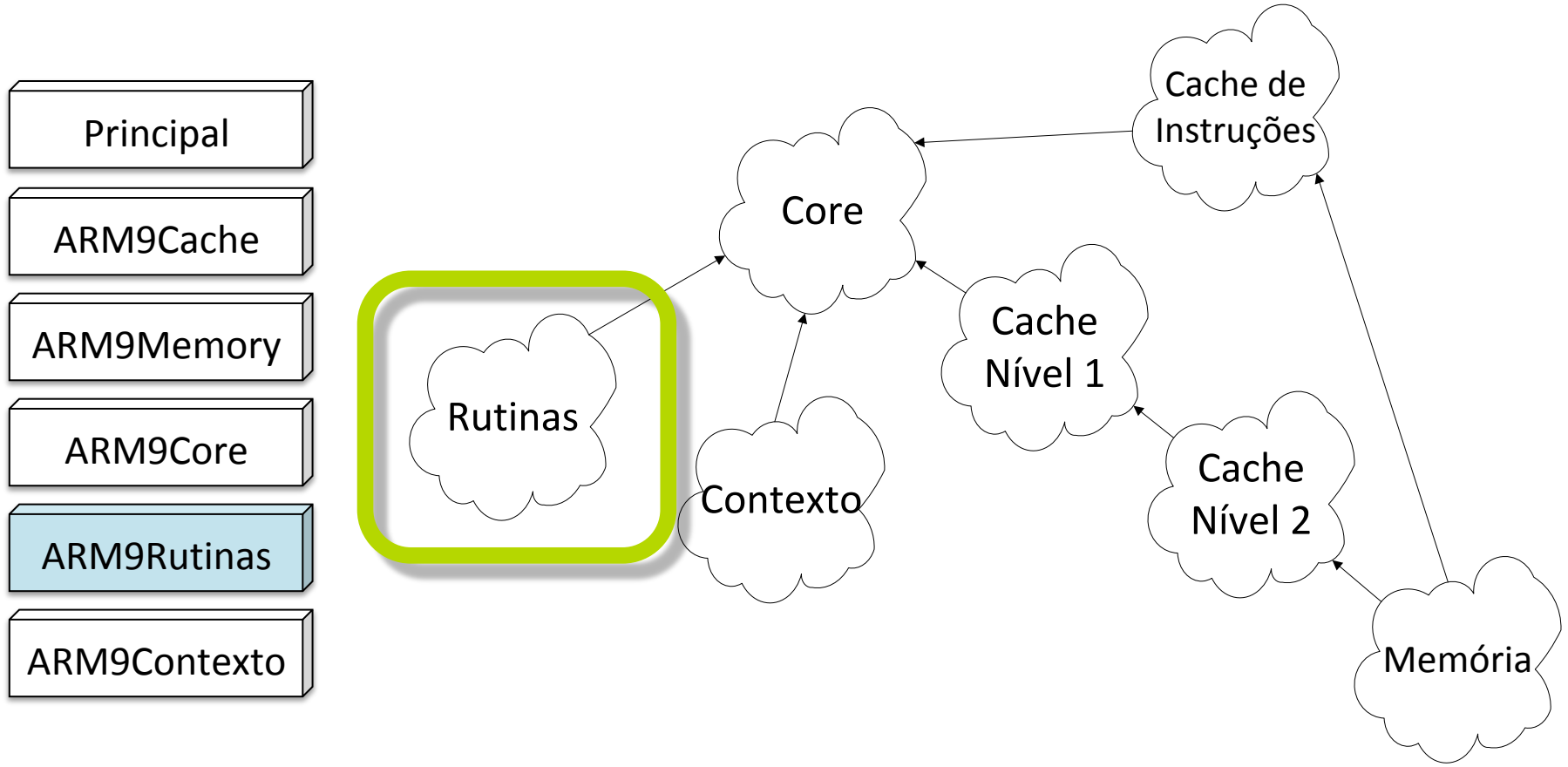
As instruções são lidas da ***Cache* de instruções**...

...Simulador



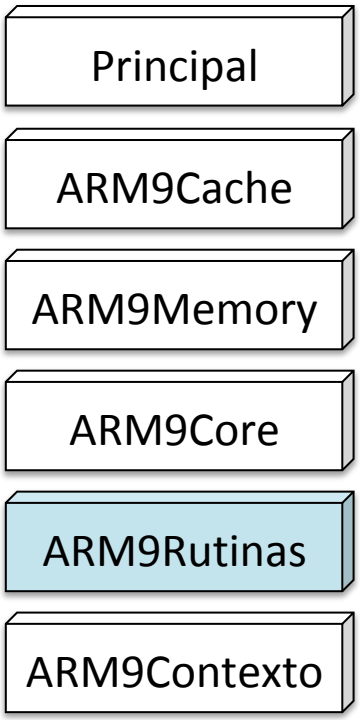
E enviadas para ***Rutinas*** onde são decodificadas e executadas...

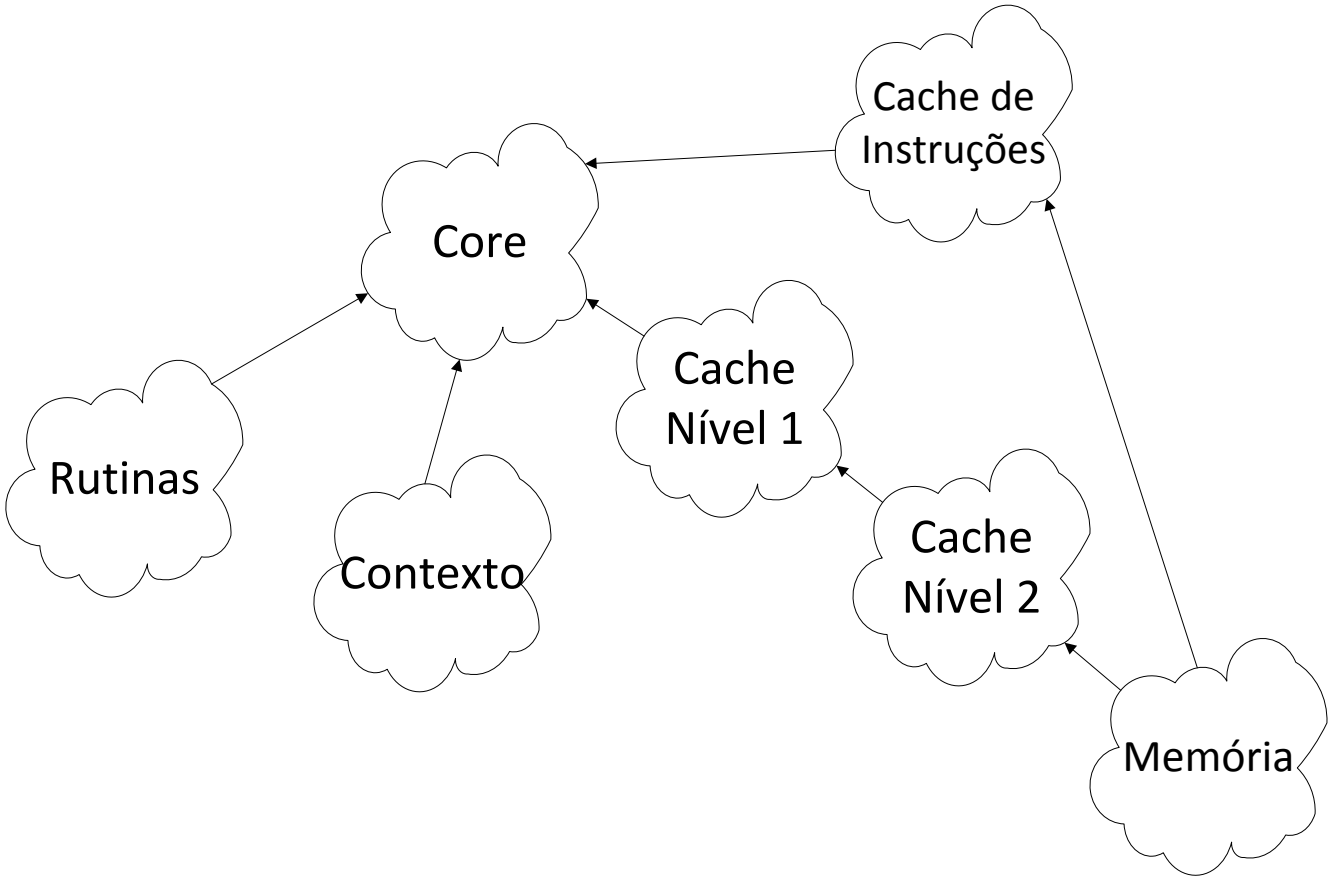
...Simulador



Rutinas foi alterado para contemplar as instruções da *scratchpad*...

