HASH INDEXES

CS 564 - Spring 2025

WHAT IS THIS LECTURE ABOUT?

Hash indexes

- Static Hashing
  - I/O cost
  - issues with static hashing
- Extendible Hashing
  - insertion
  - deletion

HASH INDEXES

- efficient for equality search

- not appropriate for range search

- Types of hash indexes:
  - static hashing
  - extendible (dynamic) hashing

STATIC HASHING

- A **<u>hash index</u>** is a collection of *buckets*
  - bucket = primary page + overflow pages
  - each bucket contains one or more index entries
- To find the bucket for each record, apply a hash function h
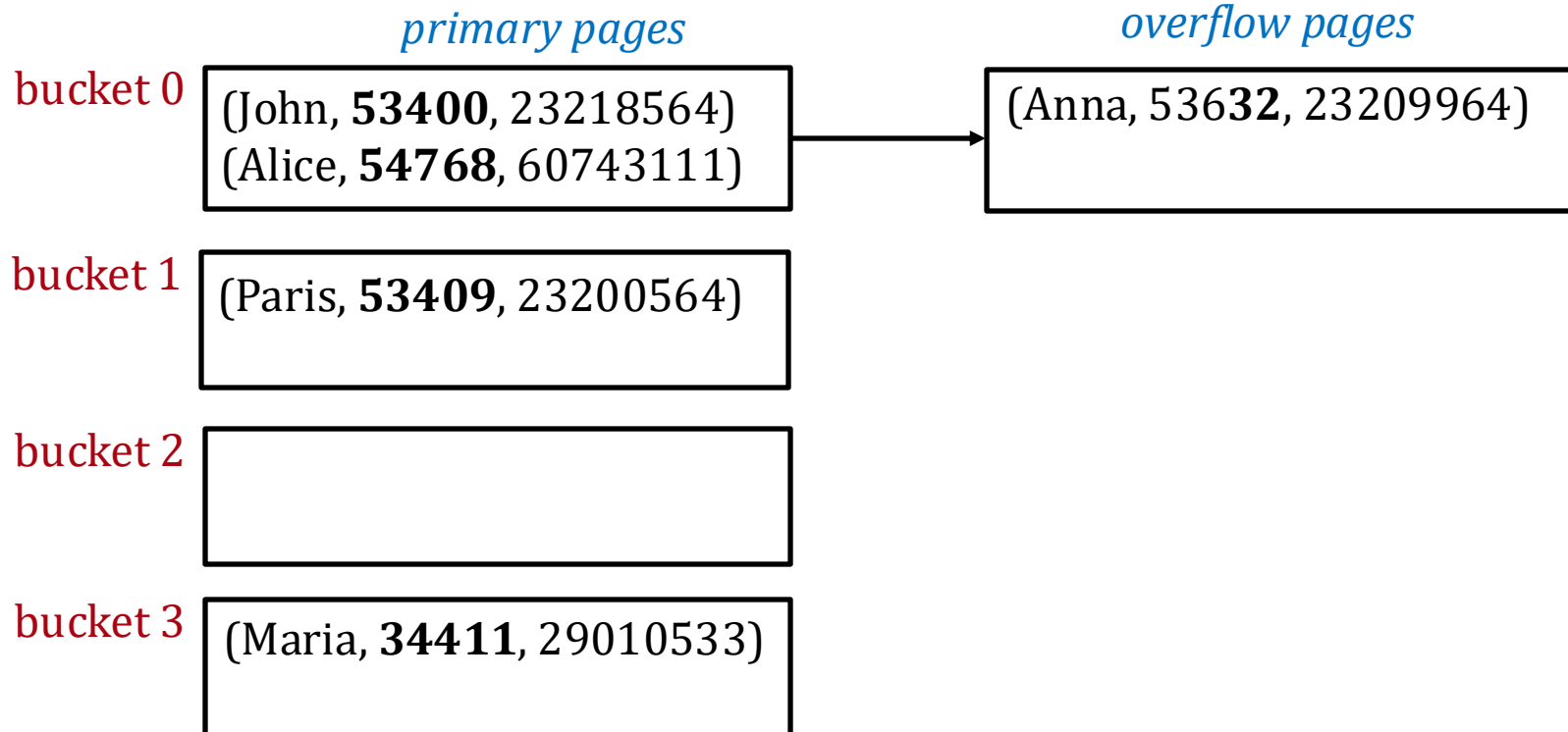- h maps a search key value to one of the buckets

STATIC HASHING: EXAMPLE

Person(name,zipcode,phone)

- *search key*: {zipcode}
- *hash function **h***: zipcode mod 4

- 4 buckets
- each bucket holds 2 index entries

primary pages

overflow pages

bucket 0
| (John, **53400**, 23218564) |
| (Alice, **54768**, 60743111) |

→ (Anna, 536**32**, 23209964)

bucket 1
| (Paris, **53409**, 23200564) |

bucket 2
| |

bucket 3
| (Maria, **34411**, 29010533) |

OPERATIONS ON HASH INDEXES

Equality search (*search-key = value*)

