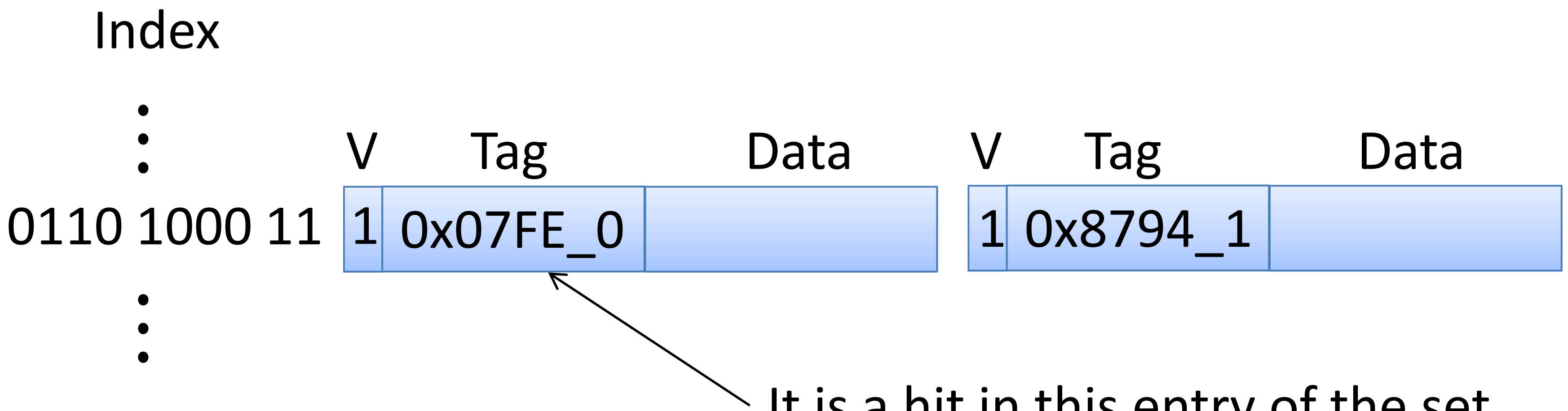
# Tree-based LRU Approximation

José Nelson Amaral

# 2-way set associative caches

1st Memory reference:  0x07FE 3460

$$\underbrace{0000\ 0111\ 1111\ 1110\ 0}_{\text{Tag = 0x07FE\_0}}\underbrace{011\ 0100\ 011}_{\text{Index}}\underbrace{0\ 0000}_{\text{Offset}}$$



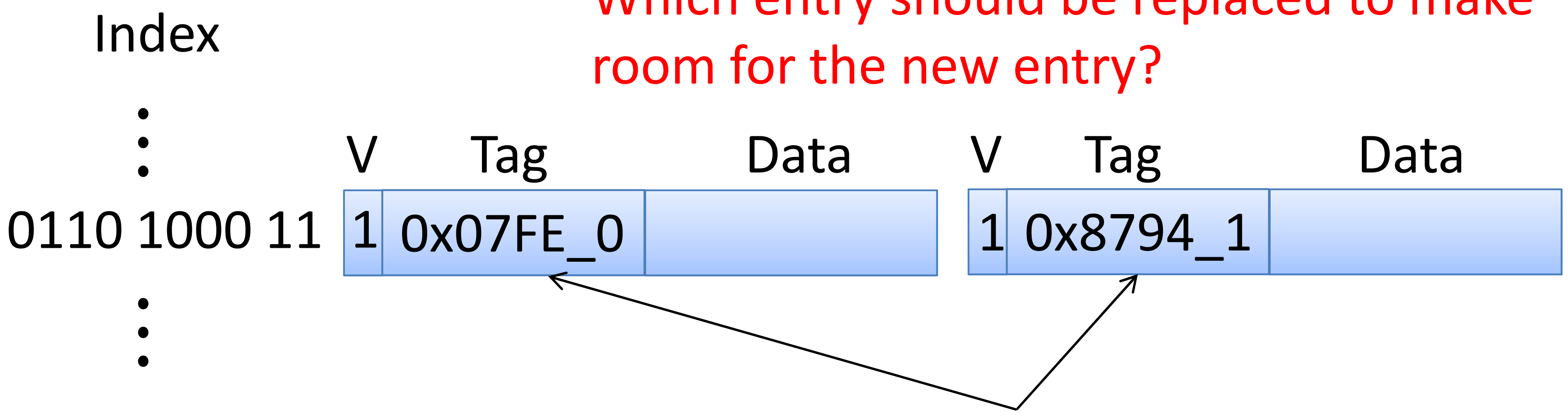
It is a hit in this entry of the set

# 2-way set associative caches

New Memory reference: 0x6541 B468

$$\underbrace{0110\ 0101\ 0100\ 0001\ 1}_{\text{Tag = 0x6541\_1}}\underbrace{011\ 0100\ 011}_{\text{Index}}\underbrace{0\ 1000}_{\text{Offset}}$$

Which entry should be replaced to make room for the new entry?