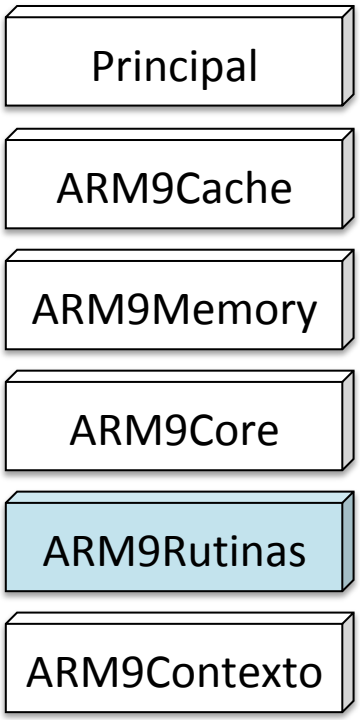
...Simulador

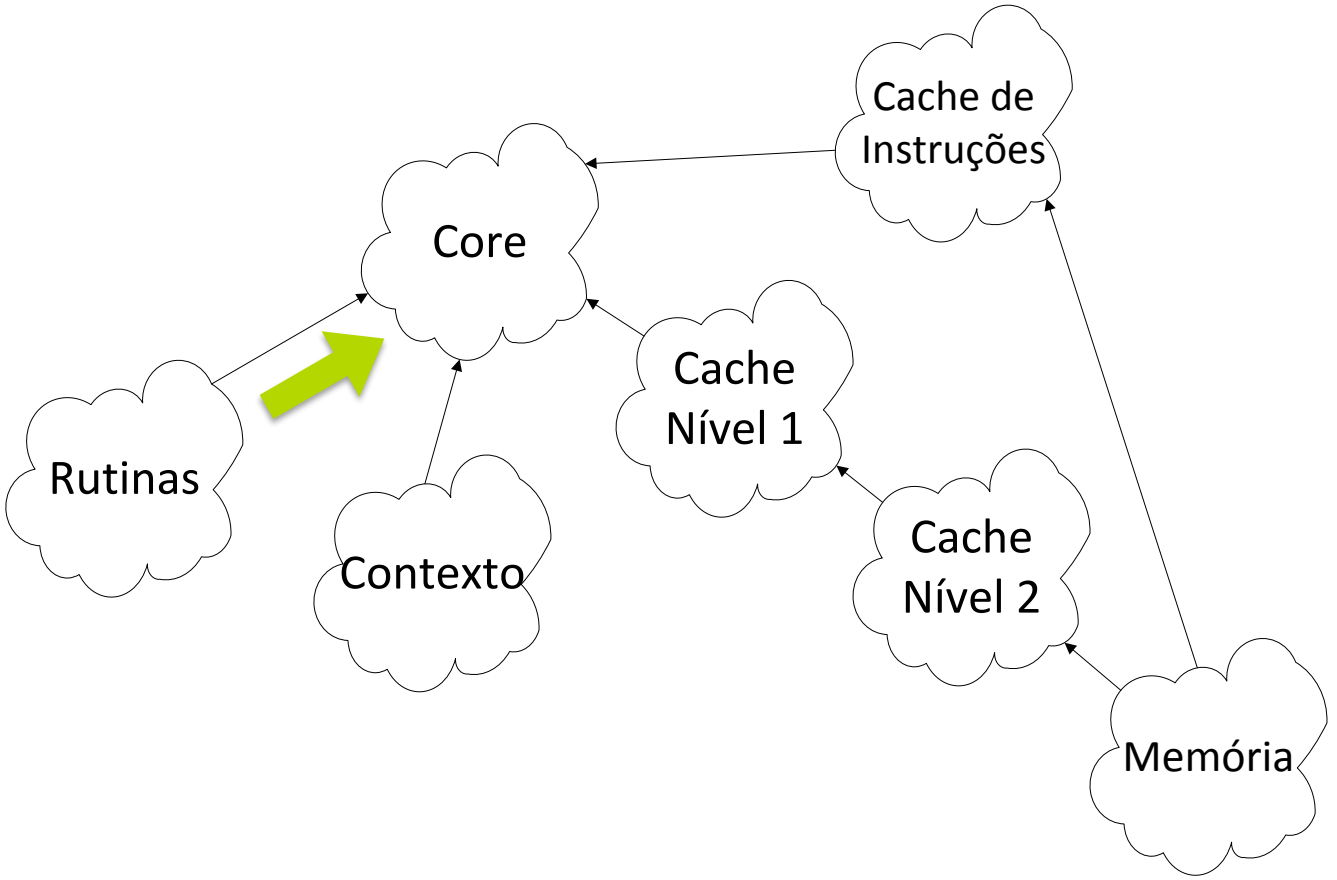




Ao encontrar uma instrução da *scratchpad* ou uma solicitação por dados.

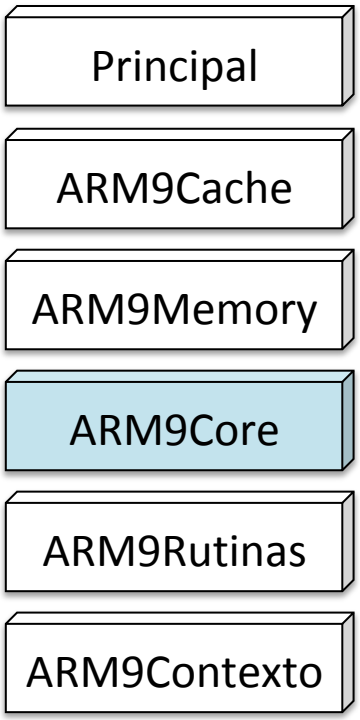
...Simulador

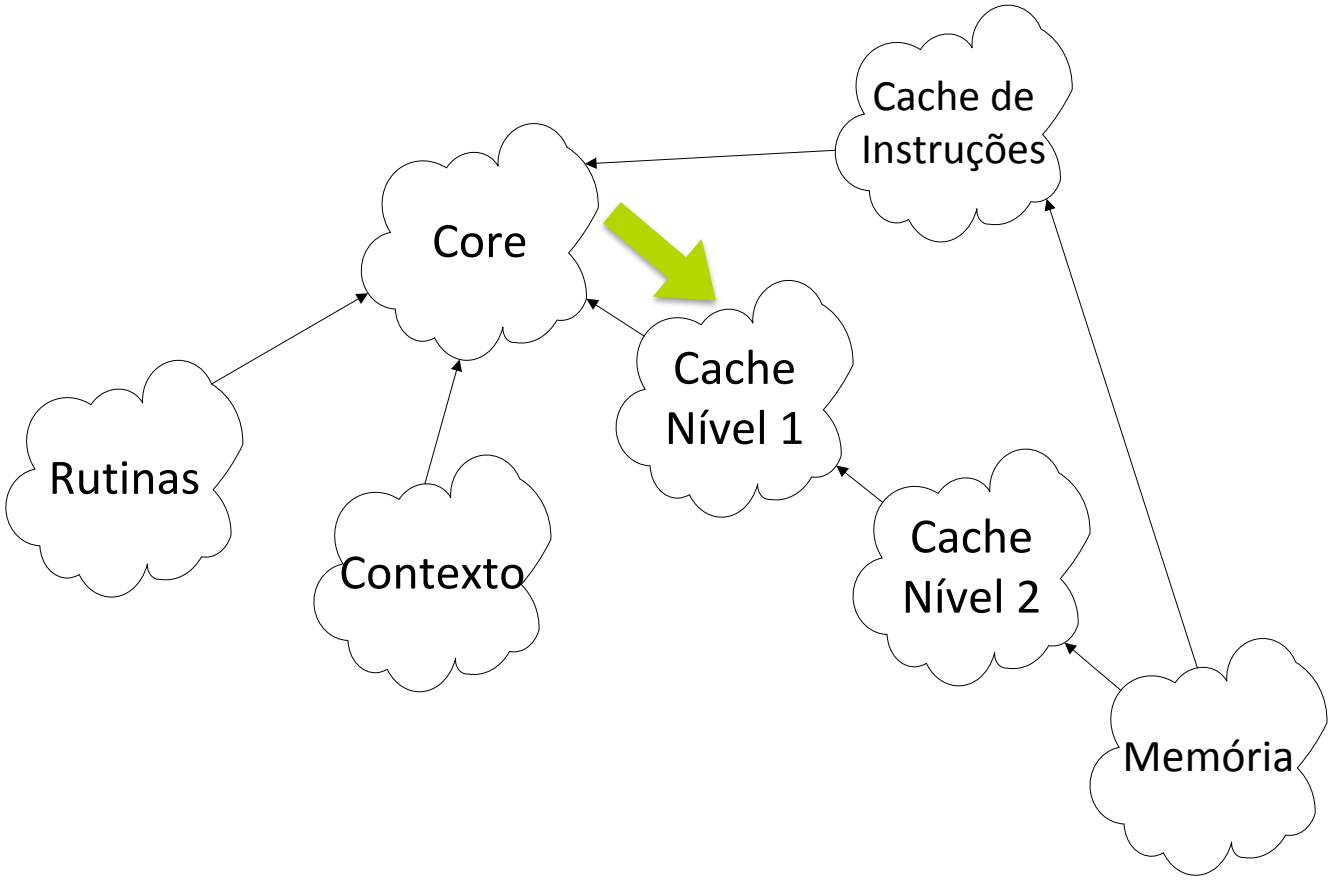




Rutinas envia uma mensagem para *Core*...

...Simulador





Core por sua vez a repassa para *Cache Nível 1.*

Instruções

SPALLOC RS

Aloca na memória *scratchpad* um bloco para o endereço dado no registrador *RS*.