- apply the hash function on the search key to locate the appropriate bucket

- search through the primary page + overflow pages to find the record(s)

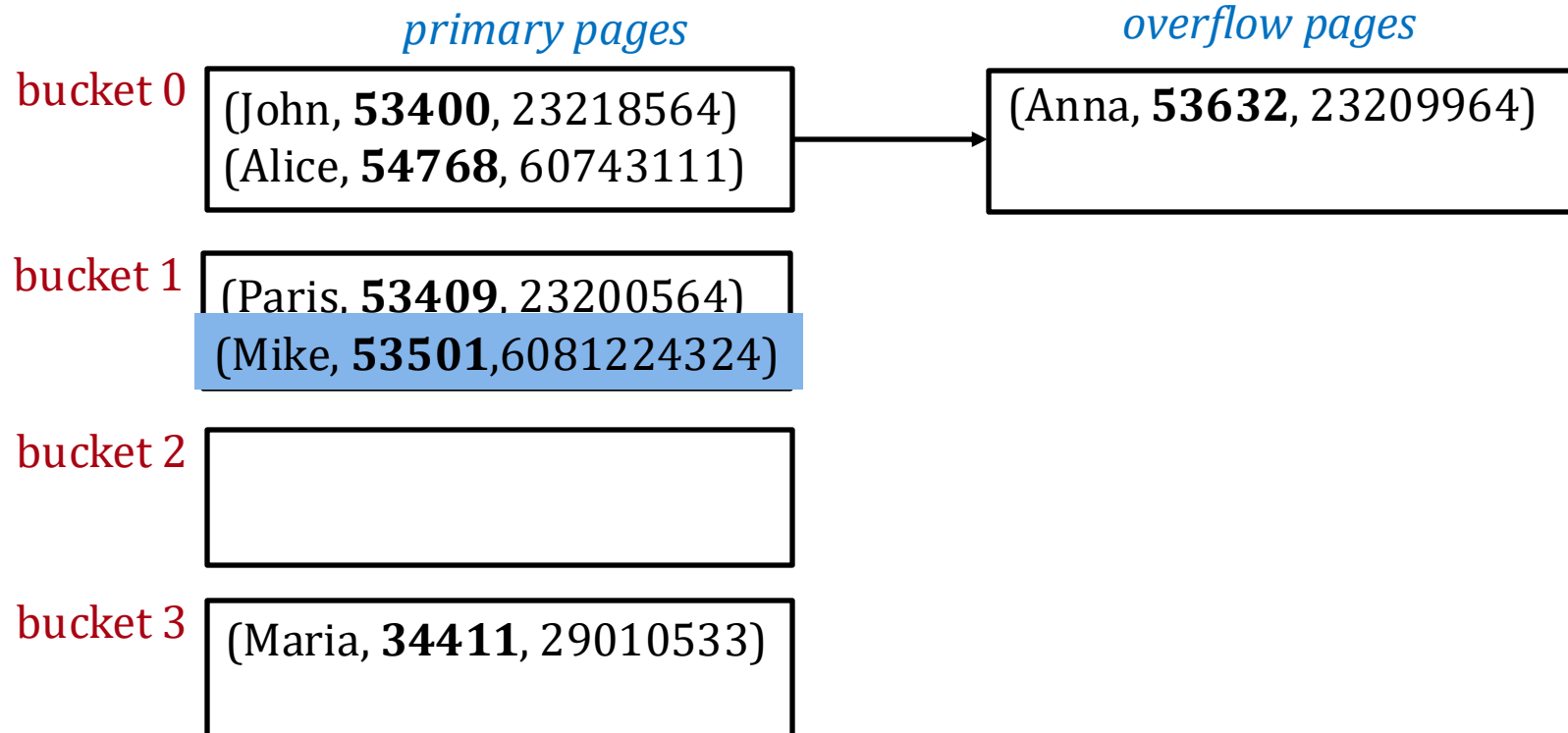$$\text{I/O cost} = 1 + \text{\#overflow pages}$$