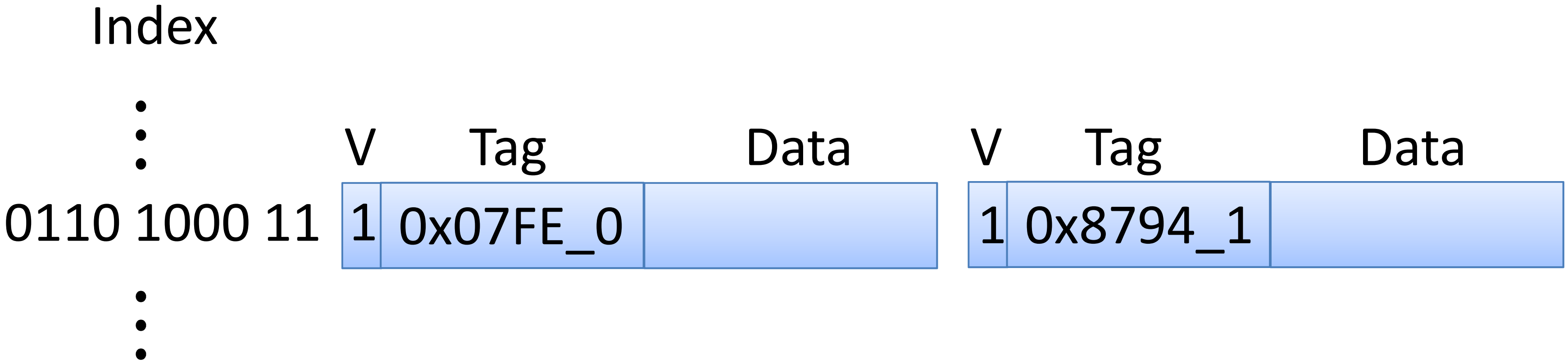


Misses both entries in the set.

# 2-way set associative caches

**LRU**: **L**east-**R**ecently **U**sed entry.

**MRU**: **M**ost-**R**ecently **U**sed entry.

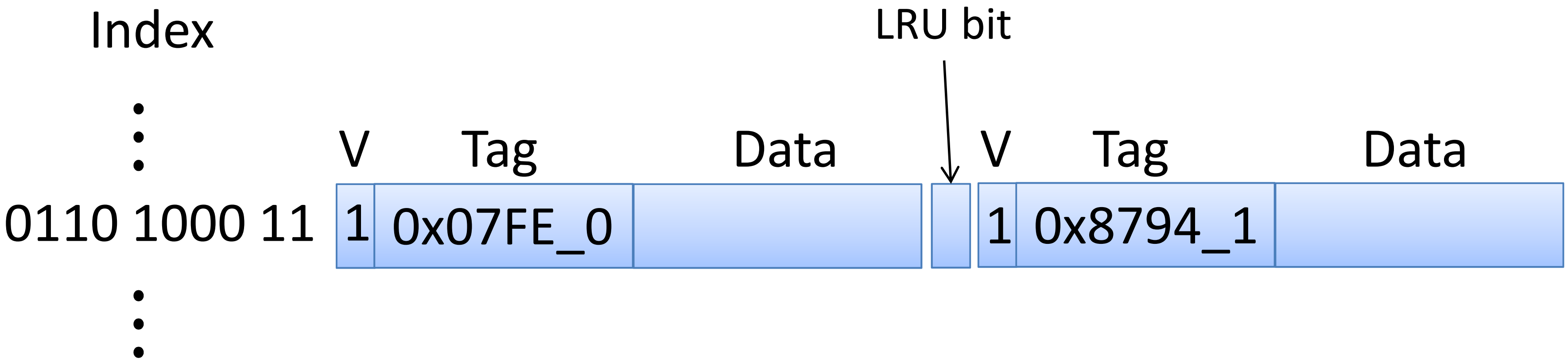


| Index | V | Tag | Data | V | Tag | Data |
|---|---|---|---|---|---|---|
| ⋮ | | | | | | |
| 0110 1000 11 | 1 | 0x07FE_0 | | 1 | 0x8794_1 | |
| ⋮ | | | | | | |

# 2-way set associative caches

**LRU**: **L**east-**R**ecently **U**sed entry.

**MRU**: **M**ost-**R**ecently **U**sed entry.

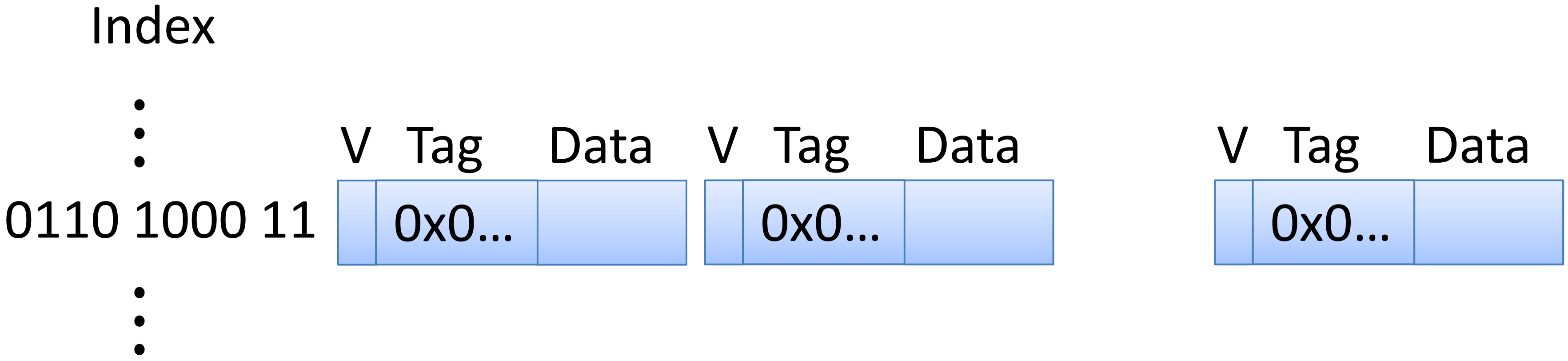


| Index | V | Tag | Data | LRU bit | V | Tag | Data |
|---|---|---|---|---|---|---|---|
| ⋮ | | | | | | | |
| 0110 1000 11 | 1 | 0x07FE_0 | | | 1 | 0x8794_1 | |
| ⋮ | | | | | | | |

# *n*-way set associative caches

An **LRU** bit is not a solution.

- Computing true **LRU** entry is too expensive.
    - Solution: use an **LRU** approximation.