...Instruções

```
Address spAlloc(Address address){

    uint32 replace = getReplaceBlockIndex(address);
    Address baseAddress =
        int(address / blockSizeBytes) * blockSizeBytes;

    // Alocado como scratchpad
    data[replace][baseAddress.index].S = true;
    // Marcar os dados como validos para uso pelos
    // ldr e str tradicionais
    data[replace][baseAddress.index].v = true;
    data[replace][baseAddress.index].tag = baseAddress.tag;

    return address;
}
```

Pseudocódigo para a instrução *SPALLOC*.

...Instruções

SPFREE RS

Libera da memória *scratchpad* o bloco do endereço dado no registrador *RS*.

...Instruções

```
void spFree(Address address){

    uint32 blockIndex = getBlockIndex(address);
    Address baseAddress =
        int(address / blockSizeBytes) * blockSizeBytes;

    // Liberar a scratchpad
    data[blockIndex][baseAddress.index].S = false;
    data[blockIndex][baseAddress.index].v = false;

}
```

Pseudocódigo para a instrução *SPFREE*.

...Instruções

SPLOAD RS RD

Transfere o bloco do próximo nível da hierarquia dado no registrador *RD* para o bloco do endereço dado em *RS* na memória *scratchpad*.

...Instruções

```
void spLoad(Address memAddress, Address spAddress){

    Address baseSpAddress =
        (int)(spAddress / blockSizeBytes) * blockSizeBytes;

    uint32 spmIndex = associativitySize;
    // Obter o buffer associado a scratchpad
    for (uint32 i = 0; i < associativitySize; i++){
        if (data[i][baseSpAddress.index].S &&
            baseSpAddress.tag == data[i][baseSpAddress.index].tag){
            spmIndex = i;
            break;
        }
    }

  ...
```

Pseudocódigo para a instrução *SPLOAD*.

...Instruções

```
...
    uint32 pos = 0;
    // Copiar os dados
    for (uint32 i = 0; i < blockSizeBytes; i += 4){

        uint32 value = nextLevel->readMemoriaWord(memAddress + i);

        *((uint32*)(
            data[spmIndex][baseSpAddress.index].data +
            (baseSpAddress.blockOffset + pos) * 4)) = value;

        pos++;
    }
}
```

Pseudocódigo para a instrução *SPLOAD*.

...Instruções

SPSTORE RS RD

Transfere dados da *scratchpad* para o próximo nível da hierarquia. O bloco de origem é dado no registrador *RS* e o bloco de destino em *RD*.

...Instruções

```
void spStore(Address memAddress, Address spAddress){

    Address baseSpAddress =
        (int)(spAddress / blockSizeBytes) * blockSizeBytes;

    uint32 spmIndex = associativitySize;
    // Obter o buffer associado a scratchpad
    for (uint32 i = 0; i < associativitySize; i++){
        if (data[i][adSp.index].S &&
            baseSpAddress.tag == data[i][baseSpAddress.index].tag){
            spmIndex = i;
        }
    }

  ...
```

Pseudocódigo para a instrução *SPLOAD*.

...Instruções

```
...

    uint32 pos = 0;

    for (uint32 i = 0; i < blockSizeBytes; i += 4){
        uint32 value = *(
            (uint32*)(
            data[spmIndex][baseSpAddress.index].data +
            (baseSpAddress.blockOffset + pos) * 4));
        nextLevel->writeMemoriaWord(baseMemAddress + i, value);

        pos++;
    }
}
```

Pseudocódigo para a instrução *SPLOAD*.

Software

Software

Solução Proposta

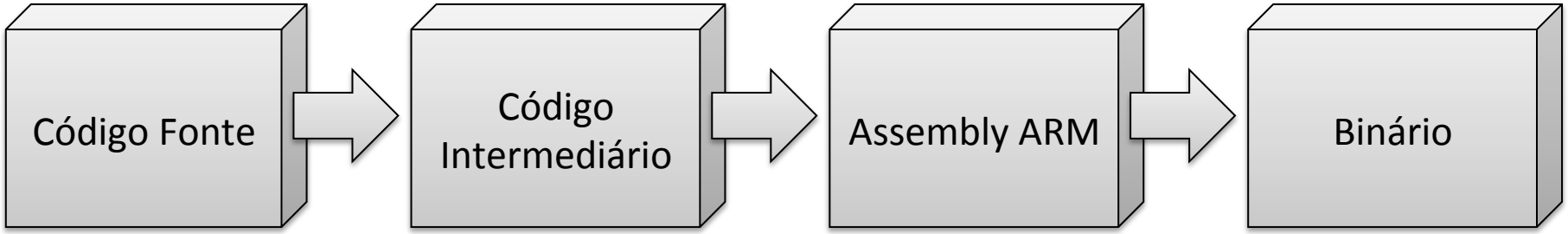
Definiu-se uma **biblioteca** (API) para a linguagem C com as quatro operações da *scratchpad*.

```
void* __cdecl spalloc(int address);

void __cdecl spfree(int address);

void __cdecl spload(int destAddress, int* src);

void __cdecl spstore(int srcAddress, int* dest);
```

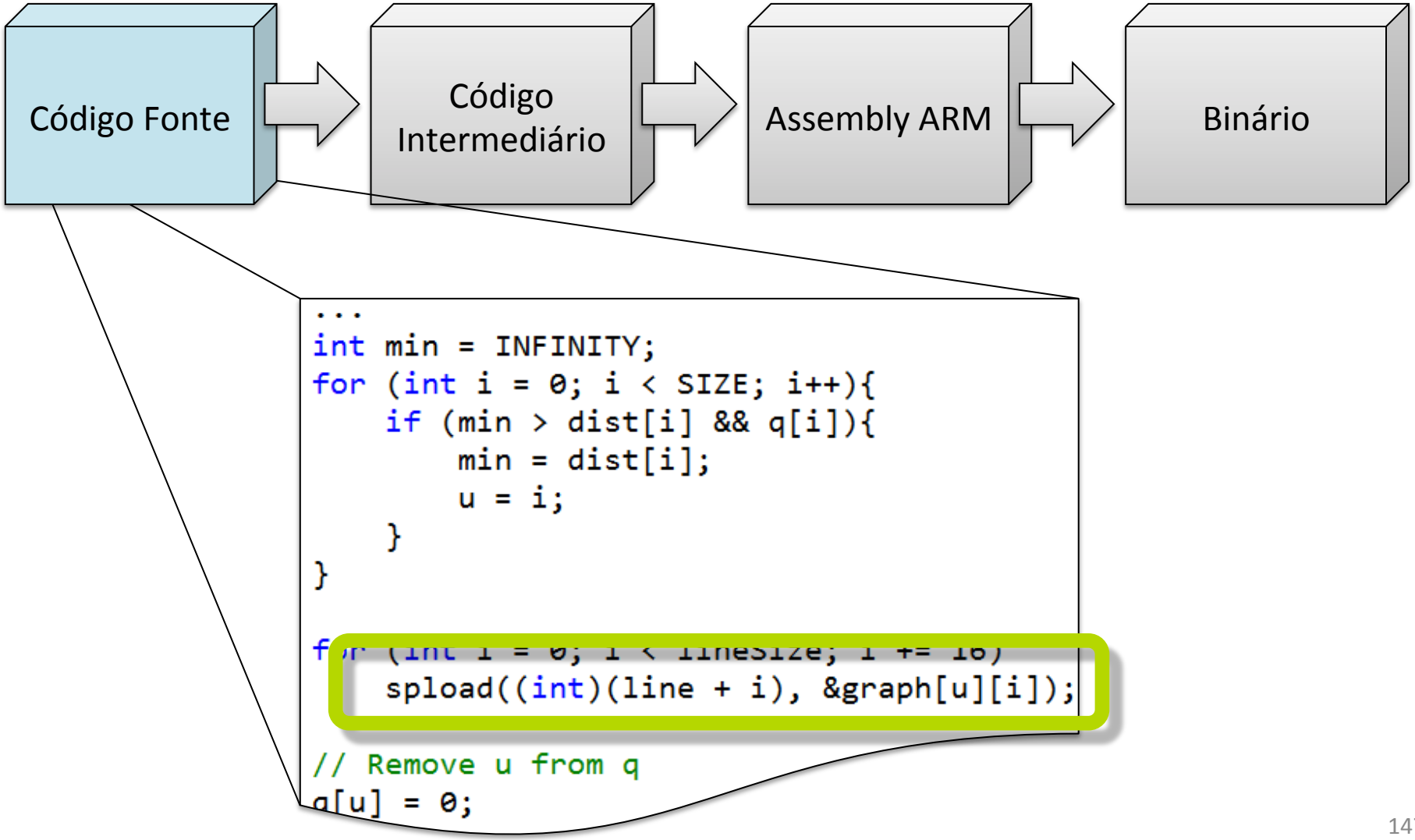
Compilador

- Compilação foi efetuada utilizando o LLVM;
  - As chamadas para as funções da biblioteca são **deixadas em aberto** pelo compilador;
  - LLC, **substitui** essas chamadas pelas **instruções equivalentes**.

