

Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

---

# Estrutura de Dados

## Tabelas Hash

(Baseado no material de Michael Main e  
Walter Savitch)

▶ Tiago Maritan

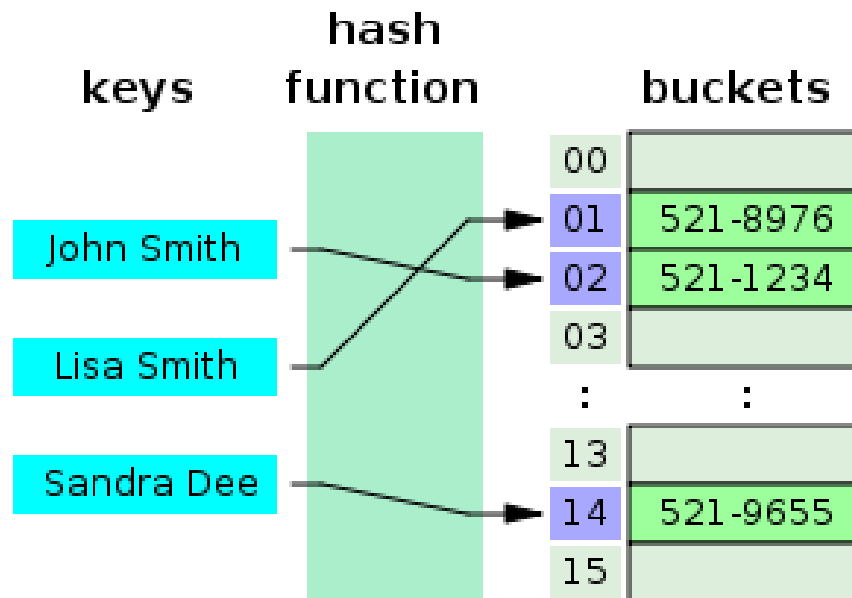
▶ [tiago@ci.ufpb.br](mailto:tiago@ci.ufpb.br)

---

# Tabelas Hash (Hash Tables)

---

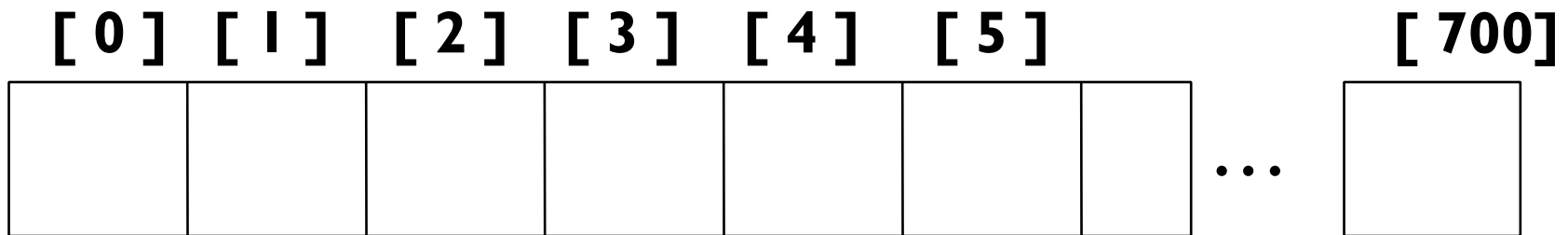
- ▶ ED que permite realizar **busca eficiente** em dados
  - ▶ A partir de uma **chave simples**, faz uma busca rápida para obter o valor desejado
- ▶ Associa **chaves de pesquisa** a **valores (dados)**