The I/O cost increases if we choose alternative #2

OPERATIONS ON HASH INDEXES

- **Deletion**
  - find the appropriate bucket, delete the record

- **Insertion**
  - find the appropriate bucket, insert the record
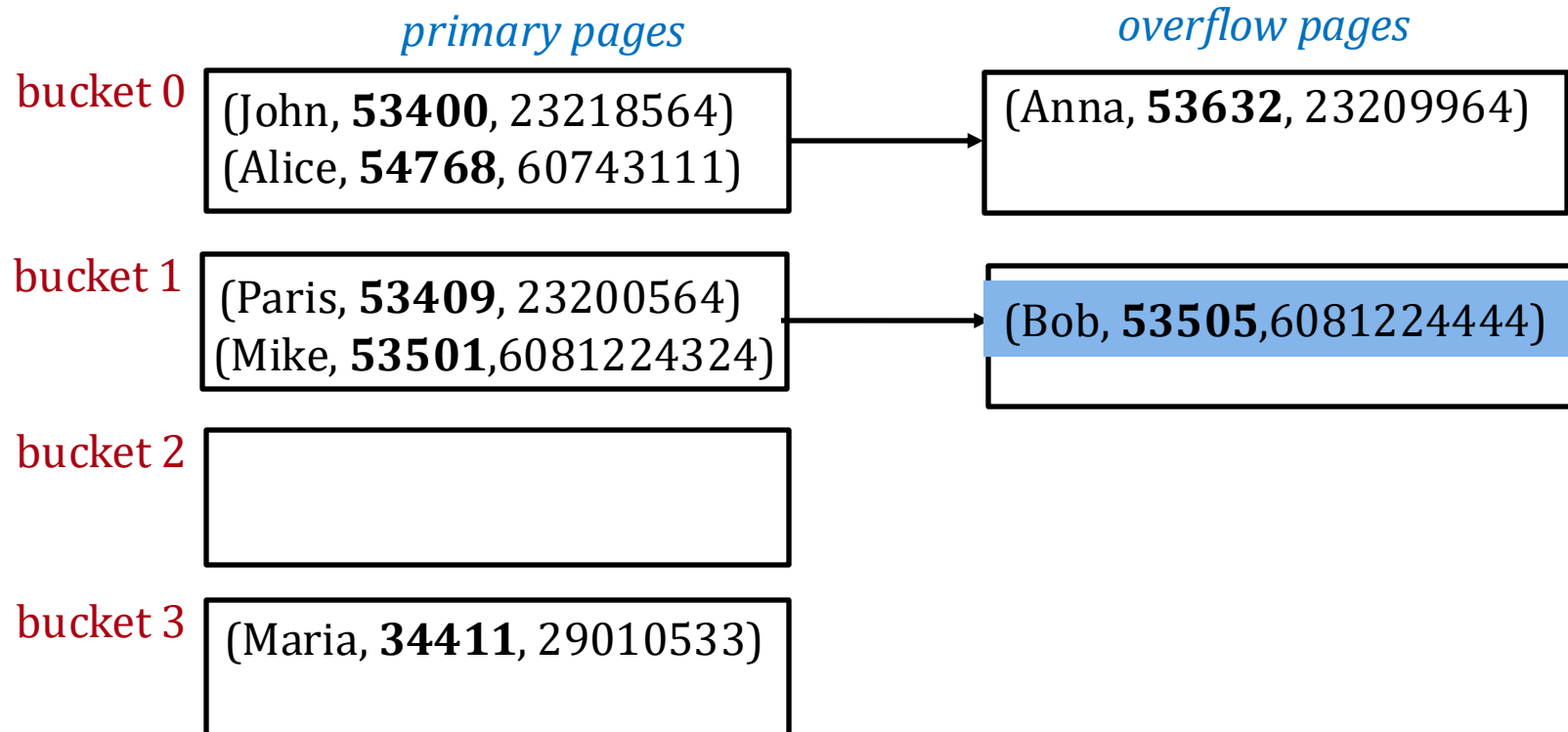 - if there is no space, create a new overflow page

STATIC HASHING: INSERT

insert: (Mike, **53501**,6081224324)

primary pages

overflow pages

bucket 0

(John, **53400**, 23218564)
(Alice, **54768**, 60743111)

(Anna, **53632**, 23209964)

bucket 1

(Paris, **53409**, 23200564)
(Mike, **53501**,6081224324)

bucket 2

bucket 3

(Maria, **34411**, 29010533)

STATIC HASHING: INSERT

insert: (Bob, **53505**,6081224444)

primary pages   *overflow pages*

bucket 0
| (John, **53400**, 23218564) |
| (Alice, **54768**, 60743111) |

→ | (Anna, **53632**, 23209964) |

bucket 1
| (Paris, **53409**, 23200564) |
| (Mike, **53501**,6081224324) |

→ | (Bob, **53505**,6081224444) |

bucket 2
| |

bucket 3
| (Maria, **34411**, 29010533) |

HASH FUNCTIONS

- An *ideal* hash function is uniform: each bucket is assigned the same number of key values