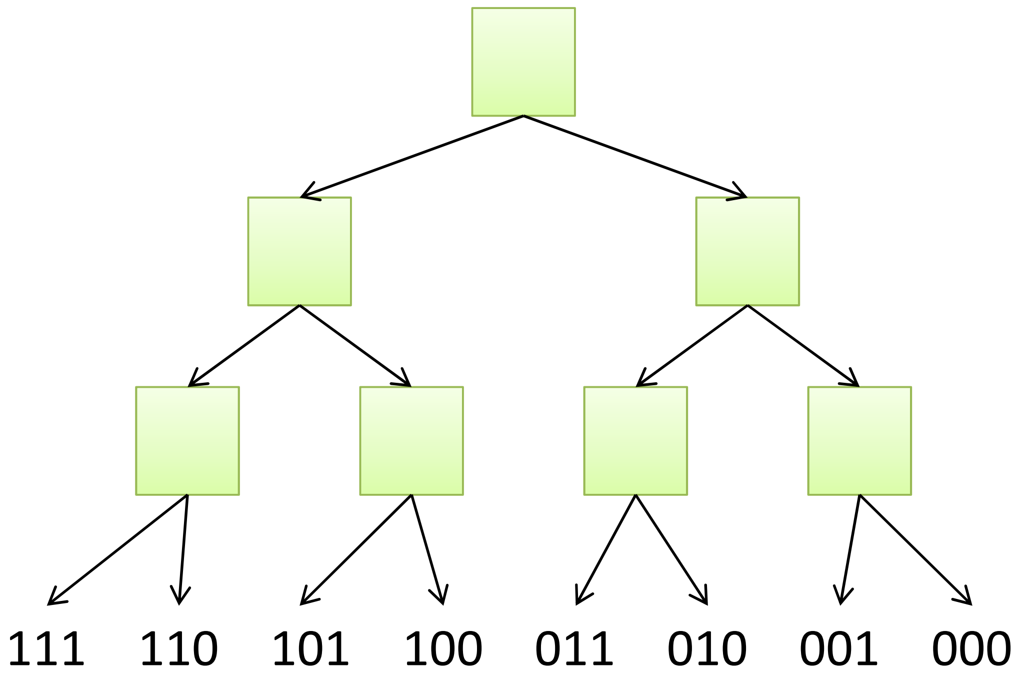


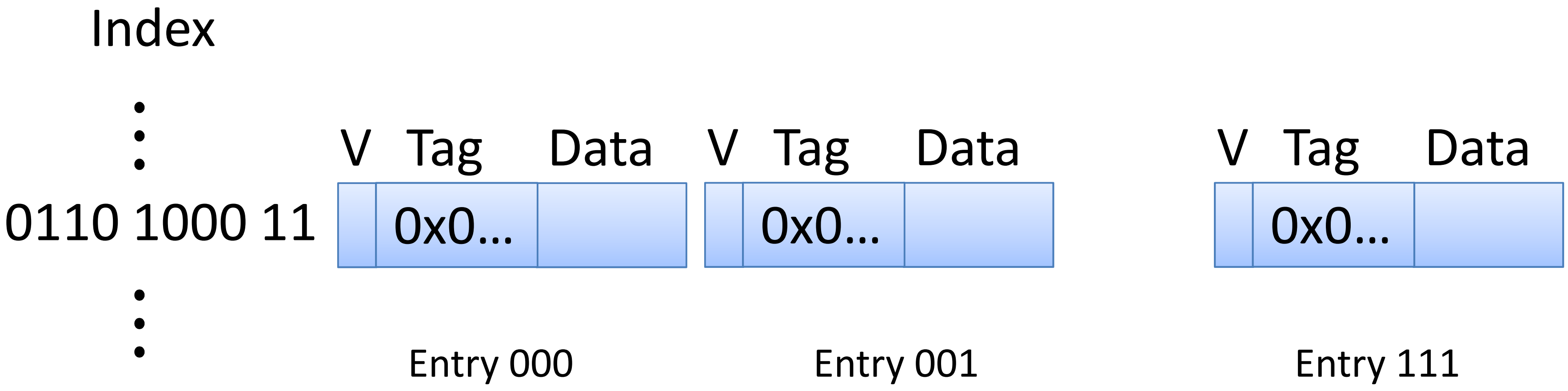
Index

0110 1000 11

| V | Tag | Data | V | Tag | Data | V | Tag | Data |
|---|---|---|---|---|---|---|---|---|
| | 0x0… | | | 0x0… | | | 0x0… | |