# Tabelas Hash

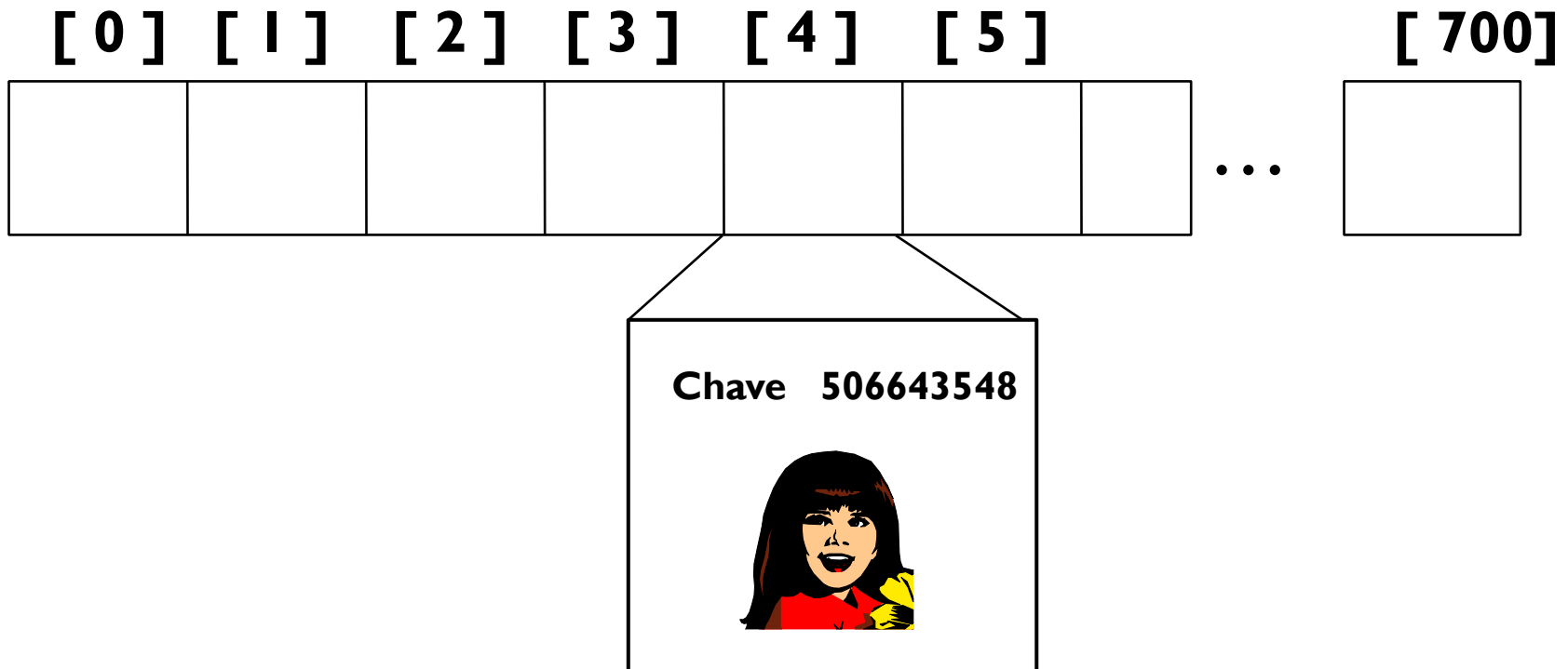
---

- ▶ Implementação de **arrays associativos**
  - ▶ Conhecidos como **maps, dictionaries**
- ▶ Ex: Array de 701 registros (estruturas)



# Tabelas Hash

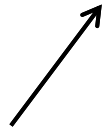
- ▶ Cada **estrutura/registro** tem uma **chave (campo especial)**
  - ▶ Geralmente são **chaves numéricas**;
  - ▶ Ex: Número do RG, matrícula;



# Inserção em Tabelas Hash

---





- ▶ Na inserção, a **chave** deve ser convertida para algum **índice inteiro no array**.
- ▶ A função que faz a conversão é chamada de **função hash**.
  - ▶ Índice gerado é chamado de **valor hash** da chave.
  - ▶ Representa a posição onde os dados serão inseridos
- ▶ Ex: Função hash: 
$$h(x) = x \% 701$$



Funções hash geralmente usam  
uso de resto da divisão e envolvem o tamanho da tabela

# Inserção em Tabelas Hash

- ▶ Ex: A chave 580625685 seria inserida em que posição?

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]		[ 700 ]
							






$$h(580625685) = 580625685 \% 701$$

$$h(580625685) = 3$$

Chave deve ser inserida na posição 3 do array.

# Inserção em Tabelas Hash

- ▶ Ex: A chave 580625685 seria inserida em que posição?

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]		[ 700 ]
							