- A *bad* hash function maps all search key values to the same bucket

- Hash functions in practice:
  - should be fast to compute
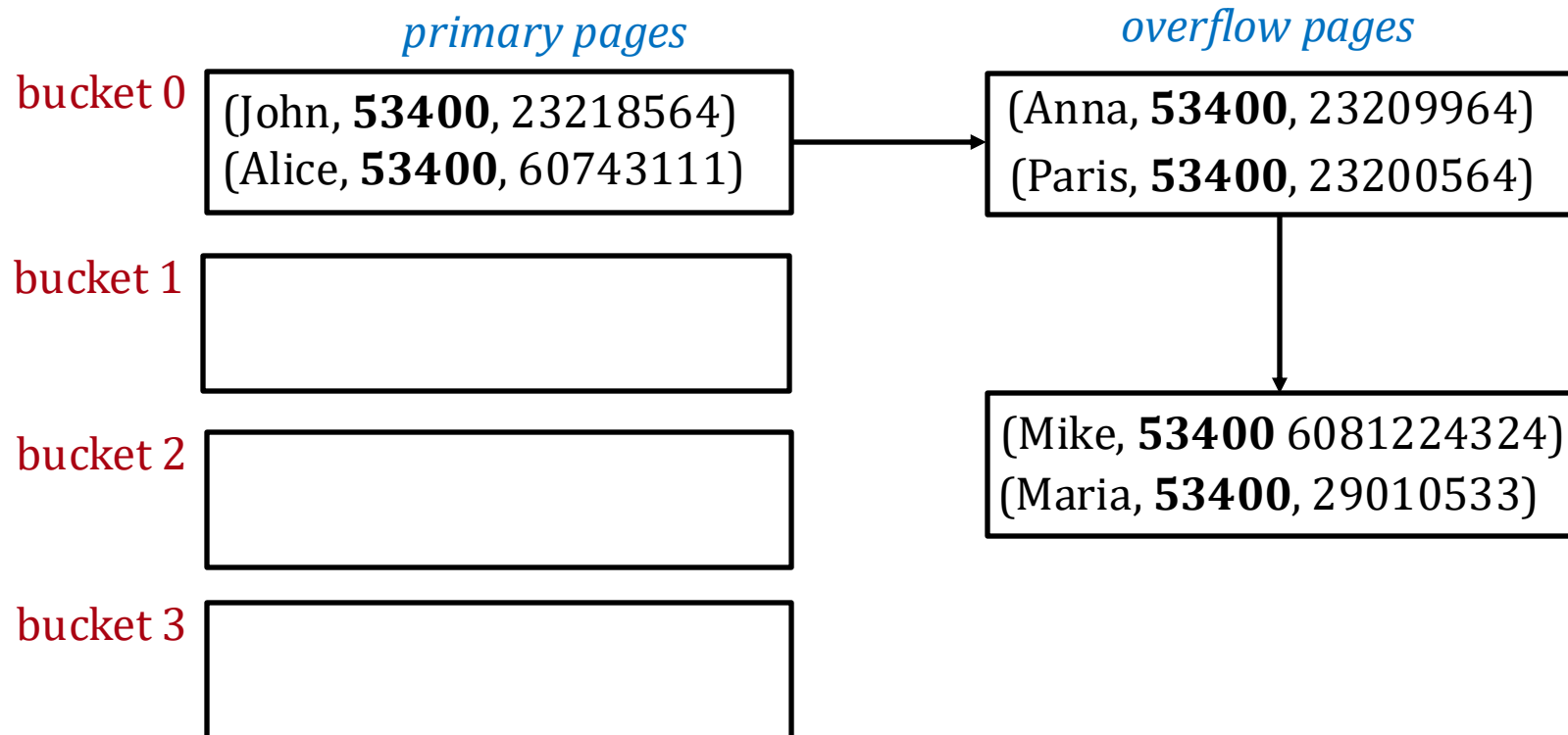 - should have a low collision rate

PROBLEMS OF STATIC HASHING

- In static hashing, there is a **fixed** number of buckets in the index

- Issues with this:
  - if the database grows, the number of buckets will be too small: long overflow chains degrade performance
  - if the database shrinks, space is wasted because of empty buckets
  - reorganizing the index is expensive and can block query execution

PROBLEMS OF STATIC HASHING

- Even with a good hash function, long overflow chains can still occur when we have **skew**

- Skew occurs when many records have the same search key value, so they will always end up in the same bucket

STATIC HASHING - SKEW

primary pages   *overflow pages*

bucket 0 | (John, **53400**, 23218564)
 (Alice, **53400**, 60743111) | → | (Anna, **53400**, 23209964)
 (Paris, **53400**, 23200564) |

bucket 1

bucket 2

(Mike, **53400** 6081224324)
 (Maria, **53400**, 29010533)

