CS 564: Database Management Systems

Lecture 16: Hash Index

Xiangyao Yu

2/28/2024

Module B2 Indexes

Hash index

B+ tree index

Advanced Indexing

External sort

Outline

Index basics

Static hashing

- Equality search
- Insert and deletes
- Hash functions
- Limitations of static hashing

Extendible hashing

- Search
- Insertion and split
- Discussion

File Organization: Recap

So far we have seen heap files

- Unordered data
- Support scanning all records in a file
- Support retrieving by record id (rid)

What if we want to find a record satisfying a certain predicate?

- E.g., all employees with salary > 5000
- E.g., the department with name “Computer Sciences”

Motivating Example 1

Sales (sid, product, date, price)

```
SELECT *
FROM Sales
WHERE Sales.date = “03-15-2021”
```

equality predicate

What happens if the data is stored in a heap file?

– Scan all the pages of the file to return the correct result?

Motivating Example 2

Sales (sid, product, date, price)

```
SELECT *
FROM Sales
WHERE Sales.date > “03-01-2021”
AND Sales.date <= “03-15-2021”
```

range predicate

What happens if the data is stored in a heap file?

– Scan all the pages of the file to return the correct result?

What is an Index?

Index: a data structure that organizes records of a table to speed up retrieval

Search Key: attribute or combination of attributes used to retrieve the records

- A search key *may and may not* be the primary key!

```
CREATE INDEX index_name
ON table_name (column_name);
```

Outline

Index basics

Static hashing

- Equality search
- Insert and deletes
- Hash functions
- Limitations of static hashing