$$h(580625685) = 580625685 \% 701$$

$$h(580625685) = 3$$

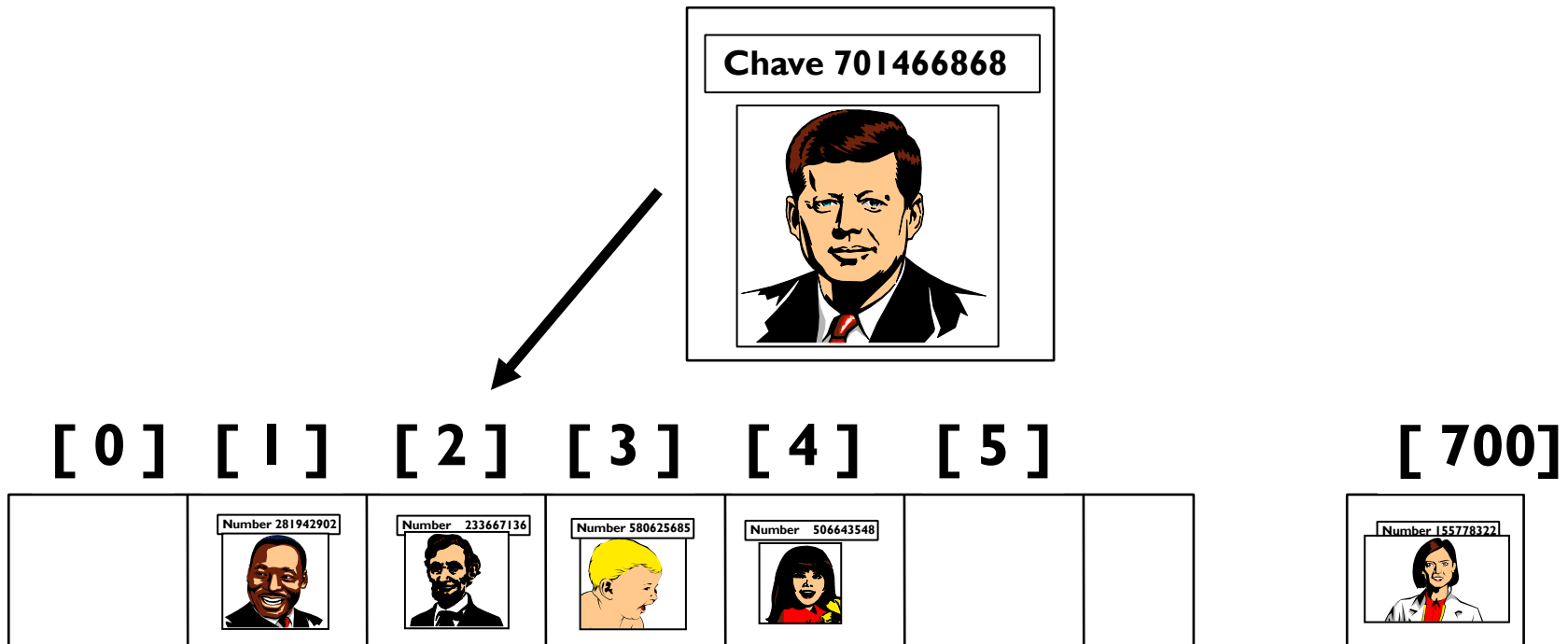
Chave deve ser inserida na posição **3** do array.

# Colisões

- ▶ Ex: Suponha agora que desejamos inserir a chave 701466868?

$$h(701466868) = 2$$

- ▶ Ocorreu uma **colisão**! Como resolver?





# Tratamento de Colisões

---

- ▶ Duas estratégias principais:
  - ▶ Reespalhamento (ou Endereçamento Aberto)
  - ▶ Encadeamento

# Reespalhamento (ou Endereçamento Aberto)

---

- ▶ Aplica uma **função de espalhamento secundária** sobre a chave.
  - ▶ Aplicada sucessivamente até uma posição vazia ser encontrada.

- ▶ **Modelo geral das funções de reespalhamento:**

$$\text{rh}(i) = (i + c) \% \text{tamanho}$$

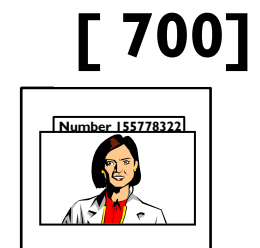
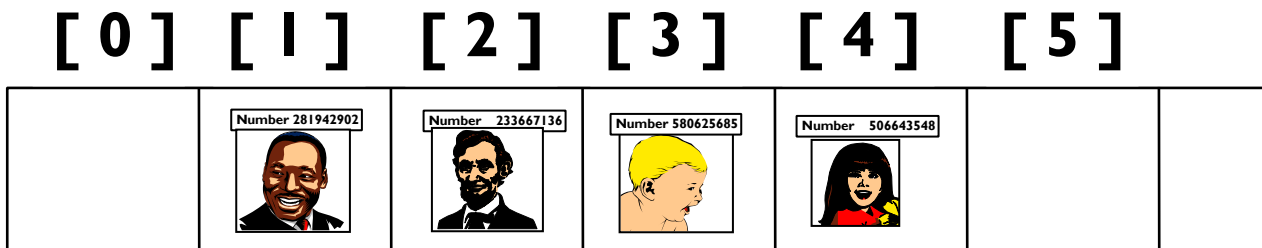
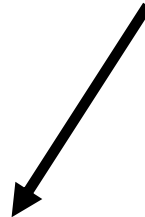
constante

tamanho  
da tabela hash

- ▶ **Ex: Teste linear**
  - ▶ Coloca o registro na próxima posição vazia disponível.
  - ▶ Função de reespalhamento:  $\text{rh}(i) = (i+1) \% 701$