bucket 3

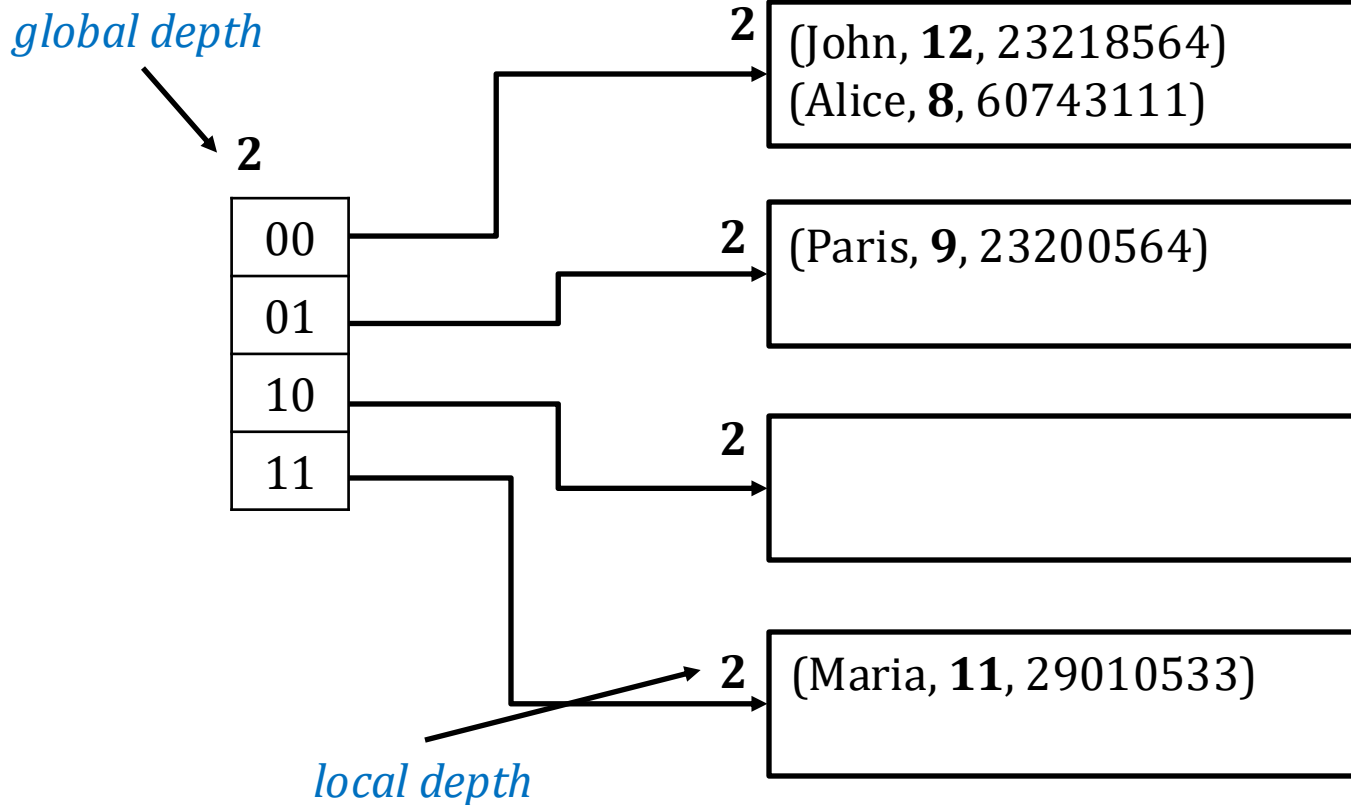# EXTENDIBLE HASHING

EXTENDIBLE HASHING

Extendible hashing is a type of *dynamic* hashing

- It keeps a directory of pointers to buckets

- On overflow, it reorganizes the index by doubling the directory (and not the number of buckets)
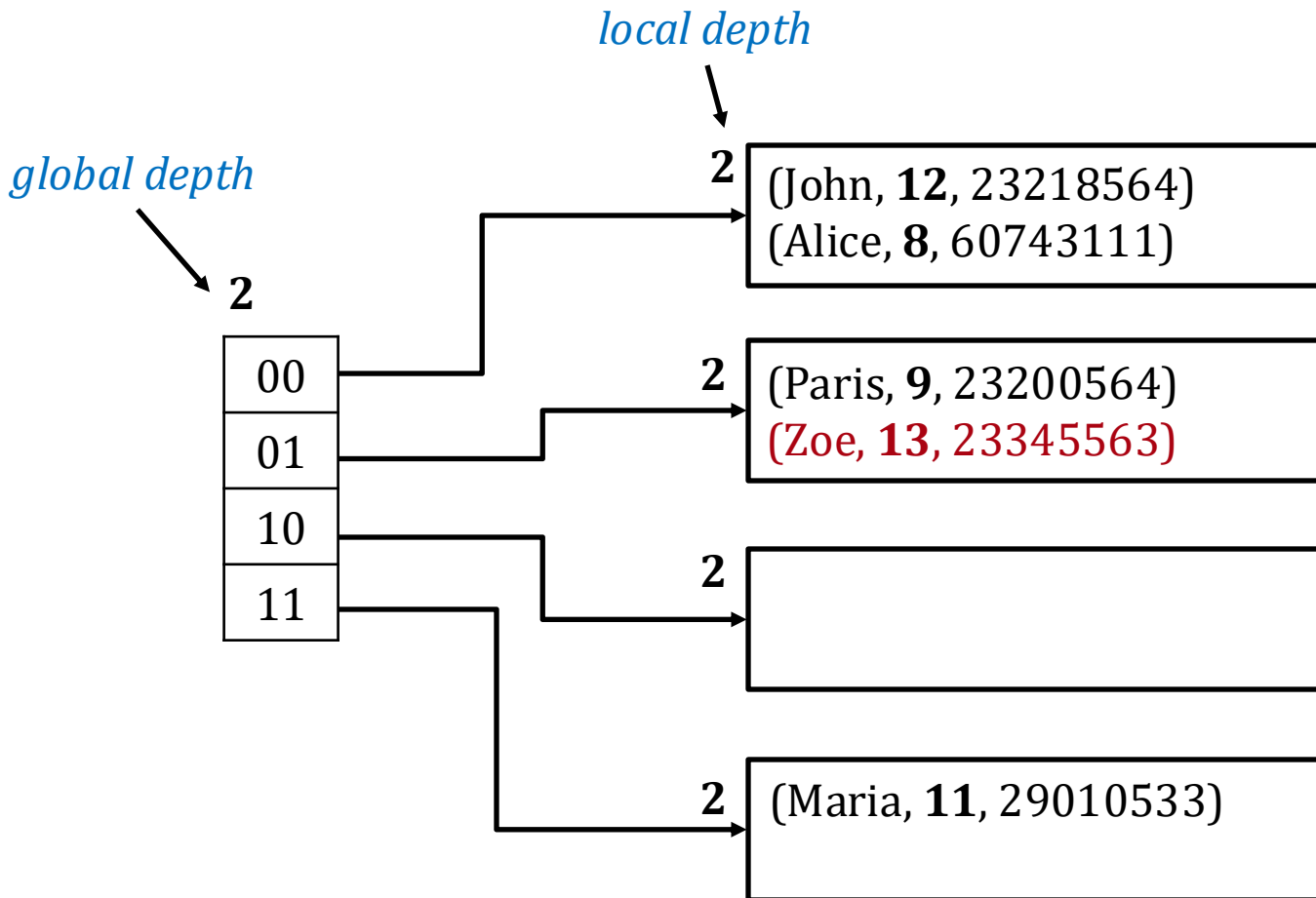
EXTENDIBLE HASHING

To search, use the last *(global depth)* digits of the **binary** form of the search key value

12 = 1100

global depth

2

| 2 | (John, **12**, 23218564) (Alice, **8**, 60743111) |

| 00 | 2 | (Paris, **9**, 23200564) |
| 01 |
| 10 | 2 | |
| 11 |

| 2 | (Maria, **11**, 29010533) |

local depth

EXTENDIBLE HASHING: INSERT

If there is space in the bucket, simply add the record

local depth

global depth

2

00
01
10
11

2 | (John, **12**, 23218564)
(Alice, **8**, 60743111)

2 | (Paris, **9**, 23200564)
(Zoe, **13**, 23345563)

2 |

2 | (Maria, **11**, 29010533)