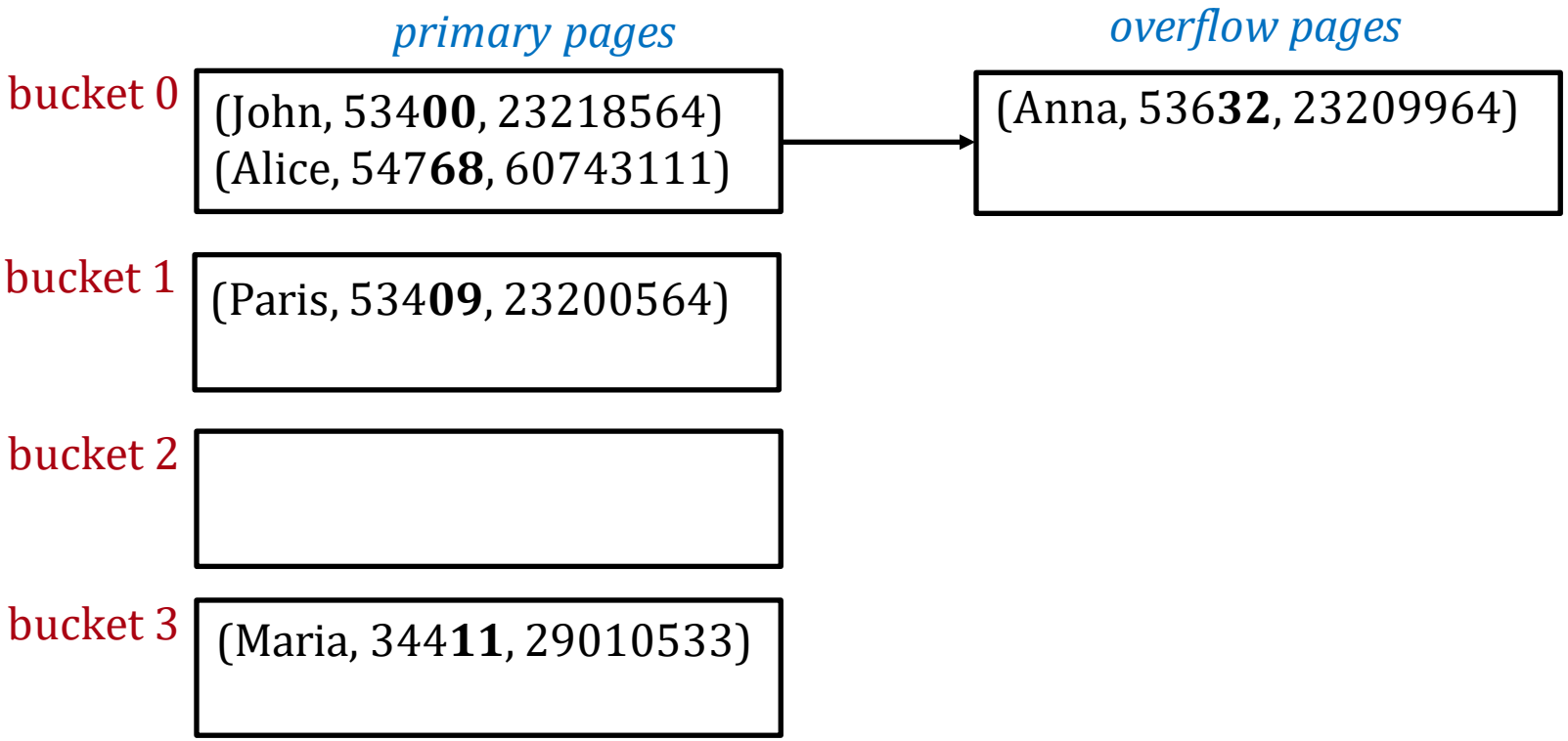
Extendible hashing

- Search
- Insertion and split
- Discussion

Static Hashing

A **hash index** is a collection of *buckets*

- Bucket = primary page + overflow pages
- Each bucket contains one or more index entries
- Bucket can store either *records* or *RID of records*



Static Hashing

A **hash index** is a collection of *buckets*

- Bucket = primary page + overflow pages
- Each bucket contains one or more index entries
- Bucket can store either *records* or *RID of records*

To find the bucket for each record, apply a hash function ***h***