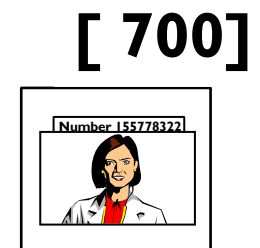
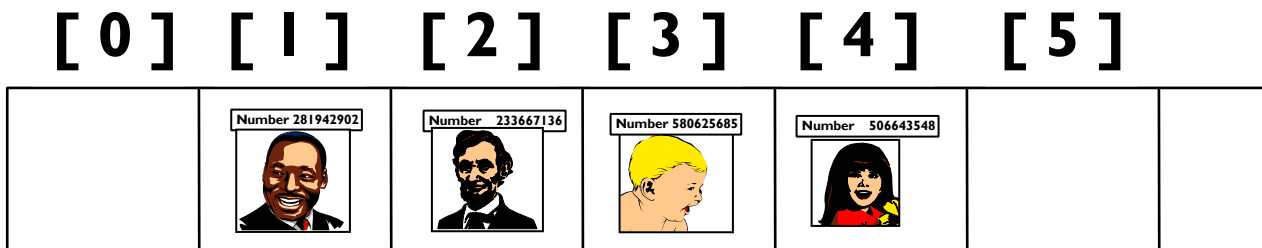
# Reespalhamento (ou Endereçamento Aberto)

Próxima posição está  
ocupada, reespalha



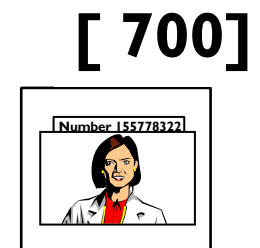
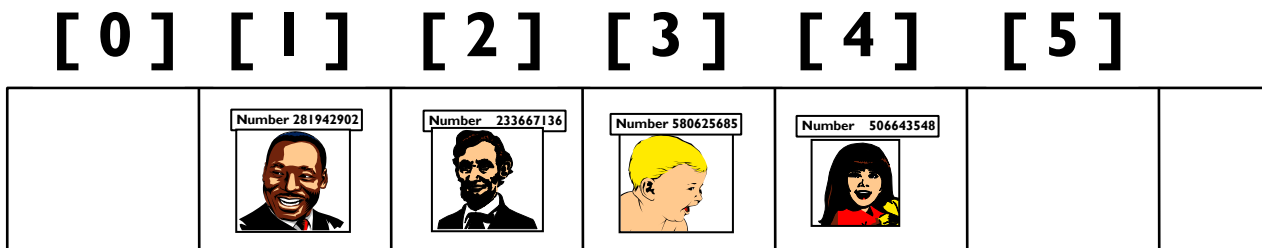
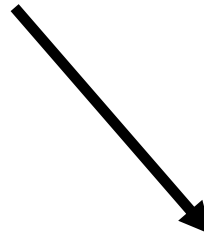
# Reespalhamento (ou Endereçamento Aberto)

**Próxima posição  
continua ocupada,  
reespalha novamente...**



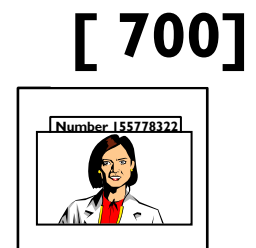
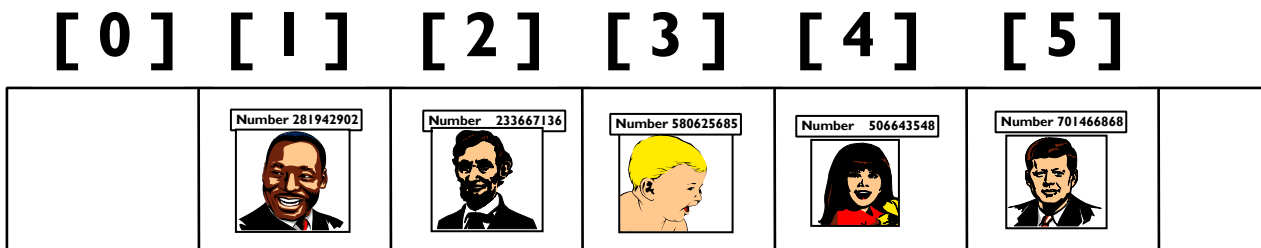
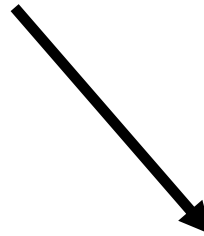
# Reespalhamento (ou Endereçamento Aberto)