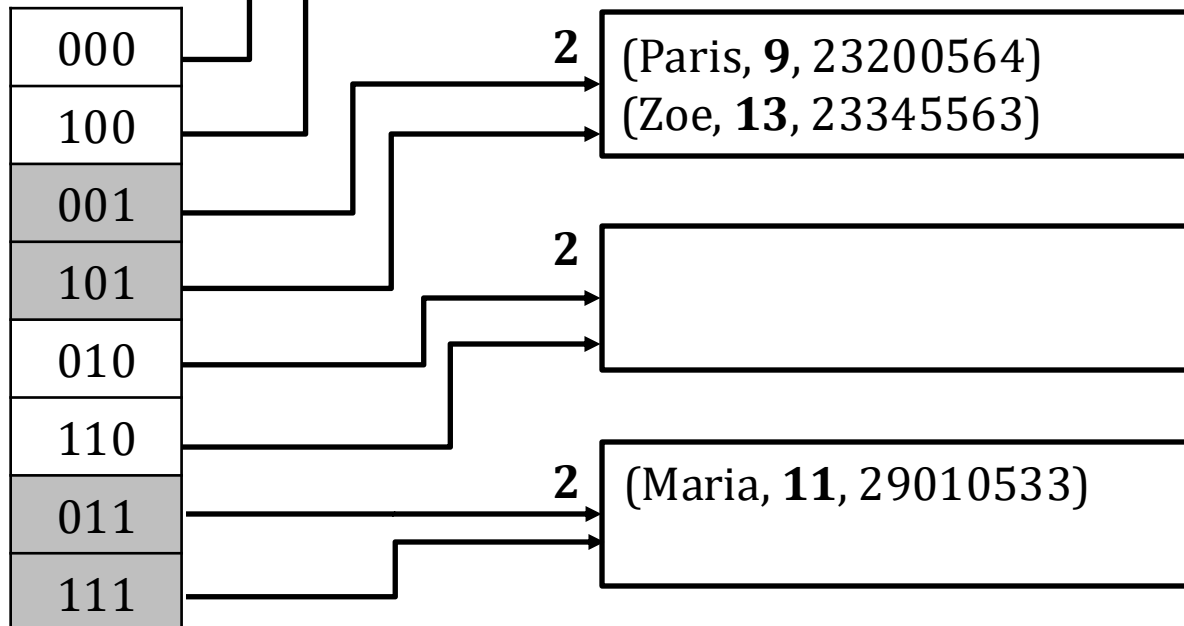# EXTENDIBLE HASHING: INSERT

If the bucket is full, split the bucket and redistribute the entries
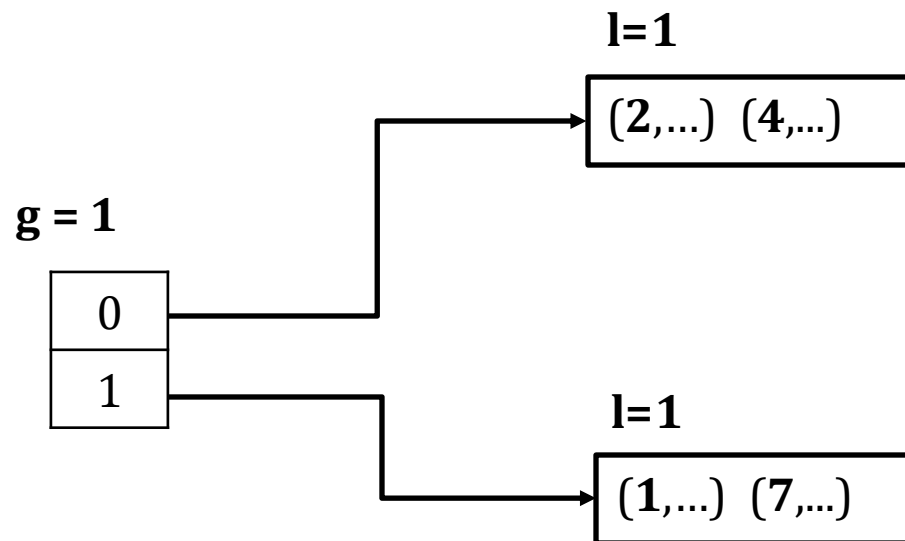
global depth increases by 1

3

3 (Alice, **8**, 60743111)

local depth increases for the split bucket!

3 (Natalie, **4**, 23200564)
(John, **12**, 23218564)

| 000 |
| 100 |
| 001 |
| 101 |
| 010 |
| 110 |
| 011 |
| 111 |

2 (Paris, **9**, 23200564)
(Zoe, **13**, 23345563)

local depth remains the same for the other buckets

2

2 (Maria, **11**, 29010533)

EXAMPLE

each page can hold at most two records

l=1

(**2**,…) (**4**,…)

g = 1

0
1

l=1

(**1**,…) (**7**,…)

INVARIANT: global depth >= local depth

EXAMPLE

insert: (**22**,…)

- The catalog doubles in size
- Global depth becomes 2

g = 2

00
01
10
11

l=2
(**4**,…)

l=2
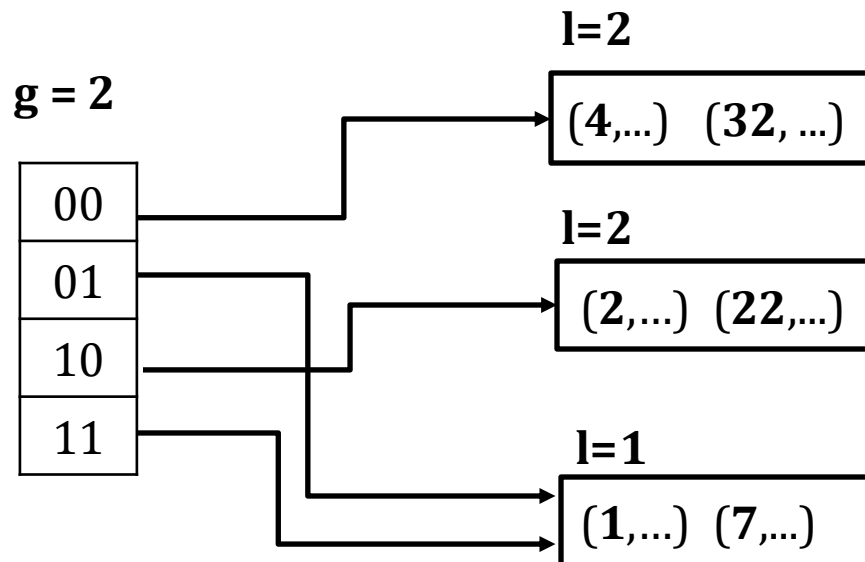(**2**,…) (**22**,…)