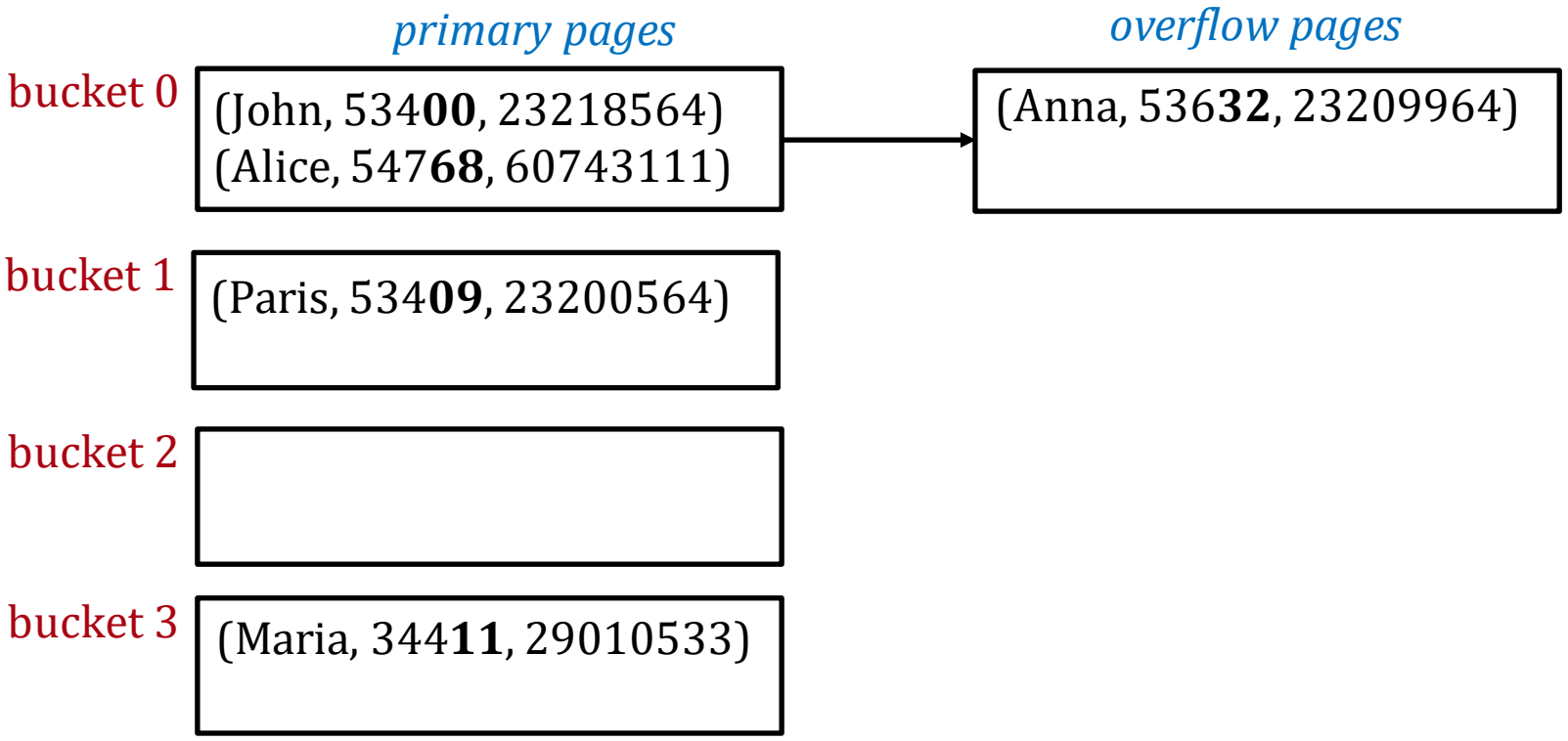
h maps a search key to one of the buckets

Static Hashing – Example

Person(name, zipcode, phone)

- *search key*: {zipcode}
- *hash function* ***h***: last 2 digits mod 4

- 4 buckets
- each bucket holds 2 index entries



Hash Index Operations – Search

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)

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

Hash Index Operations – Search

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)

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

I/O cost if we choose to store RID in index entries?