**Posição vazia! Insere o registro na posição [5].**



# Reespalhamento (ou Endereçamento Aberto)

**Posição vazia! Insere o registro na posição [5].**



# Pesquisando Elementos em Tabelas Hash

---

- ▶ Um dos benefícios de uso de Tabelas Hash é que a busca geralmente é **muito eficiente**.
  - ▶ Especialmente quando há poucas colisões.
- ▶ Pseudocódigo da Pesquisa:
  1. Calcula o valor hash da chave pesquisada;
  2. Verifica a posição indicada pelo valor hash no array;
  3. Se não for a chave pesquisada, aplique função de reespalhamento até encontrar a chave ou uma posição vazia.

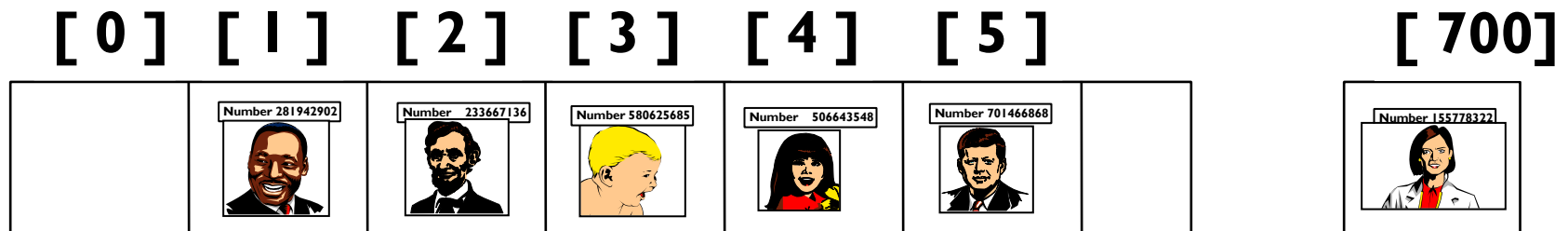
# Pesquisando Elementos em Tabelas Hash

- ▶ Ex: Pesquisar a chave 701466868:

$$h(701466868) = (701466868 \% 701)$$

$$h(701466868) = 2$$

**Não é o elemento procurado,  
reespalha**





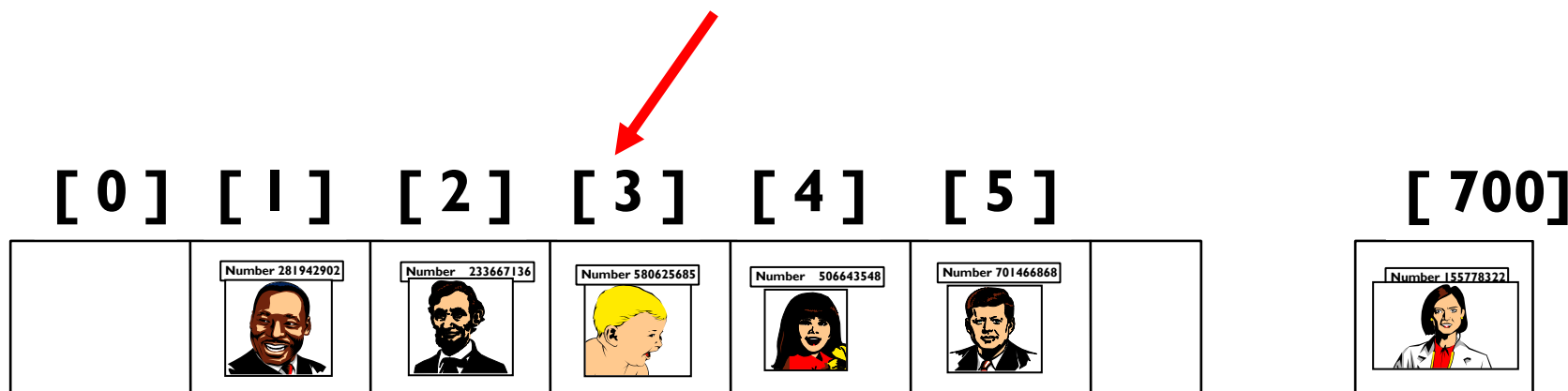
# Pesquisando Elementos em Tabelas Hash

- ▶ Ex: Pesquisar a chave 701466868:

$$h(701466868) = (701466868 \% 701)$$

$$h(701466868) = 2$$

**Não é o elemento procurado, e posição não é vazia então reespalha novamente**



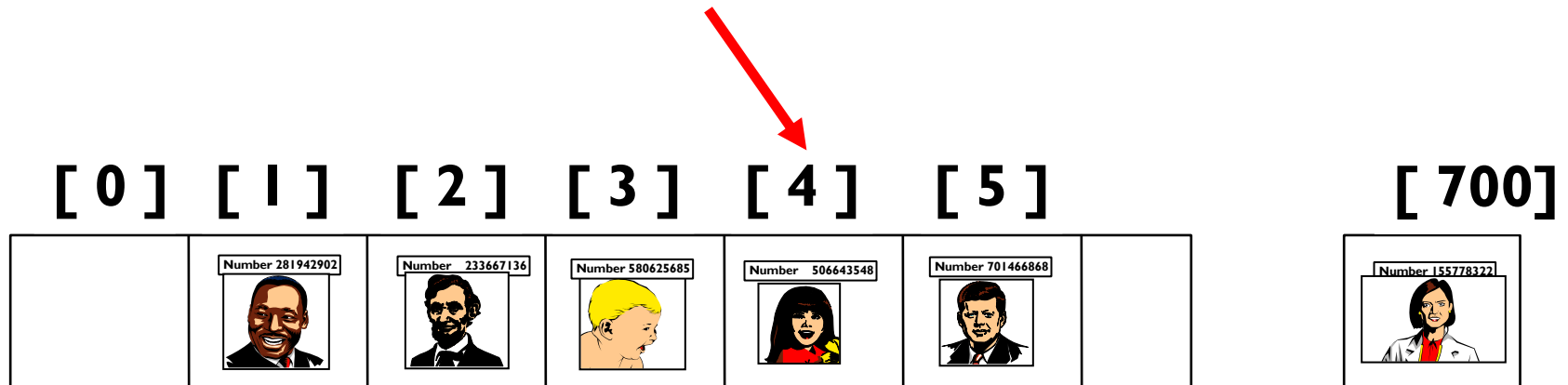
# Pesquisando Elementos em Tabelas Hash

- ▶ Ex: Pesquisar a chave 701466868:

$$h(701466868) = (701466868 \% 701)$$

$$h(701466868) = 2$$

**Não é o elemento procurado, e posição não é vazia então reespalha novamente**



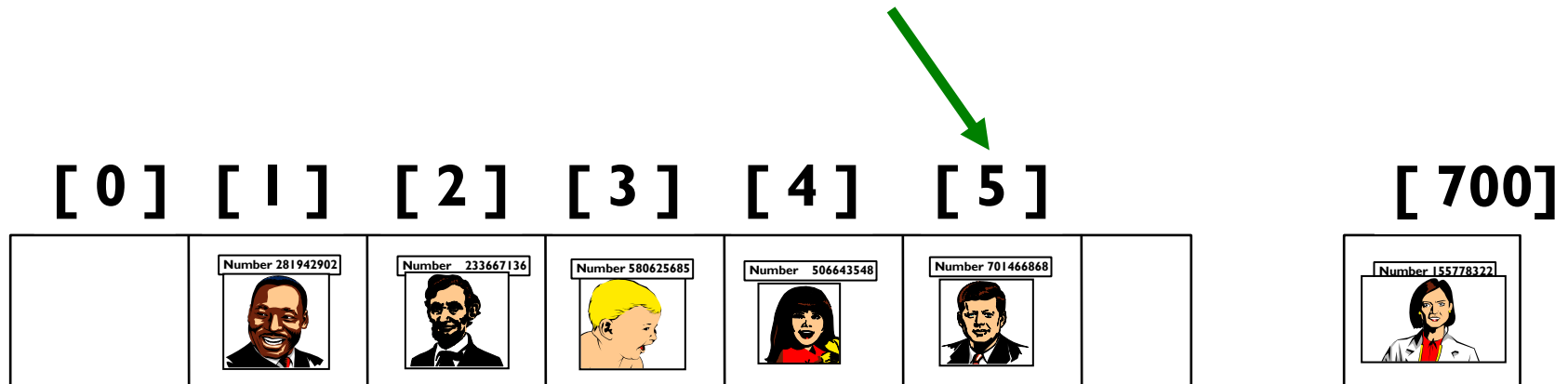
# Pesquisando Elementos em Tabelas Hash

- ▶ Ex: Pesquisar a chave 701466868:

$$h(701466868) = (701466868 \% 701)$$

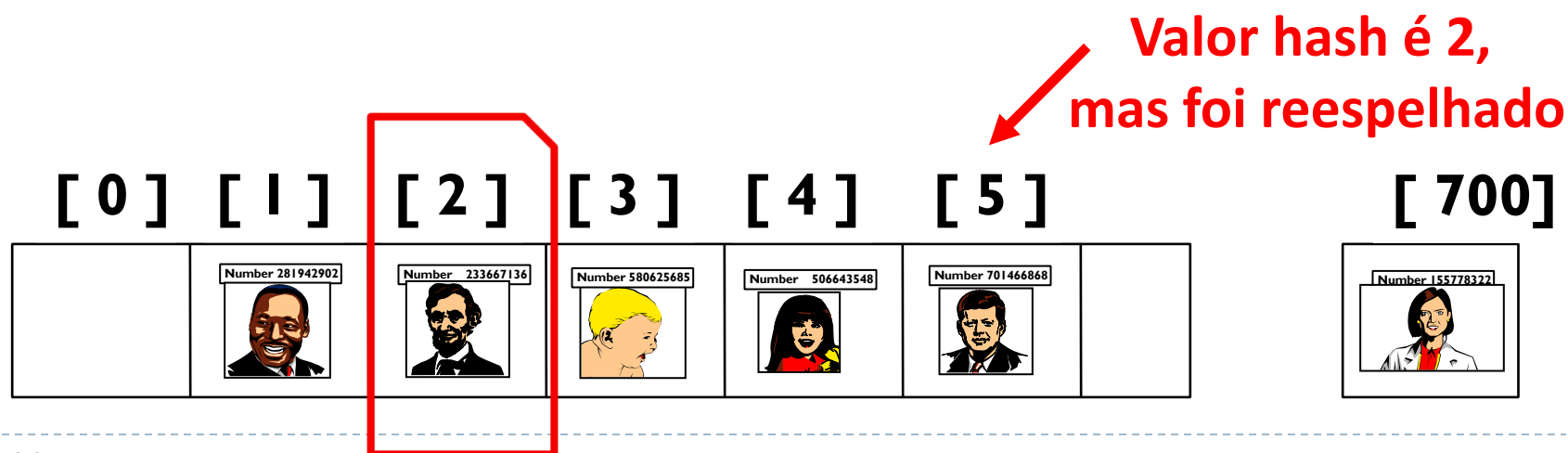
$$h(701466868) = 2$$

Localizou o elemento procurado,  
retorna o registro



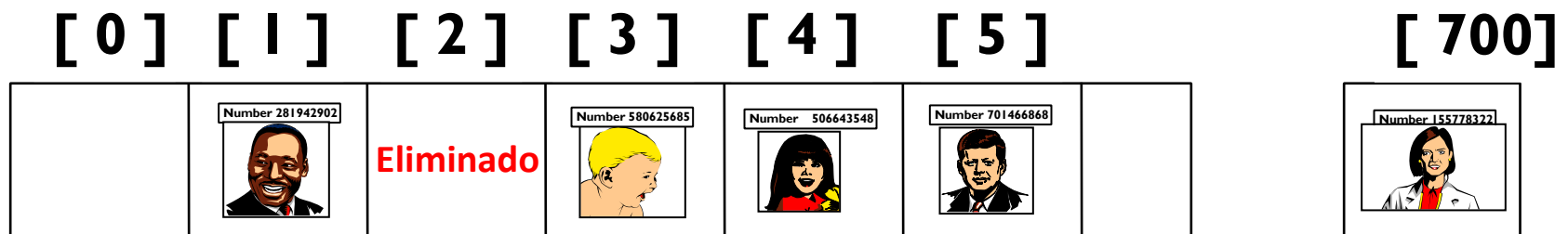
# Remoção de Elementos em Tabelas Hash

- ▶ Difícil eliminar itens em uma Tabela Hash que usa reespalhamento na busca e inserção.
- ▶ Ex: Remoção no elemento [2]
  - ▶ Pode interferir nos elementos que foram reespalhados;
  - ▶ Inviabilizaria a localização do elemento [5] (reespalhado).