# LRU Approximation



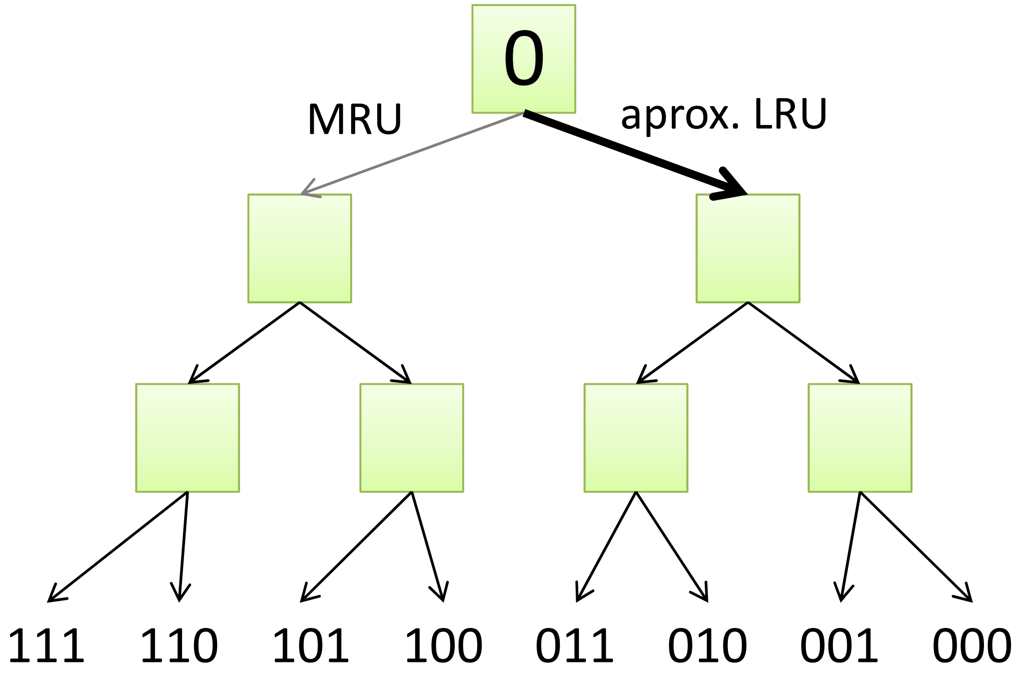


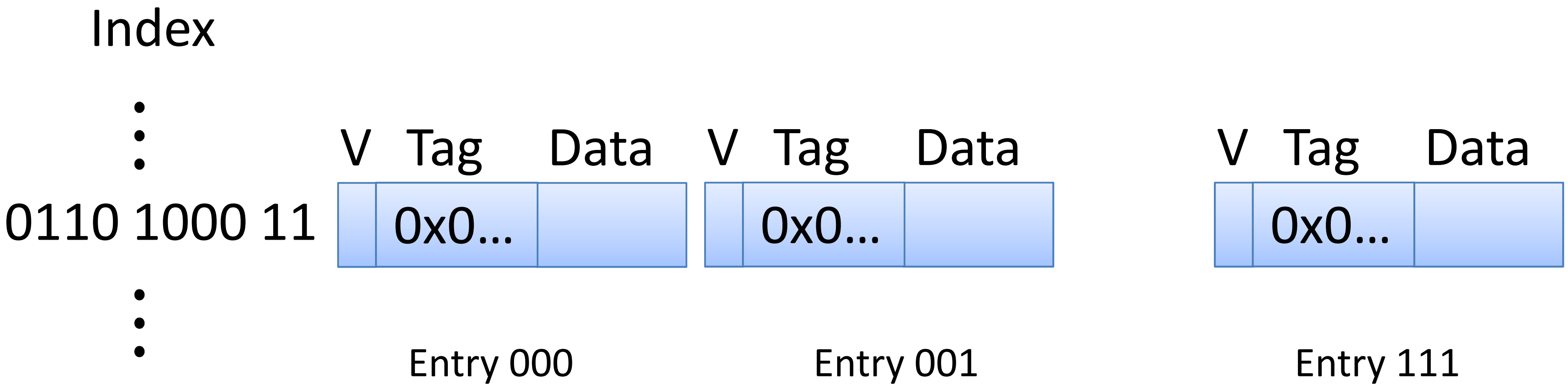
Entry 000

Entry 001

Entry 111

# LRU Approximation




Index

⋮

0110 1000 11

⋮

V Tag Data

0x0…

Entry 000

V Tag Data

0x0…

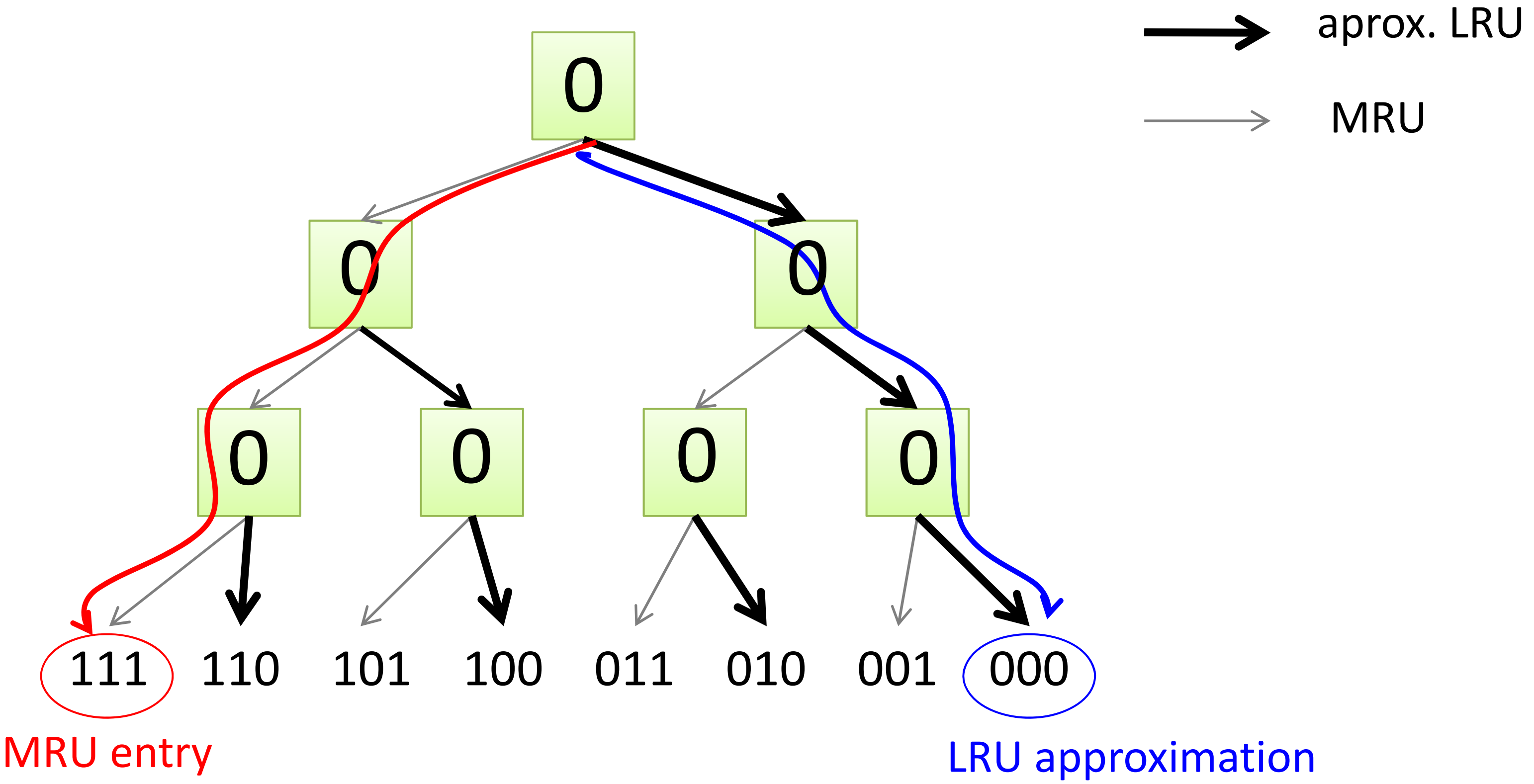
Entry 001

V Tag Data

0x0…

Entry 111

# LRU Approximation

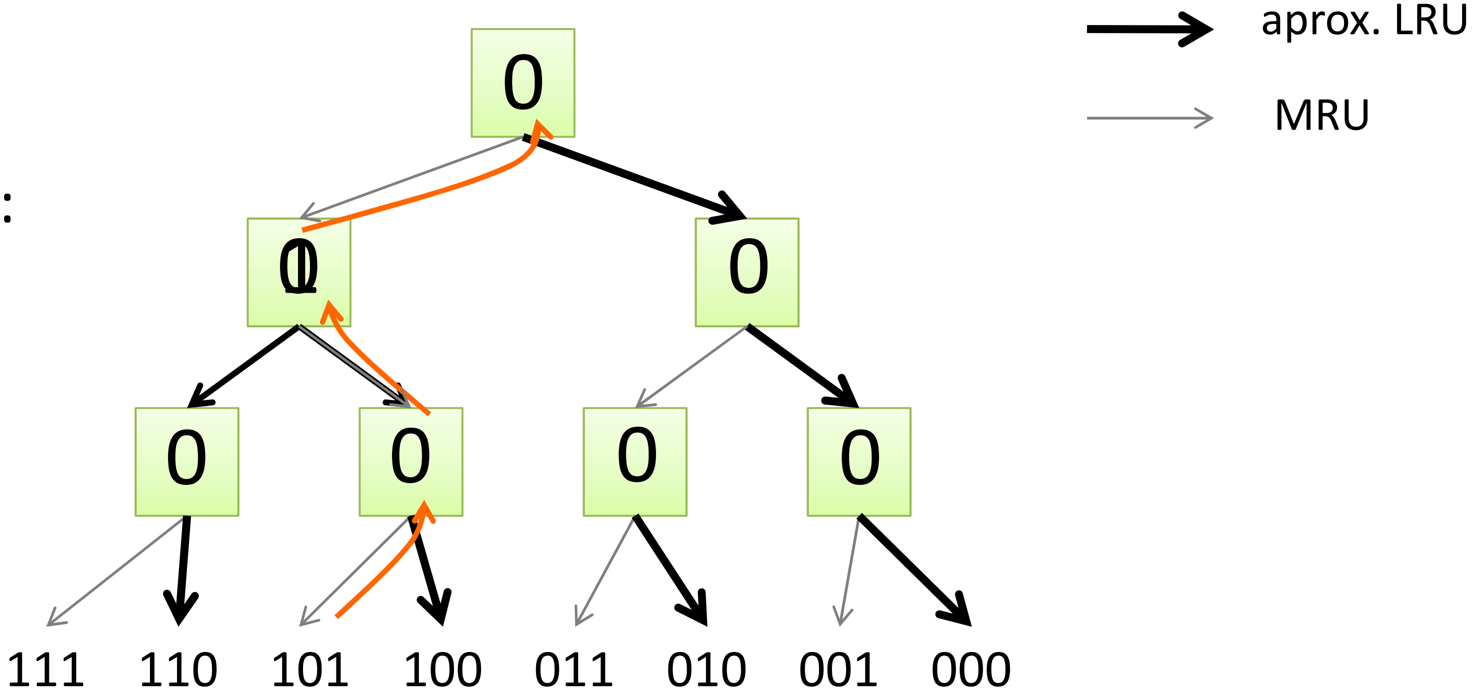


| Index | V | Tag | Data | V | Tag | Data | V | Tag | Data |
|---|---|---|---|---|---|---|---|---|---|
| ⋮ | | | | | | | | | |
| 0110 1000 11 | | 0x0… | | | 0x0… | | | 0x0… | |
| ⋮ | | | | | | | | | |
| | Entry 000 | | | Entry 001 | | | Entry 111 | | |

# LRU Approximation

References:

101



Index

⋮

0110 1000 11

⋮

| V | Tag | Data | V | Tag | Data | V | Tag | Data |
|---|---|---|---|---|---|---|---|---|
|  | 0x0… |  |  | 0x0… |  |  | 0x0… |  |