Hash Index Operations – Search

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)

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

I/O cost if we choose to store RID in index entries?

- Need one more I/O to access the data page. But may have fewer overflow pages

$$\text{I/O cost} = 2 + \#\text{overflow pages}$$

Outline

Static hashing
- Equality search
- **Insert and deletes**
- Hash functions
- Limitations of static hashing

Extendible hashing
- Search
- Insertion and split
- Discussion

Hash Index Operations – Insert/Delete

Insertion

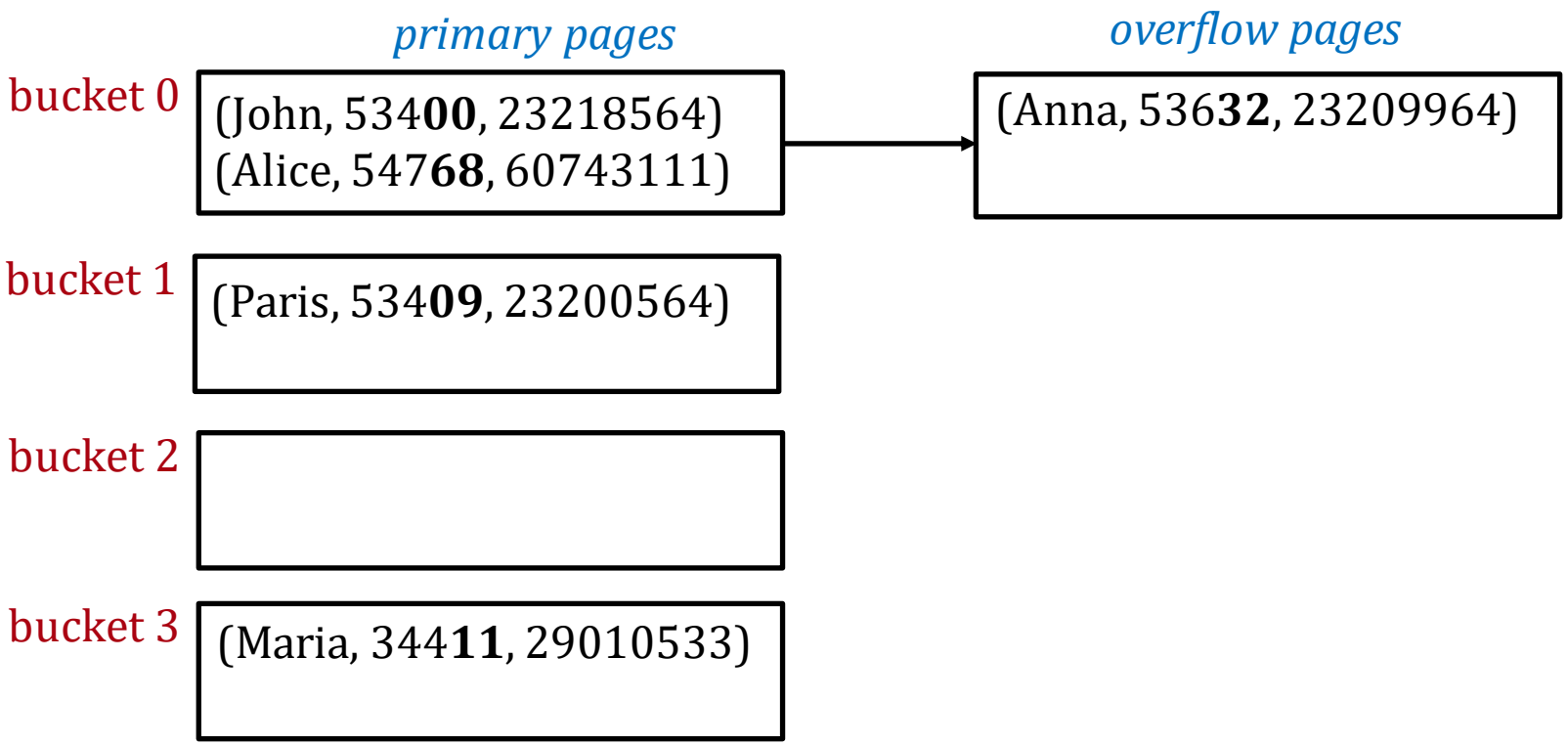
- Find the appropriate bucket, insert the record
- If there is no space, create a new overflow page