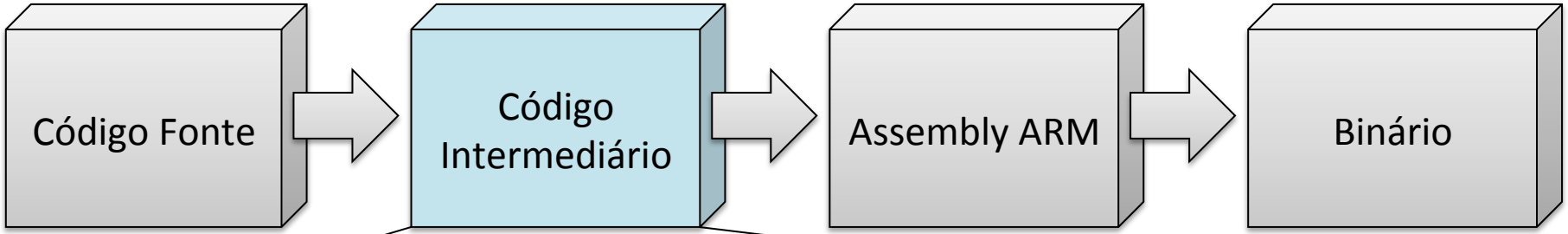
...Compilador



...Compilador

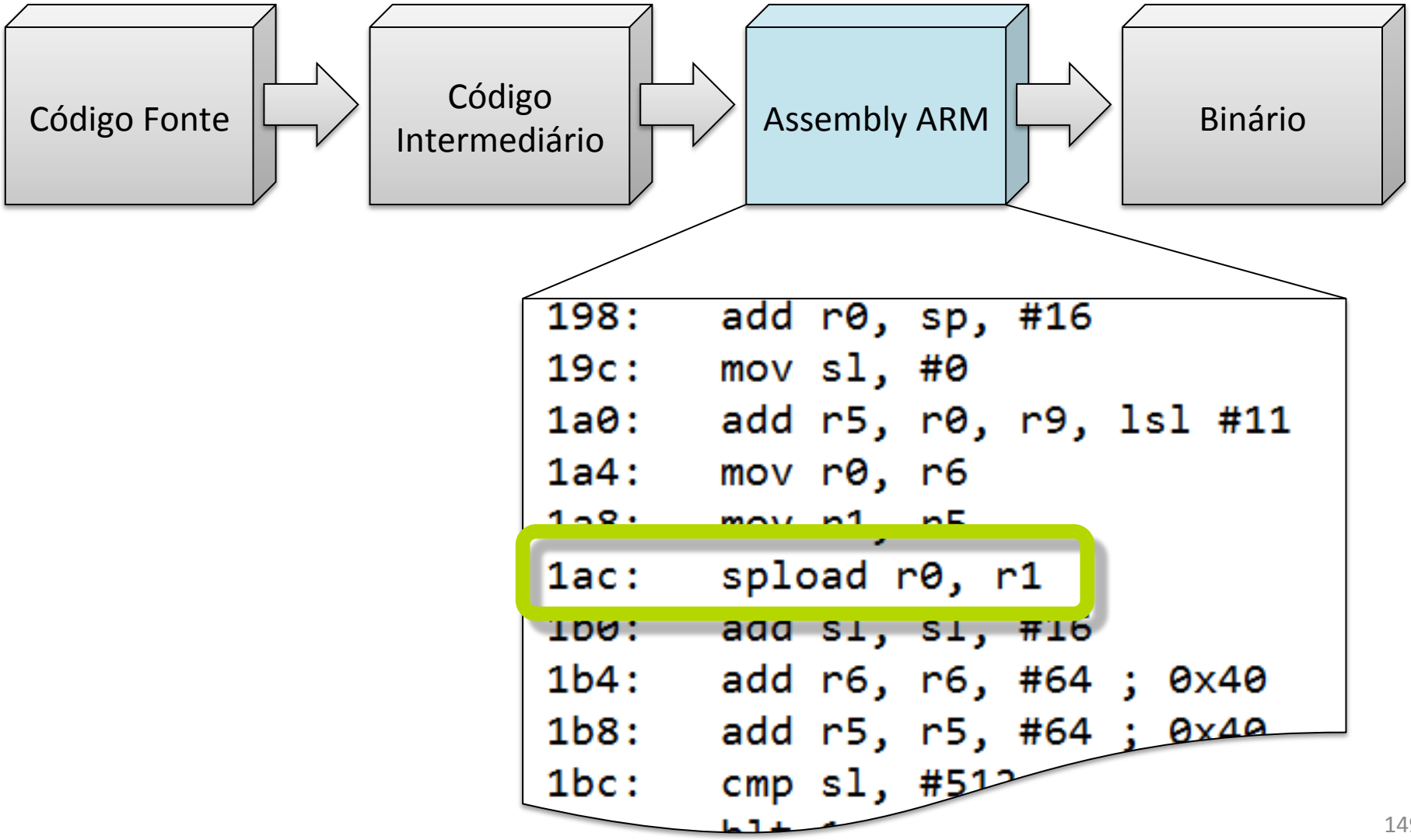

Código Fonte
Código Intermediário
Assembly ARM
Binário
...
int min = INFINITY;
for (int i = 0; i < SIZE; i++){
if (min > dist[i] && q[i]){
min = dist[i];
u = i;
}
}
spload((int)(line + i), &graph[u][i]);
// Remove u from q
q[u] = 0;

...Compilador

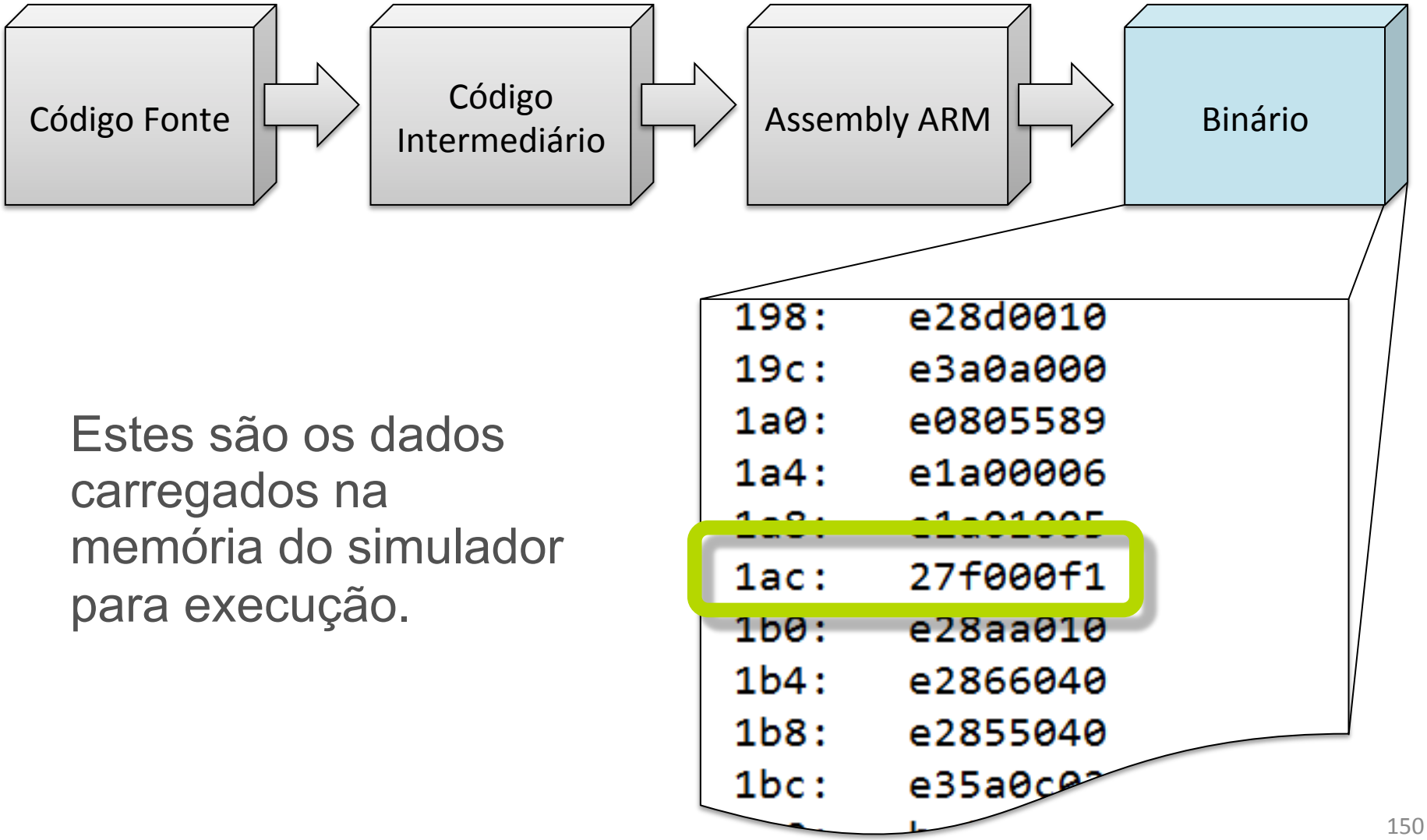



```
  br label %for.body51

for.body51:                                       ; pr
  %i48.0160 = phi i32 [ %add56, %for.body51 ], [ 0, %f
  %add.ptr52 = getelementptr inbounds i32* %13, i32 %i
  %16 = ptrtoint i32* %add.ptr52 to i32
  %arrayidx54 = getelementptr inbounds [512 x [512 x i
  call void @spload(i32 %16, i32* %arrayidx54) #1
  %add56 = add nsw i32 %i48.0160, 16
  %cmp50 = icmp slt i32 %add56
```

...Compilador


Código Fonte
Código Intermediário
Assembly ARM
Binário
198: add r0, sp, #16
19c: mov sl, #0
1a0: add r5, r0, r9, lsl #11
1a4: mov r0, r6
1ac: spload r0, r1
1b4: add r6, r6, #64 ; 0x40
1b8: add r5, r5, #64 ; 0x40
1bc: cmp sl, #512

...Compilador



Estes são os dados carregados na memória do simulador para execução.