Entry 000

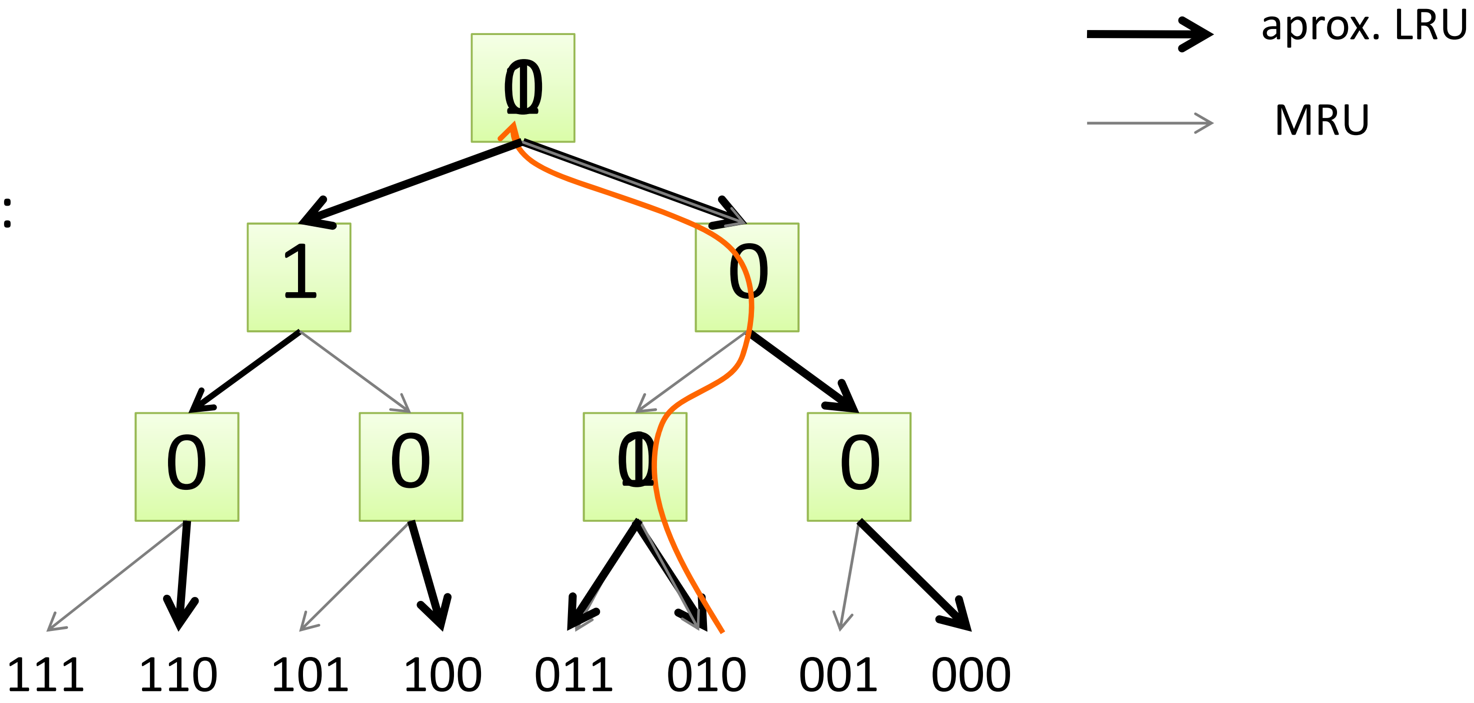
Entry 001

Entry 111

# LRU Approximation

References:

101
010

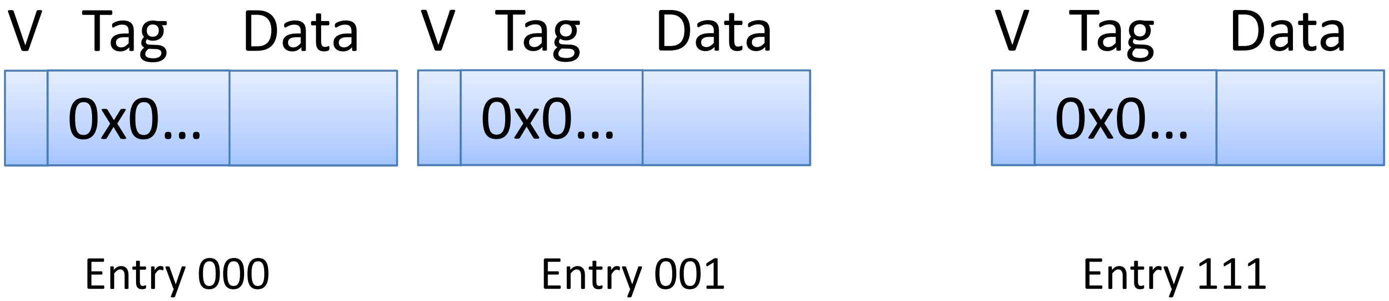


Index

⋮

0110 1000 11

⋮


V Tag Data

0x0…

Entry 000

V Tag Data

0x0…

Entry 001

V Tag Data

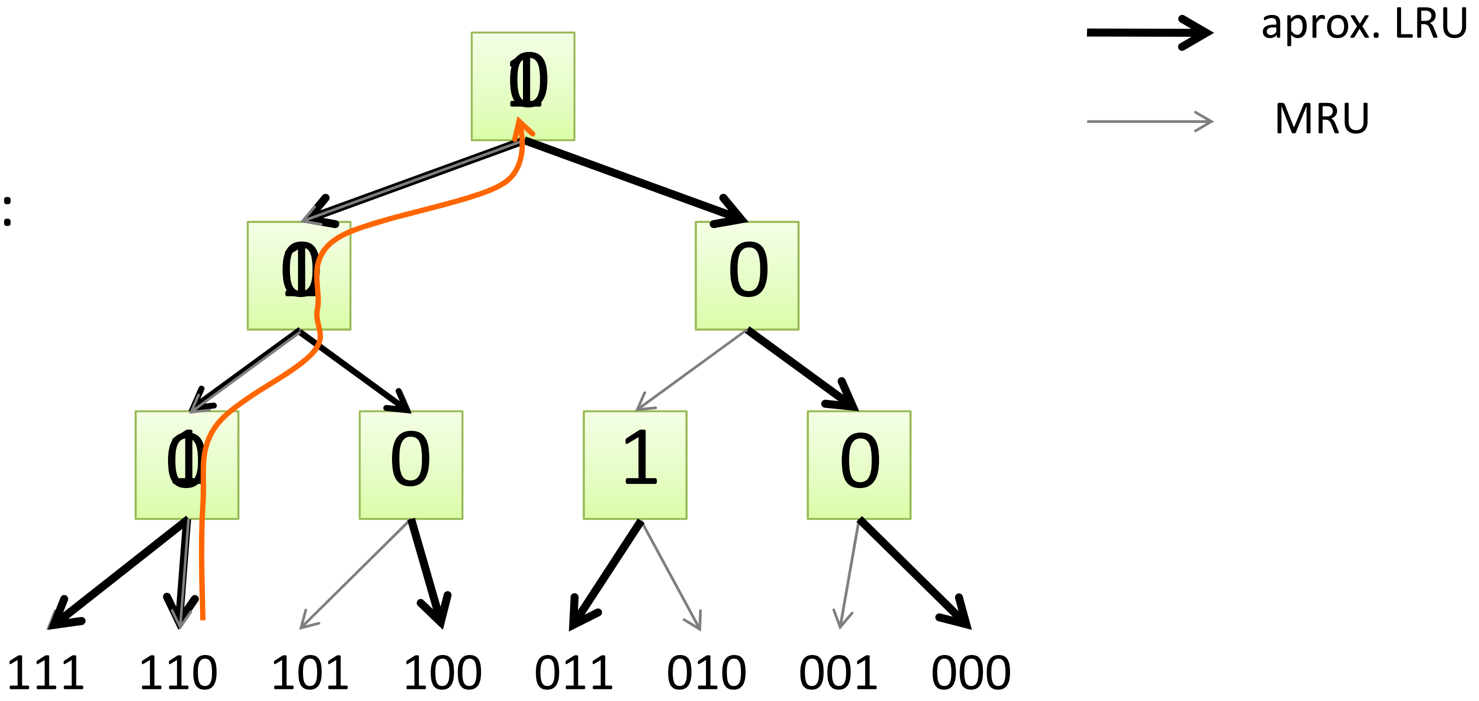
0x0…

Entry 111

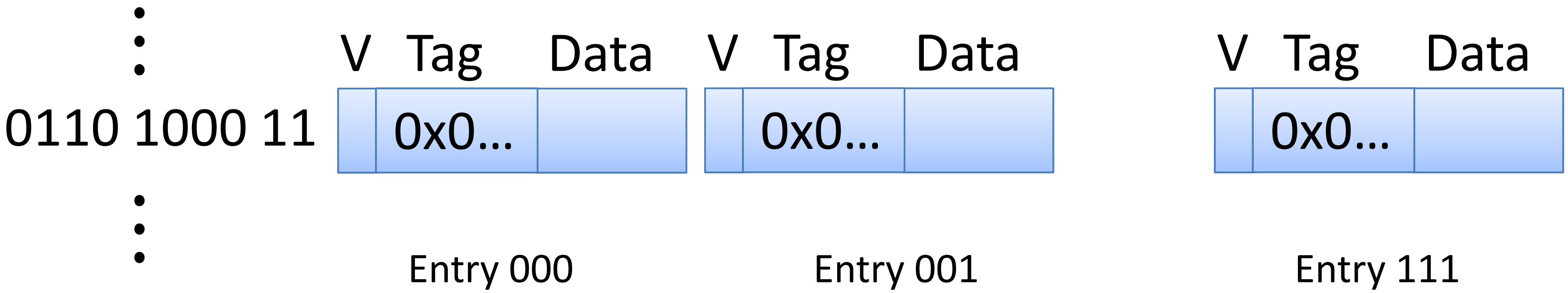
# LRU Approximation

References:

101
010
110



Index


⋮

0110 1000 11

⋮

| V | Tag | Data | V | Tag | Data | V | Tag | Data |
|---|---|---|---|---|---|---|---|---|
| | 0x0… | | | 0x0… | | | 0x0… | |