Deletion

- Find the appropriate bucket, delete the record
- If it is the last record in an overflow page, remove the page from the chain

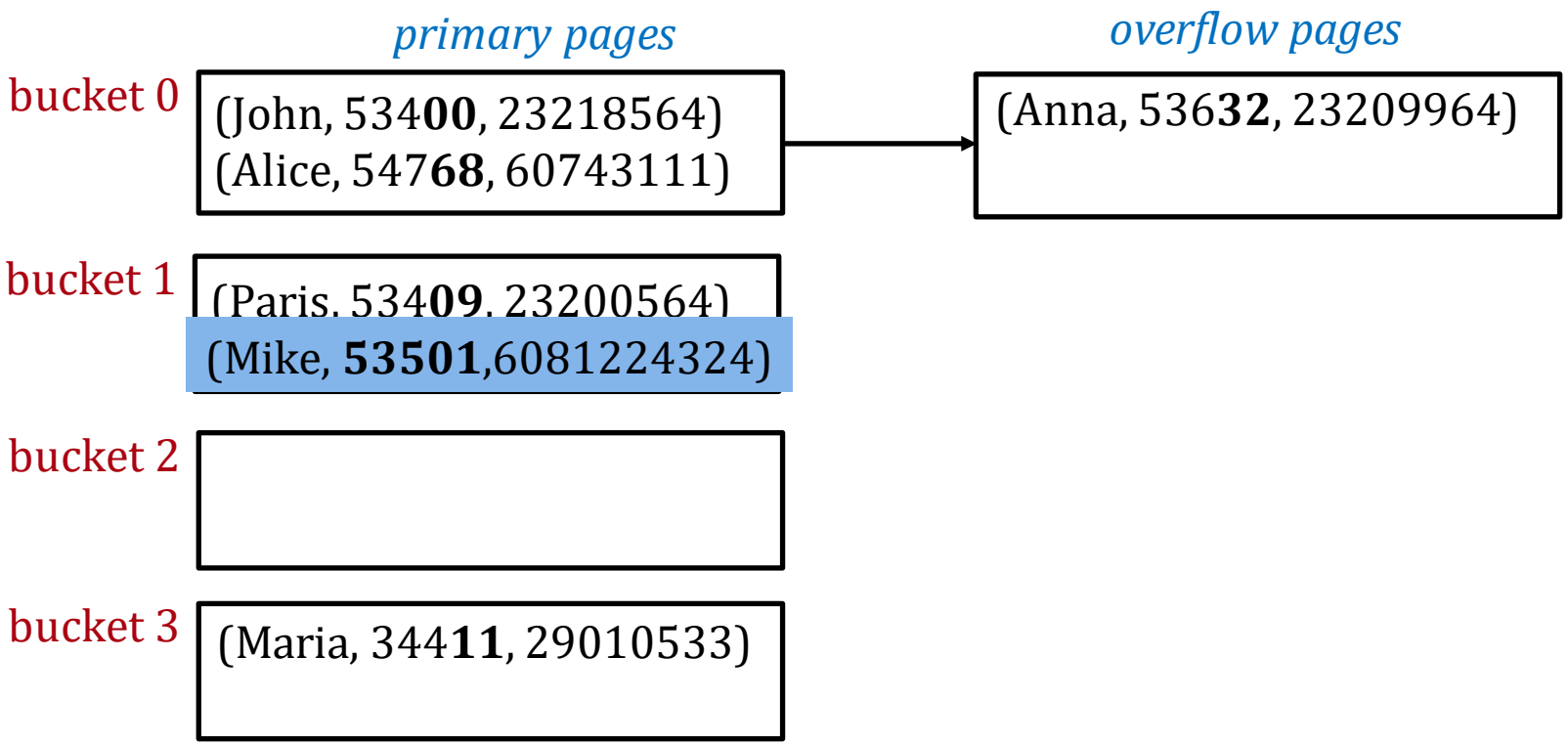
Insertion – Example

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



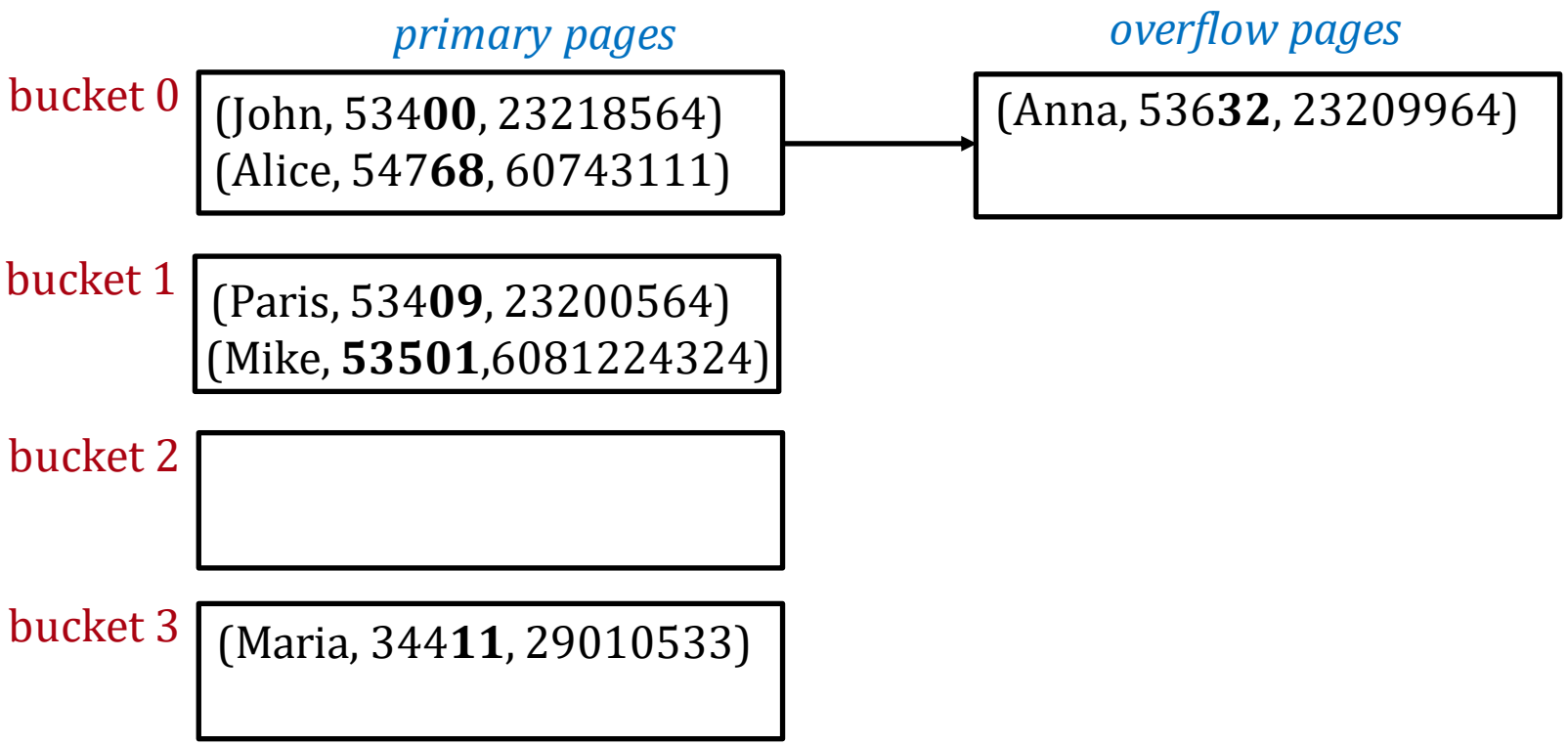
Insertion – Example

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



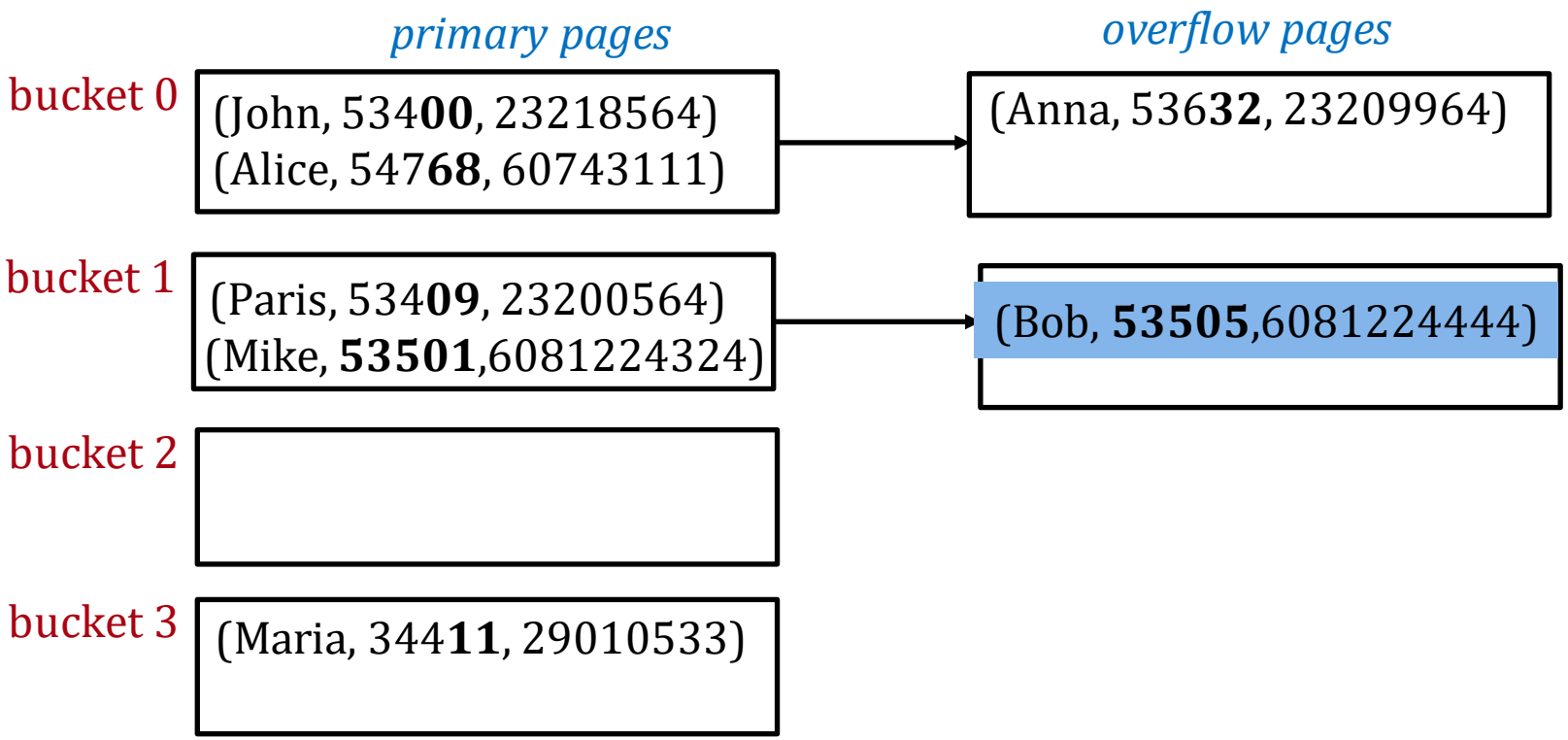
Insertion – Example

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



Insertion – Example

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



Outline

Static hashing
- Equality search
- Insert and deletes
- **Hash functions**
- Limitations of static hashing

Extendible hashing
- Search
- Insertion and split
- Discussion

Hash Functions

A good hash function is

- Uniform: each bucket is assigned the same number of key values
- Fast to compute

Hash Functions

A good hash function is

- Uniform: each bucket is assigned the same number of key values
- Fast to compute

A hash function ***h*** of the form $h(value) = (a \times value + b)$ works well in practice.

- Constants ***a*** and ***b*** can be chosen to tune the hash function

Outline

Static hashing
- Equality search
- Insert and deletes
- Hash functions
- **Limitations of static hashing**

Extendible hashing
- Search
- Insertion and split
- Discussion

Limitations of Static Hashing

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

- 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

Limitations of Static Hashing

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

- 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

What about rehashing?

- Double the bucket count and redistribute data?
- Problem 1: expensive computation and data movement
- Problem 2: block query execution (cannot access hash table during rehashing)

Outline

Static hashing

- Equality search
- Insert and deletes
- Hash functions
- Limitations of static hashing

Extendible hashing

- Search
- Insertion and split
- Discussion

Extendible Hashing

Intuition: when a bucket overflows, reorganize only the overflowing bucket (rather than all buckets)

Extendible Hashing

Intuition: when a bucket overflows, reorganize only the overflowing bucket (rather than all buckets)

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)