l=1
(**1**,…) (**7**,…)
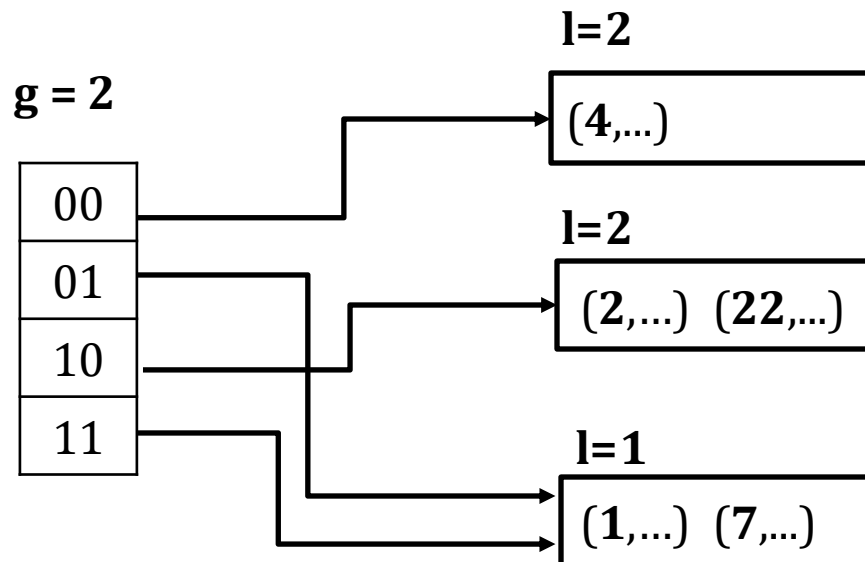
This bucket is split into two buckets with local depth 2

This bucket remains the same

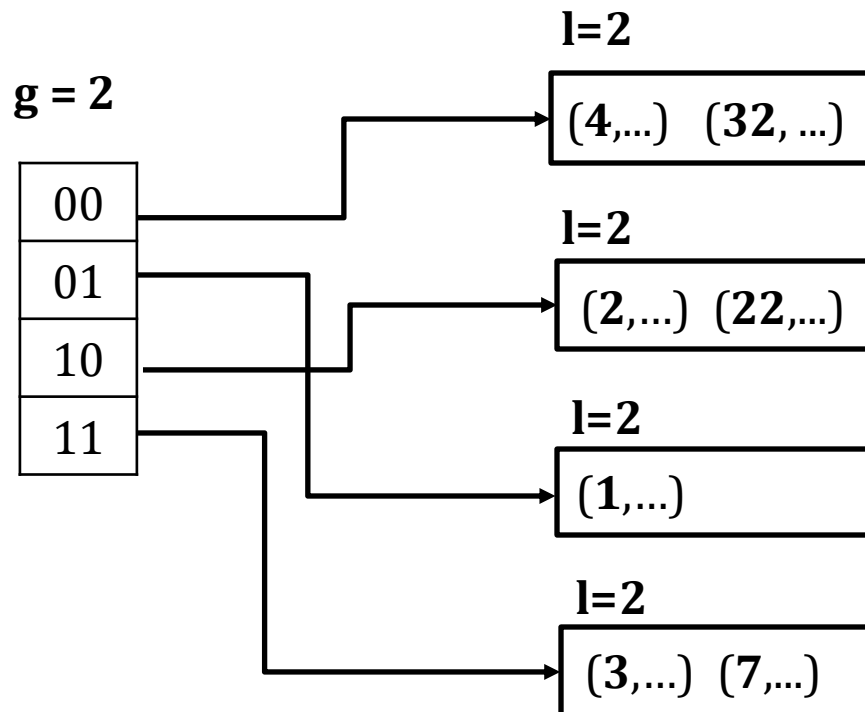# EXAMPLE

There is space in the bucket so nothing changes!

insert: (**32**,…)

l=2

g = 2

(**4**,…)   (**32**, …)

00

l=2

01

(**2**,…) (**22**,…)

10

11

l=1

(**1**,…) (**7**,…)

EXAMPLE

Since local depth is smaller than global, no need to change the directory size!

insert: (**3**,...)

g = 2

00
01
10
11

l=2

(**4**,...)   (**32**, ...)

l=2

(**2**,...) (**22**,...)

l=2

(**1**,...)

l=2

(**3**,...) (**7**,...)

The bucket is split into two buckets with local depth 2

EXTENDIBLE HASHING: DELETE

- Locate the bucket of the record and remove it

- If the bucket becomes empty, it can be removed (and update the directory)

- Two buckets can also be coalesced together if the sum of the entries fit in a single bucket

- Decreasing the size of the directory can also be done, but it is expensive

MORE ON EXTENDIBLE HASHING

- How many I/Os for equality search?
  - One if directory fits in memory, else two
- The directory grows in spurts, and, if the distribution of hash values is skewed, the directory can grow very large
- We may need overflow pages when multiple entries have the same hash value!