Entry 000

Entry 001

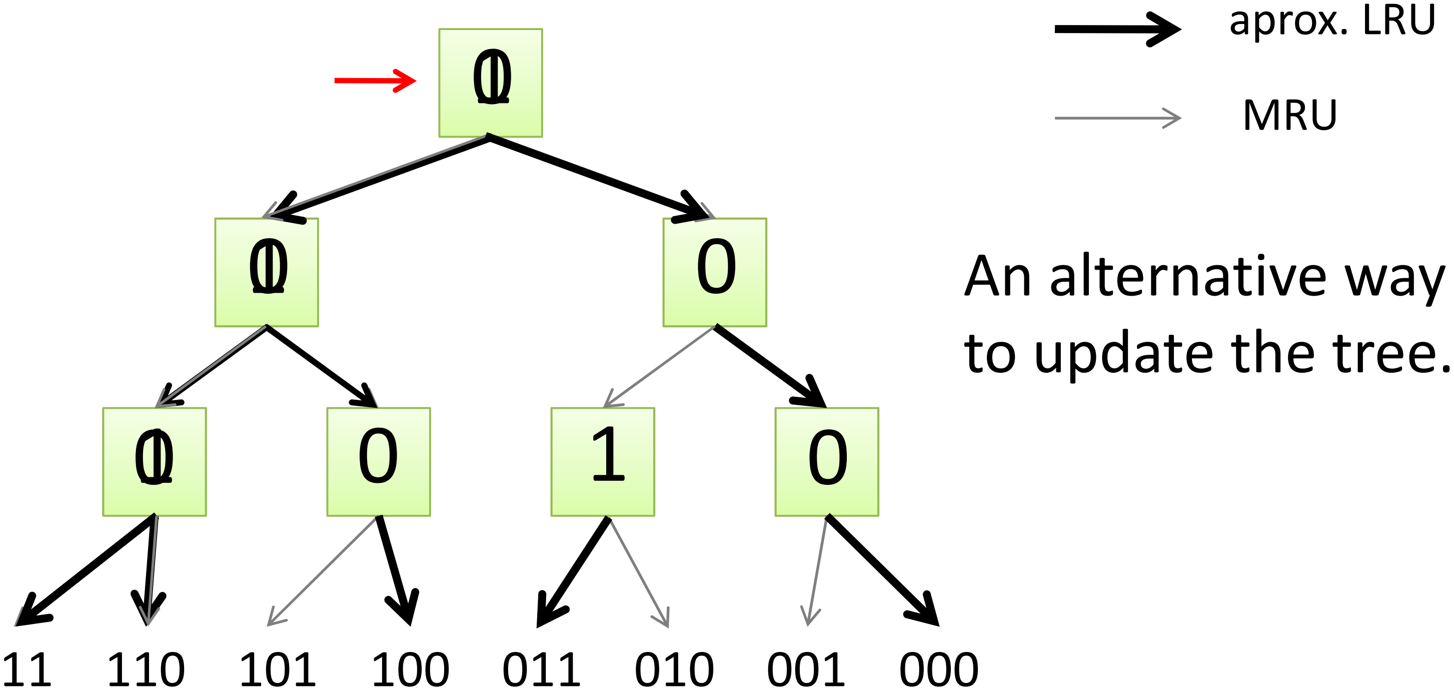
Entry 111

# LRU Approximation

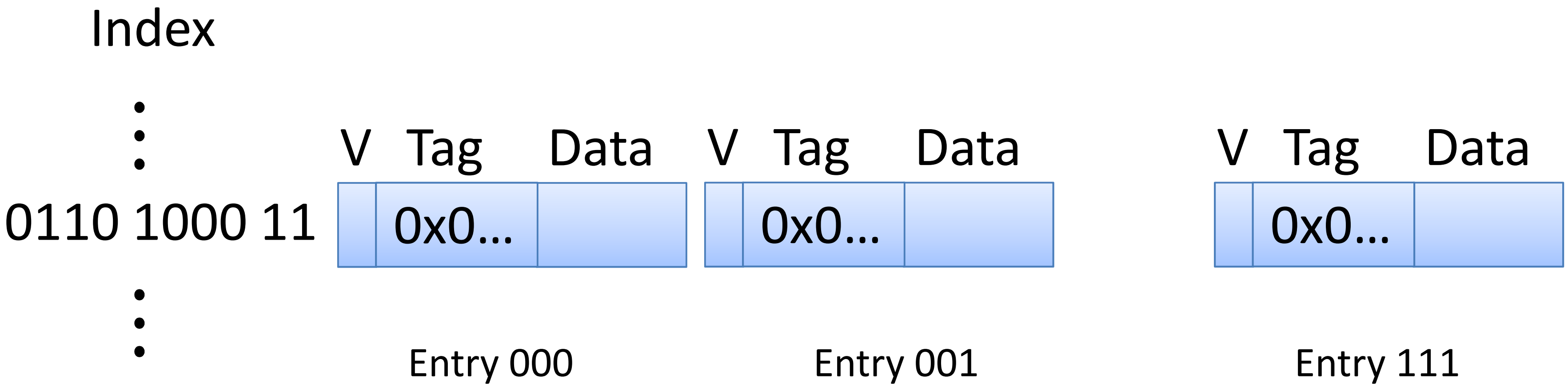
References:

101
010
110

001



An alternative way to update the tree.



# Assignment

- startCache
  - Arguments:
    - $a0 = the associativity of the cache
  - Return Values: None
  - Execution:
    - Initialize cache set with all nodes equal 0.

# Assignment (cont.)

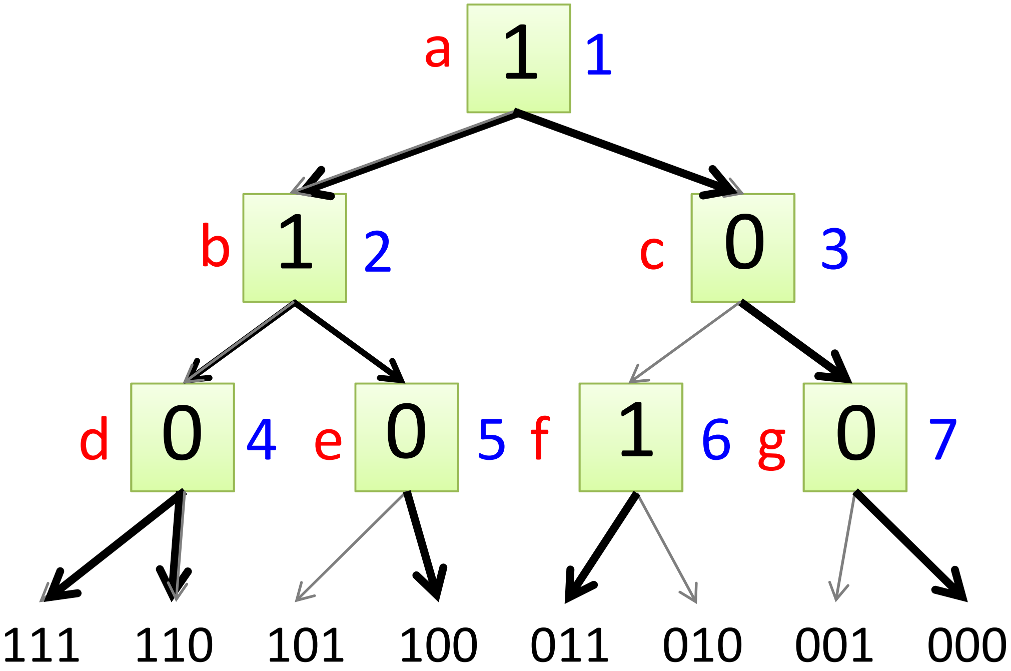
- getLRU
  - Arguments:
    - $a0 = Pointer to a stream of bits
    - $a1 = the number of references to be read from stream
  - Return Values:
    - $v0 = identifies the approximate LRU entry. For example is the entry 10011 is the LRU approximation, then return:
      - $V0 = 0000 0000 0000 0000 0000 0000 0001 0011

# Example

- $a0 = 0x10010044, $a1 = 7

| Address | Value |
|---|---|
| 0x 1001 0044 | 0101 0101 1011 0010 0111 0110 1110 0000 |
| 0x 1001 0048 | 0000 0000 0000 0000 0000 0000 1111 1101 |

# Storing a binary tree in memory



left child = 2 × parent

right child = 2 × parent + 1

parent = $\lfloor \text{child}/2 \rfloor$

| Address | Value<br>hgfe dcba |
|---|---|
| 0x 1000 1000 | 0000 0000 0000 0000 0000 0000 0010 0011 |
| 0x 1000 1004 | 0000 0000 0000 0000 0000 0000 0000 0000 |