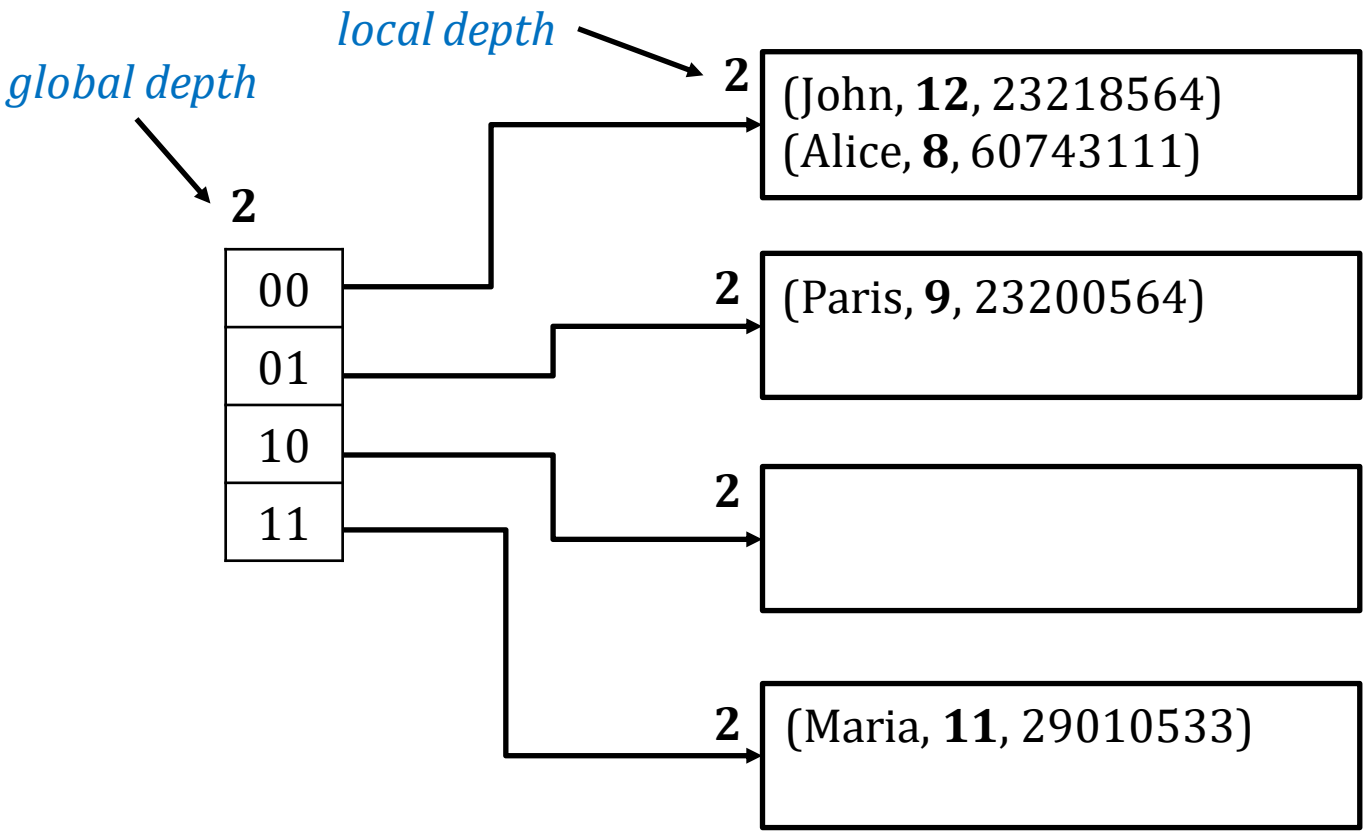
Outline

Static hashing
- Equality search
- Insert and deletes
- Hash functions
- Limitations of static hashing

Extendible hashing
- **Search**
- Insertion and split
- Discussion

Extendible Hashing – Search

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



Outline

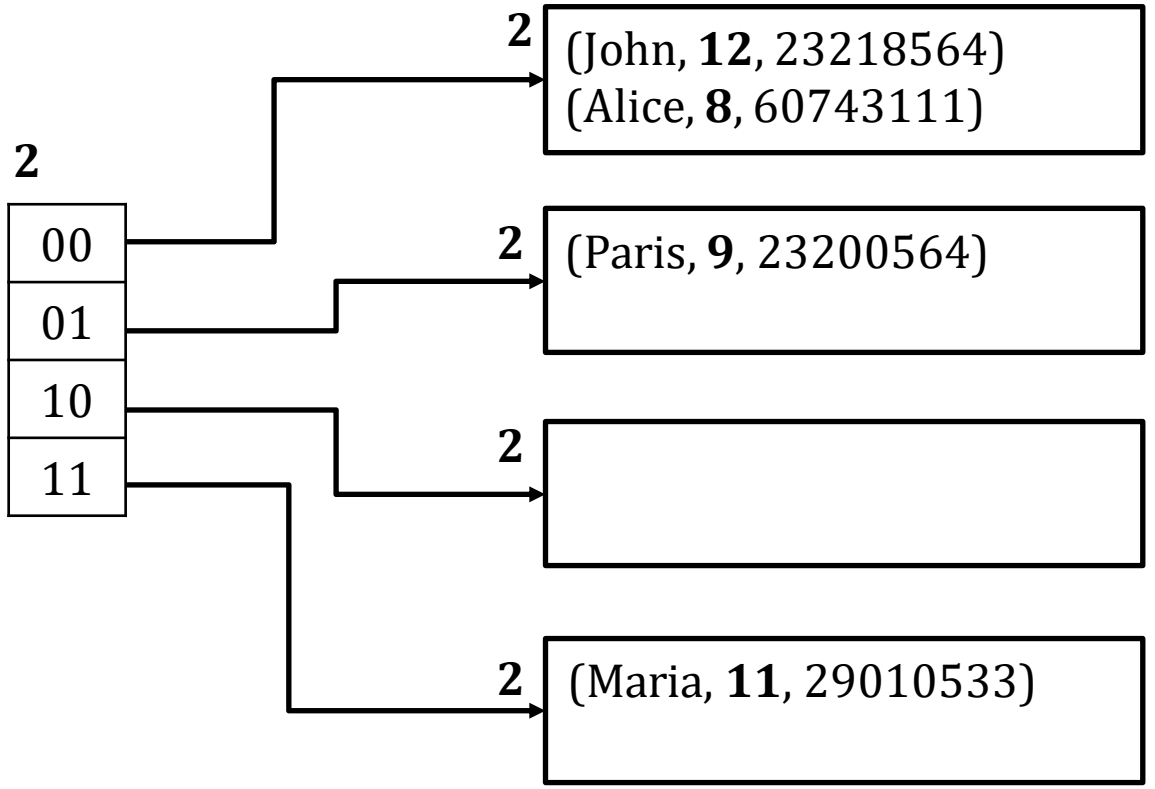
Static hashing
- Equality search
- Insert and deletes
- Hash functions
- Limitations of static hashing