Experimentos

Experimentos

Benchmark com quatro aplicações

- Adição de matrizes;
- Histograma;
- *Quicksort*;
- Algoritmo de Dijkstra.

...Experimentos

- Escritos em linguagem C;
- Reescritos para incluir a *scratchpad* usando a biblioteca criada;
- Múltiplas execuções, com
  - entradas distintas,
  - aleatórias
  - e de tamanhos diferentes.

...Experimentos

Mediram-se

- Tempo de execução em ciclos (eficiência);
- Tamanho do código gerado;
- Número de falhas (misses).

...Experimentos

Simulador foi configurado baseado em um processador real (Samsung Exynos 4210):

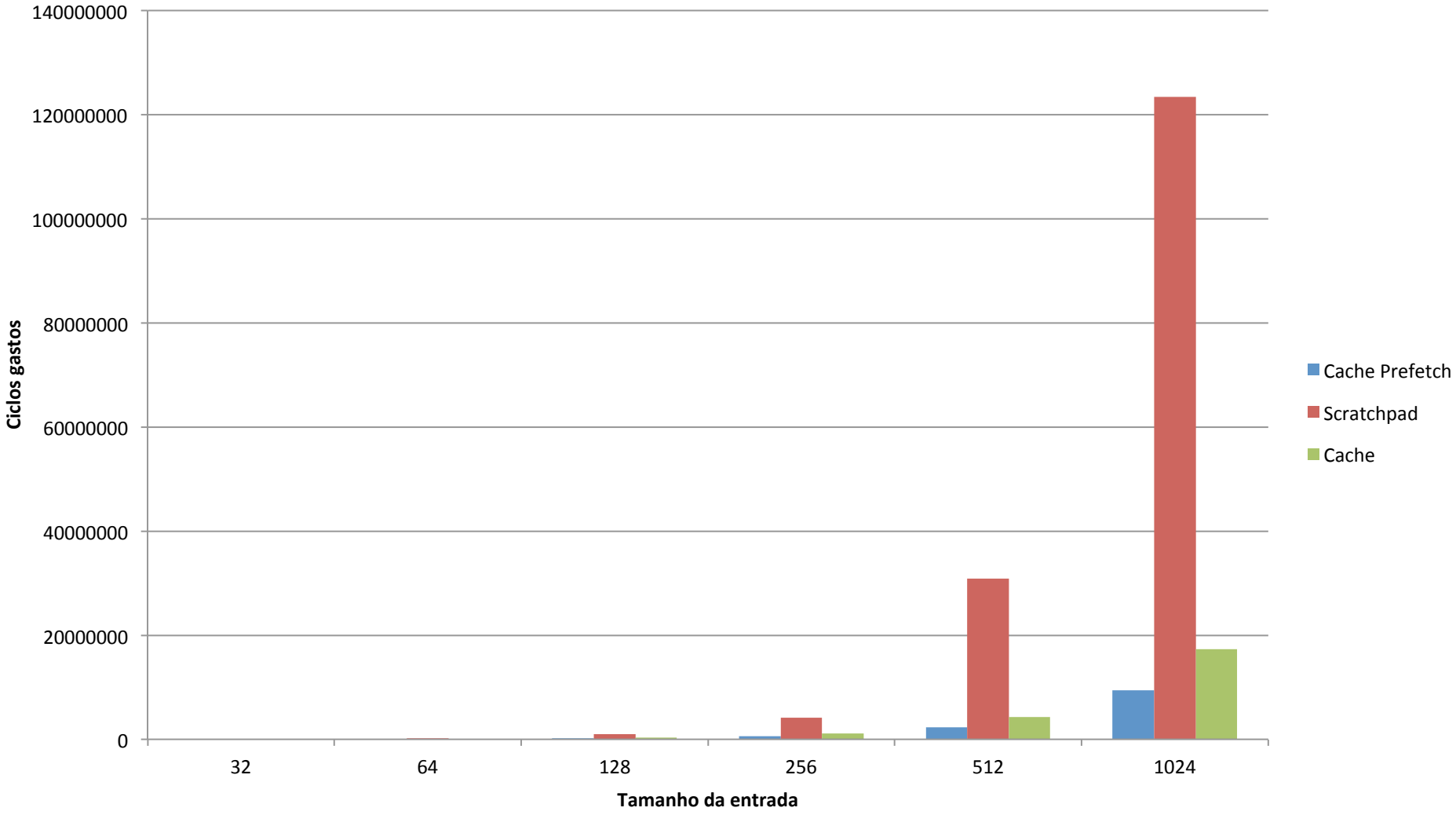
- 64 MB de memória principal;
- 32 KB de cache (256 B/line, 4-WAY e latência de 4 ciclos);
- 1 MB de cache nível 2 (256 B/line, 16-WAY e latência de 20 ciclos).

Adição de matrizes

Entrada	*Cache*	*Scratchpad*	Diferença	Desempenho ganho
32	103915	110758	-6843	-6,6%
64	38931	171681	-132750	-341,0%
128	151135	1023174	-872039	-577,0%
256	596733	4095411	-3498678	-586,3%
512	2372777	30846884	-28474107	-1200,0%
1024	9464114	123436090	-113971976	-1204,3%

Melhora observada pelo emprego da *scratchpad* e número de ciclos gastos pelo melhor caso da *cache* e da *scratchpad*.

...Adição de matrizes

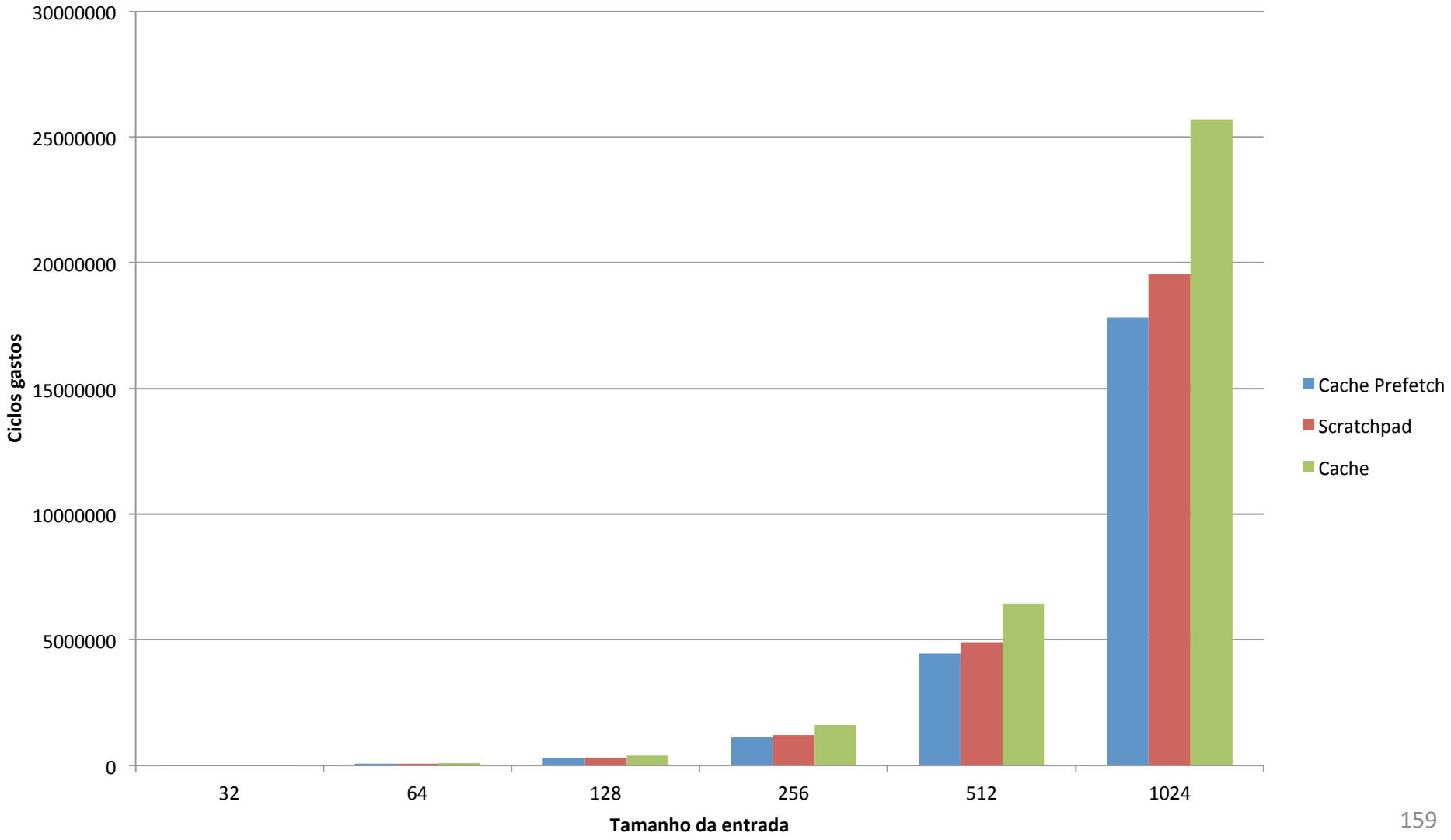


Histograma

Entrada	*Cache*	*Scratchpad*	Diferença	Desempenho ganho
32	18461	20677	-2216	-12,0%
64	70885	77383	-6498	-9,2%
128	280130	303505	-23375	-8,3%
256	1116480	1205985	-89505	-8,0%
512	4459560	4892343	-432783	-9,7%
1024	17832770	19549469	-1716699	-9,6%

Melhora observada pelo emprego da *scratchpad* e número de ciclos gastos pelo melhor caso da *cache* e da *scratchpad*.