# Remoção de Elementos em Tabelas Hash

- ▶ Solução: Marcar a posição como “Eliminado”.
- ▶ Algoritmo de busca deve continuar pesquisando quando encontrar uma posição “Eliminado”.
- ▶ Problemático quando há muitas eliminações
  - ▶ Desperdiça espaço.



# Encadeamento Separado

---

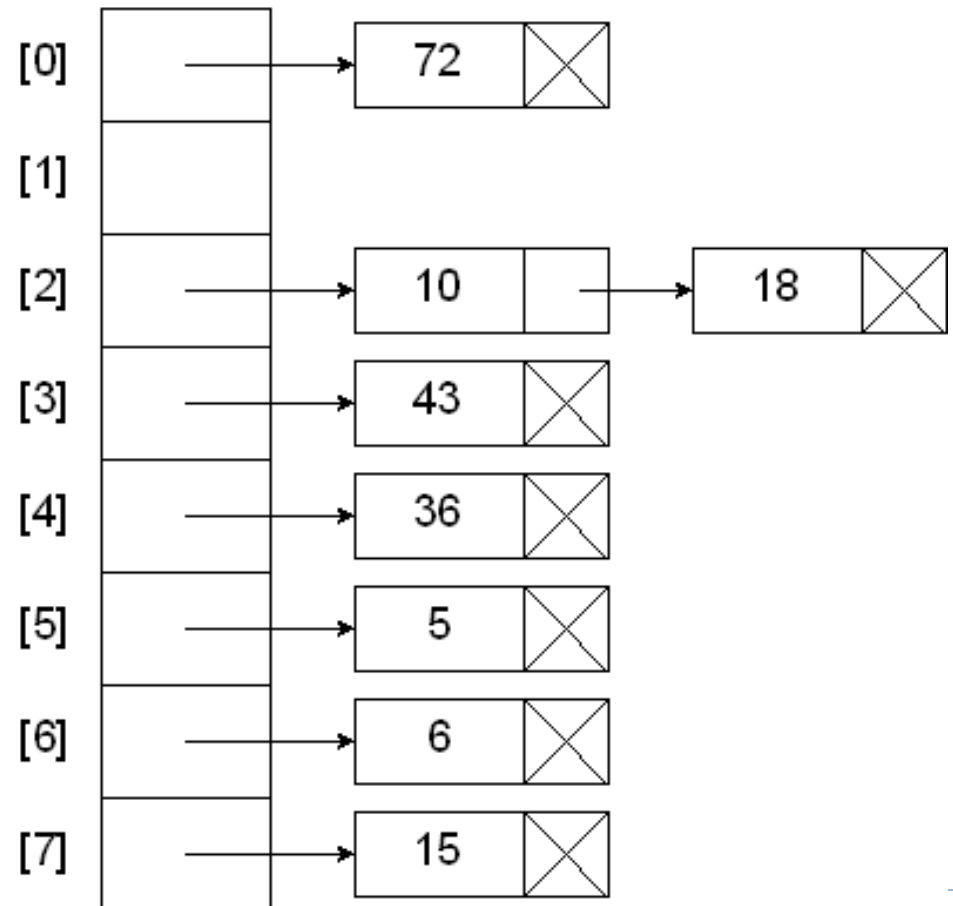
- ▶ Reespalhamento assume que tamanho da tabela é fixa.
- ▶ Se  $n^{\circ}$  elementos  $>$  tamanho\_tabela, saída é:
  - ▶ Alocar uma tabela maior e recalcular o valor da chave de todos os registros já inseridos.
- ▶ Outro método pra tratar colisões, que não usa tabela de tamanho fixo, é o **encadeamento separado**.

# Encadeamento Separado

- Mantém uma lista encadeada para cada conjunto de chaves que colidem.

Hash key = key % table size

4	=	36	%	8
2	=	18	%	8
0	=	72	%	8
3	=	43	%	8
6	=	6	%	8
2	=	10	%	8
5	=	5	%	8
7	=	15	%	8



# Características das Tabelas Hash

---

## ▶ Vantagens:

- ▶ Muito eficiente para pesquisa
  - ▶ Inclusive em dados não ordenados;
- ▶ Fácil implementação;

## ▶ Desvantagens:

- ▶ Demanda mais armazenamento;
- ▶ Pode desperdiçar memória
  - ▶ Demandar mais memória que o necessário;
- ▶ Remoção “problemática”;



Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

---

# Estrutura de Dados

## Tabelas Hash

(Baseado no material de Michael Main e  
Walter Savitch)

▶ Tiago Maritan

▶ [tiago@ci.ufpb.br](mailto:tiago@ci.ufpb.br)

---