...Histograma

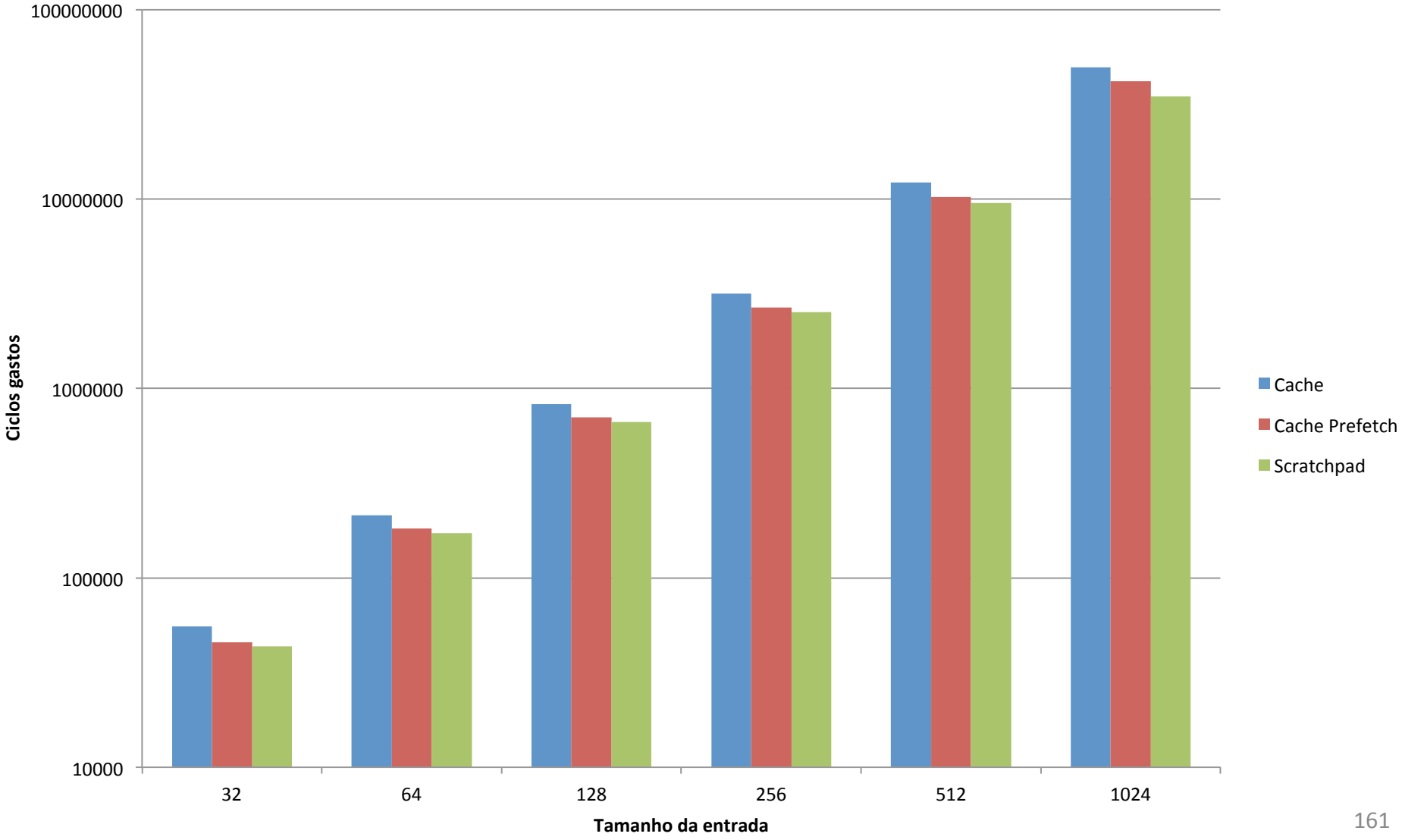

30000000
25000000
20000000
15000000
10000000
5000000
0
Ciclos gastos
32
64
128
256
512
1024
Tamanho da entrada
Cache Prefetch
Scratchpad
Cache

Algoritmo de Dijkstra

Entrada	*Cache*	*Scratchpad*	Diferença	Desempenho ganho
32	45750	43479	2271	5,0%
64	181688	172349	9339	5,1%
128	704233	665377	38856	5,5%
256	2683327	2525210	158117	5,9%
512	10259179	9513725	745454	7,3%
1024	41867728	34689802	7177926	17,1%

Melhora observada pelo emprego da *scratchpad* e número de ciclos gastos pelo melhor caso da *cache* e da *scratchpad*.

...Algoritmo de Dijkstra

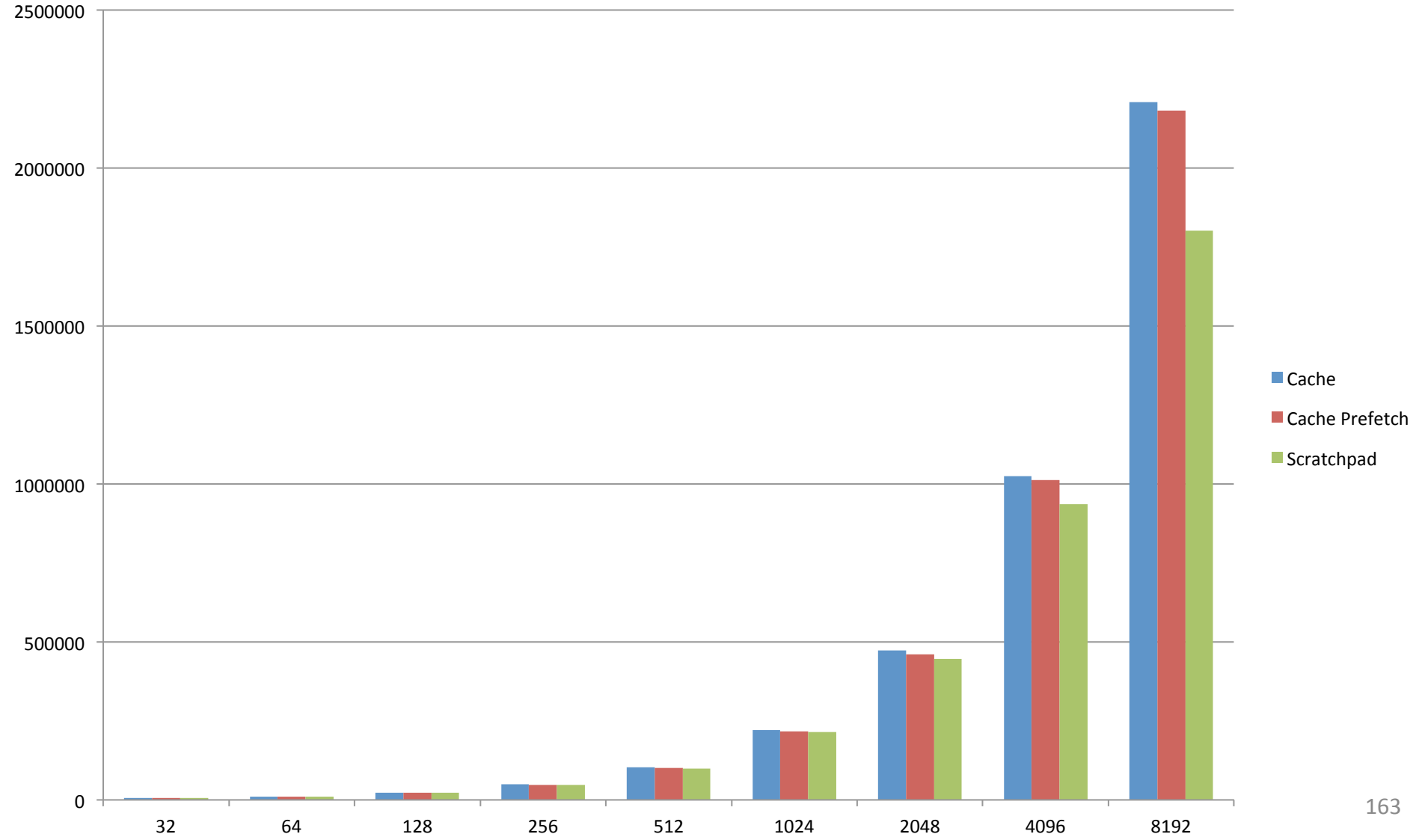


Quicksort

Entrada	*Cache*	*Scratchpad*	Diferença	Desempenho ganho
32	5291	5298	-7	-0,1%
64	10119	9980	139	1,4%
128	22138	22040	98	0,4%
256	47353	46971	382	0,8%
512	100971	99452	1519	1,5%
1024	217325	213771	3554	1,6%
2048	460109	450235	14874	3,2%
4096	1011502	975189	76313	7,5%
8192	2180744	1800756	379988	17,4%

Melhora observada pelo emprego da *scratchpad* e número de ciclos gastos pelo melhor caso da *cache* e da *scratchpad*.

...Quicksort


2500000
2000000
1500000
1000000
500000
0
32
64
128
256
512
1024
2048
4096
8192
Cache
Cache Prefetch
Scratchpad

Tamanho do código gerado

Tamanho da entrada	Matriz		
	Cache	**Scratchpad**	**Cresc.**
32	224	312	39%
64	124	225	81%
128	124	266	115%
256	124	387	212%
512	124	222	79%
1024	124	222	79%

Tamanho da entrada	Histograma		
	Cache	**Scratchpad**	**Cresc.**
32	97	155	60%
64	97	177	82%
128	97	166	71%
256	97	222	129%
512	97	142	46%
1024	97	142	46%

Número de linhas do código binário geradas pelo compilador de acordo com o tamanho da entrada para a *cache* e para *scratchpad*.

...Tamanho do código gerado

Tamanho da entrada

	Dijkstra		
	Cache	**Scratchpad**	**Cresc.**
32	219	247	13%
64	315	365	16%
128	157	183	17%
256	159	207	30%
512	159	183	15%
1024	159	191	20%

Tamanho da entrada

	Quicksort		
	Cache	**Scratchpad**	**Cresc.**
32	117	130	11%
64	117	140	20%
128	117	160	37%
256	118	201	70%
512	119	138	16%
1024	119	138	16%
2048	119	138	16%
4096	119	140	18%
8192	119	140	18%

Número de linhas do código binário geradas pelo compilador de acordo com o tamanho da entrada para a *cache* e para *scratchpad*.

Análise de resultados

Análise de resultados

Benchmark:

- Conjunto **representativo**;
- Algoritmos **úteis**;
- Que não fossem **demasiadamente complexos**.

...Análise de resultados

Histograma e adição de matrizes:

- Apresentam alta localidade espacial;
- Percorrem matrizes sequencialmente.

...Análise de resultados

Histograma e adição de matrizes:

• O compilador lineariza as matrizes de modo que seus acessos resultem em **acertos** em um *hardware* com *prefetching*.

...Análise de resultados

Falhas		
Entrada	Cache	Cache Prefetch
32	129	2
64	515	3
128	2051	3
256	8195	3
512	32771	3
1024	131075	3

Essa vantagem fica evidenciada na tabela ao lado para o problema da adição de matrizes.

Somente as falhas compulsórias ocorrem.

...Análise de resultados

Misses		
Entrada	Cache	Cache Prefetch
32	81	8
64	273	8
128	1041	8
256	4113	8
512	16401	8
1024	65553	8

Tabela para o histograma.

Somente as falhas compulsórias ocorrem.

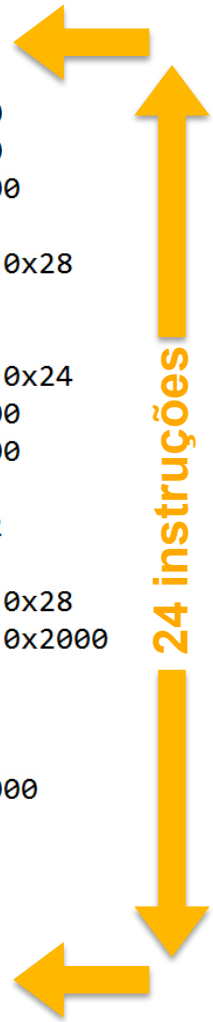
...Análise de resultados

Histograma e adição de matrizes:

- A *scratchpad* precisaria superar um dos melhores casos da *cache*: a **altíssima localidade espacial** desses testes;
- Além disso, é necessário compensar o número de instruções menor.

...Análise de resultados

```
210:    spload r0, r1
214:    add r7, r7, #16
218:    add r5, r5, #64 ; 0x40
21c:    add r6, r6, #64 ; 0x40
220:    cmp r7, #1024   ; 0x400
224:    blt 208 <main+0x208>
228:    ldr r0, [sp, #40]   ; 0x28
22c:    mov r1, #0
230:    mov r2, r4
234:    str r4, [sp, #36]   ; 0x24
238:    cmp r0, #1024   ; 0x400
23c:    mov r0, #1024   ; 0x400
240:    moveq   r0, #0
244:    add ip, r9, r0, lsl #2
248:    mov r9, #12
24c:    str r0, [sp, #40]   ; 0x28
250:    orr r9, r9, #8192   ; 0x2000
254:    mov r3, ip
258:    ldr lr, [r3, r1]!
25c:    add r1, r1, #16
260:    cmp r1, #4096   ; 0x1000
264:    ldr r6, [r3, r9]
268:    ldr r0, [r3, #12]
26c:    ldr r5, [r3, #4]
270:    ldr r4, [r3, #8]
274:    ldr r7, [r3, r8]
```



A **quantidade de instruções** entre o *spload* e o uso dos dados é **muito pequena**.

...Análise de resultados

Possíveis soluções para os dois problemas:

- Carregar os dados com grande antecedência;
- Carregar as matrizes direto na *scratchpad* na inicialização.

...Análise de resultados

Histograma:

- O algoritmo povoa sequencialmente **todas as posições da *cache***;
- A *scratchpad* restringe-se a um **pequeno espaço** – um efeito de **compactação**;

...Análise de resultados

Essa diferença poderia ser extremamente útil para uma abordagem paralela para **evitar colisões**;

O espaço utilizado por cada *thread* seria bem definido.

...Análise de resultados

Algoritmo de Dijkstra:

- Nó de menor distância da lista de nós pendentes;
- No caso geral **não há alta localidade espacial**.

...Análise de resultados

Algoritmo de Dijkstra:

- **Poucas instruções** entre o *sploa*d e o uso do endereço na *scratchpad*;
- Observou-se que a *scratchpad é* melhor para **problemas maiores**;
- Com entradas maiores:
  - o **número de instruções aumenta neste intervalo**;
  - o **número de falhas** da *cache* **aumenta**.

...Análise de resultados

Quicksort:

- Dados representados como um vetor;
- **Chamadas recursivas** e **grandes entradas**: complicam para *prefetching*;
- O vetor é completamente colocado na *scratchpad*.

...Análise de resultados

Tamanho do código

- No pior caso o código cresceu até 212%;
- Otimizações do compilador (-O3) – *loop unrolling*, etc..

...Análise de resultados

Tamanho da entrada

	Matriz		
	Cache	Scratchpad	Cresc.
32	224	312	39%
64	124	225	81%
128	124	266	115%
256	124	387	[illegible]
512	124	222	79%
1024	124	222	79%

Tamanho da entrada

	Histograma		
	Cache	Scratchpad	Cresc.
32	97	155	60%
64	97	177	82%
128	97	166	71%
256	97	222	[illegible]
512	97	142	46%
1024	97	142	46%

O tamanho do código se **estabiliza** para entradas maiores.

...Análise de resultados

Tamanho da entrada

	Matriz		
	Cache	Scratchpad	Cresc.
32	224	312	39%
64	124	225	81%
128	124	266	115%
256	124	387	212%
512	124	222	79%
1024	124	222	79%

Tamanho da entrada

	Histograma		
	Cache	Scratchpad	Cresc.
32	97	155	60%
64	97	177	82%
128	97	166	71%
256	97	222	129%
512	97	142	46%
1024	97	142	46%

Para casos pequenos o código sem a *scratchpad* também apresenta **variações** no número de instruções.

...Análise de resultados

Tamanho da entrada

	Dijkstra		
	Cache	Scratchpad	Cresc.
32	219	247	13%
64	315	365	16%
128	157	183	17%
256	159	207	30%
512	159	183	15%
1024	159	191	20%

Tamanho da entrada

	Quicksort		
	Cache	Scratchpad	Cresc.
32	117	130	11%
64	117	140	20%
128	117	160	37%
256	118	201	70%
512	119	138	16%
1024	119	138	16%
2048	119	138	16%
4096	119	140	18%
8192	119	140	18%

Para casos pequenos o código sem a *scratchpad* também apresenta **variações** no número de instruções.

...Análise de resultados

Tamanho da entrada

	Dijkstra		
	Cache	Scratchpad	Cresc.
32	219	247	13%
64	315	365	16%
128	157	183	17%
256	159	207	30%
512	159	183	15%
1024	159	191	20%

Tamanho da entrada

	[illegible]		
	Cache	Scratchpad	Cresc.
32	117	130	11%
64	117	140	20%
128	117	160	37%
256	118	201	70%
512	119	138	16%
1024	119	138	16%
2048	119	138	16%
4096	119	140	18%
8192	119	140	18%

Exemplo mais evidente:
variação para a *scratchpad*.

...Análise de resultados

Tamanho da entrada	Dijkstra		
	Cache	Scratchpad	Cresc.
32	219	247	13%
64	315	365	16%
128	157	183	17%
256	159	207	30%
512	159	183	15%
1024	159	191	20%

Tamanho da entrada	Quicksort		
	Cache	Scratchpad	Cresc.
32	117	130	11%
64	117	140	20%
128	117	160	37%
256	118	201	70%
512	119	138	16%
1024	119	138	16%
2048	119	138	16%
4096	119	140	18%
8192	119	140	18%

Tamanho do código gerado **estável** para a *scratchpad*.

...Análise de resultados

Após estabilizado, o tamanho do código:

Tamanho do Código Gerado			
Teste	**Cache**	**Scratchpad**	**Crescimento**
Matriz	124	222	79%
Histograma	97	142	46%
Dijkstra	159	191	20%
Quicksort	119	140	18%

...Análise de resultados

Tamanho do Código Gerado			
Teste	Cache	Scratchpad	Crescimento
Matriz	124	222	79%
Histograma	97	142	46%
Dijkstra	159	191	20%
Quicksort	119	140	18%

Resultado **pior**:

- **Crescimento mais expressivo**;
- Exemplos com **maior localidade**;

...Análise de resultados

Tamanho do Código Gerado			
Teste	**Cache**	**Scratchpad**	**Crescimento**
Matriz	124	222	79%
Histograma	97	142	46%
Dijkstra	159	191	20%
Quicksort	119	140	18%

Resultado **melhor**:

- **Menor crescimento do código**;
- Exemplos com **menor localidade**;

Contribuições

Contribuições

- Memória *scratchpad* **transparente**:
  - código legado preservado;
  - aplicações tradicionais – caso geral;
  - aplicações com padrão de acesso complexo.

...Contribuições

- Simulador de hardware com a arquitetura mista;
- Geração de código para a arquitetura proposta;
- Análise e avaliação do ambiente apresentado.

Conclusão

Conclusão

- ***Cache*** é melhor para casos com **mais localidade**;
- ***Scratchpad*** pode melhorar o desempenho das aplicações em que **localidade não é evidente**;
- Com a arquitetura mista proposta ambos podem ser utilizados para cada caso;

...Conclusão

- Problemas maiores podem implicar em resultados melhores;
- *Scratchpad* adiciona bastante **complexidade ao desenvolvimento**: a gestão da memória **abstraída** pela ***cache*** é **repassada** ao desenvolvedor;

...Conclusão

- A *scratchpad* deve ser utilizada para casos
  - críticos onde a *cache* é **muito ineficiente**: soluções paralelas com **muitos conflitos**, **localidade pouco evidente**, etc.;
  - onde **eficiência é muito relevante**: aplicações de tempo real, processamento multimídia, etc.;

...Conclusão

Este trabalho apresenta uma solução para a produção de um **ambiente diversificado**, que **concilia** as **demandas de eficiência** com as de **retrocompatibilidade**, possibilitando atender de forma **eficiente** um **maior número de aplicações** com requisitos distintos.

Trabalhos Futuros

- Compilação com atribuição automática dos dados a memória *scratchpad;*
- Gestão automática dos endereços da *scratchpad;*
- Associação da instrução *spload* a *prefetching*;
- Avaliação de aplicações mais complexas e com entradas maiores.

Trabalhos Relacionados

Trabalhos Relacionados

Panda, P. R.; Dutt, N. D. & Nicolau, A. (1997). Efficient utilization of scratch-pad memory in embedded processor applications. Proceedings of the 1997 European Design and Test Conference.

Um dos primeiros trabalhos relacionados a memória *scratchpad*, apresentando-a com uma solução para diversos requisitos das arquiteturas embarcadas.

...Trabalhos Relacionados

Williams, S.; Shalf, J.; Oliker, L.; Kamil, S.; Husbands, P. & Yelick, K. (2007). Scientific computing kernels on the cell processor. International Journal of Parallel Programming.

Os autores avaliam o uso do processador *cell* para computação científica comparando com alternativas existentes. Para os autores a hierarquia de memória gerida por *hardware,* embora exponha muita complexidade, resultou em acelerações de 2 a 150 vezes nos experimentos.

...Trabalhos Relacionados

Francesco, P.; Marchal, P.; Atienza, D.; Benini, L.; Catthoor, F. & Mendias, J. M. (2004). An integrated hardware/software approach for run-time scratchpad management. DAC '04 Proceedings of the 41st annual Design Automation Conference.

Expõe as desvantagens de uma alocação estática da *scratchpad* e apresenta uma abordagem para a sua gestão em tempo de execução.

...Trabalhos Relacionados

Cook, H.; Asanovi, K. & Patterson, D. A. (2009). Virtual local stores: Enabling software-managed memory hierarchies in mainstream computing environments. UCB/EECS-2009-131, Electrical Engineering and Computer Sciences. University of California at Berkeley.

Apresenta uma abordagem para a introdução da *scratchpad* em processadores de propósito geral.

...Trabalhos Relacionados

Hiser, J. D. & Davidson, J. W. (2004). Embarc: An Efficient memory bank assignment algorithm for retargetable compilers. SIGPLAN Not., 39(7): 182--191. ISSN 0362-1340.

Utilizando *profiling* apresenta-se um algoritmo de compilação que tenta atribuir automaticamente diferentes porções de dados a estruturas heterogêneas de memória.

...Trabalhos Relacionados

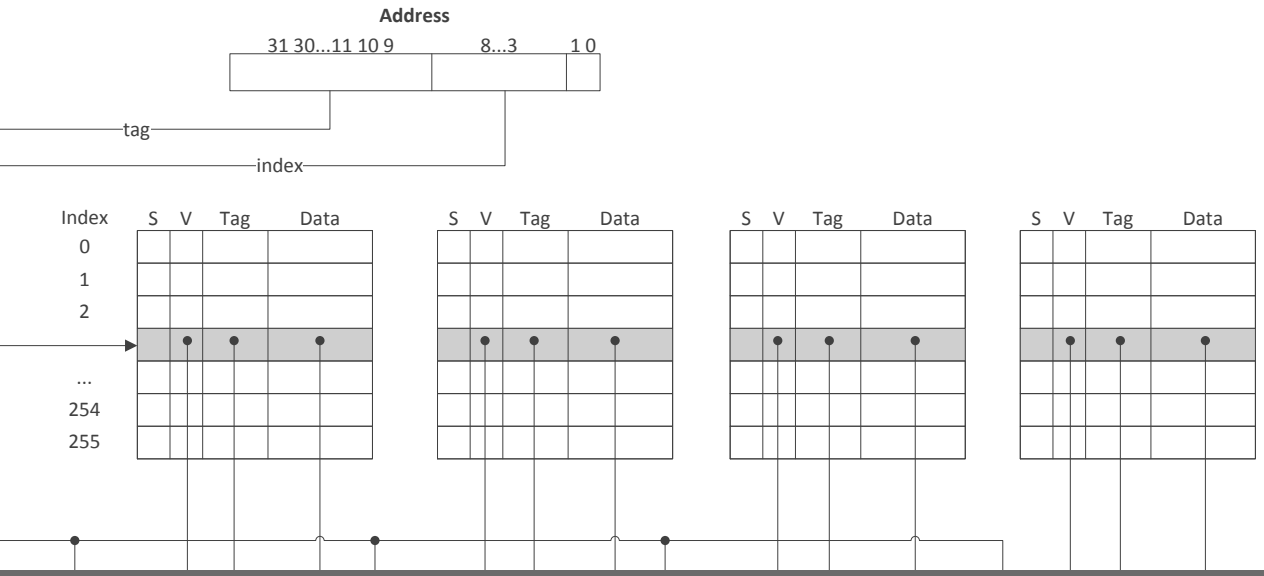
Nguyen, N.; Dominguez, A. & Barua, R. (2005). Memory allocation for embedded systems with a compile-time-unknown scratch-pad size. pp. 115--125.

Como o tamanho da *scratchpad* pode variar entre diferentes arquiteturas, propõe-se um pré-processador que resolve a alocação de dados da *scratchpad* antes da primeira execução da aplicação na máquina alvo.

...Trabalhos Relacionados

Udayakumaran, S. & Barua, R. (2003). Compiler-decided dynamic memory allocation for scratch-pad based embedded systems. Em Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES '03, pp. 276--286, New York, NY, USA. ACM.

Apresentam uma abordagem de compilação que permite a gestão de variáveis globais e da pilha utilizando a *scratchpad*.



Um Modelo Transparente de Memória *Scratchpad* para Arquiteturas de Propósito Geral

Felipe Silva Loredo
Orientadora: Mariza Andrade da Silva Bigonha
Coorientador: Claudionor José Nunes Coelho Junior

```
%16 = ptrtoint i32* %add.ptr52 to i32
%arrayidx54 = getelementptr inbounds [512 x [512 x i32]]* %graph, i32 0
call void @spload(i32 %16, i32* %arrayidx54) #1
%add56 = add nsw i32 %i48.0160, 16
```