Extendible hashing
- Search
- **Insertion and split**
- Discussion

Extendible Hashing – Insert

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

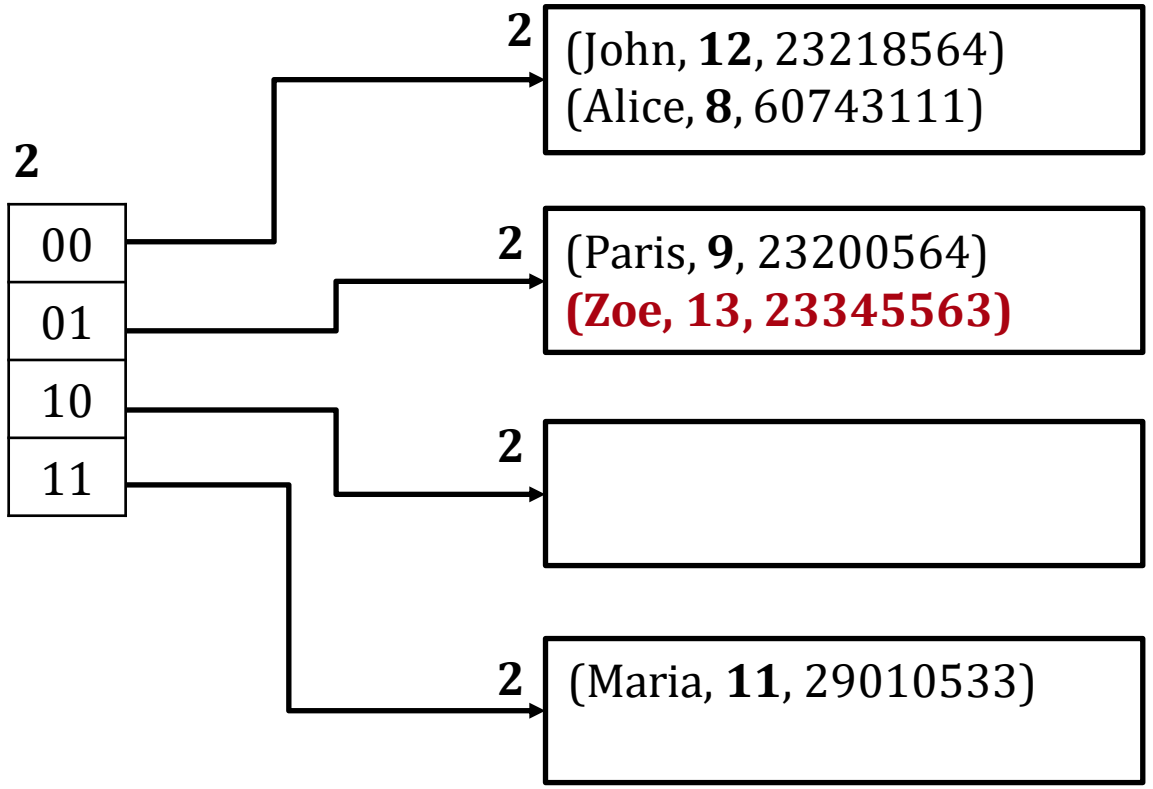
insert: (Zoe, 13, 23345563)



Extendible Hashing – Insert

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

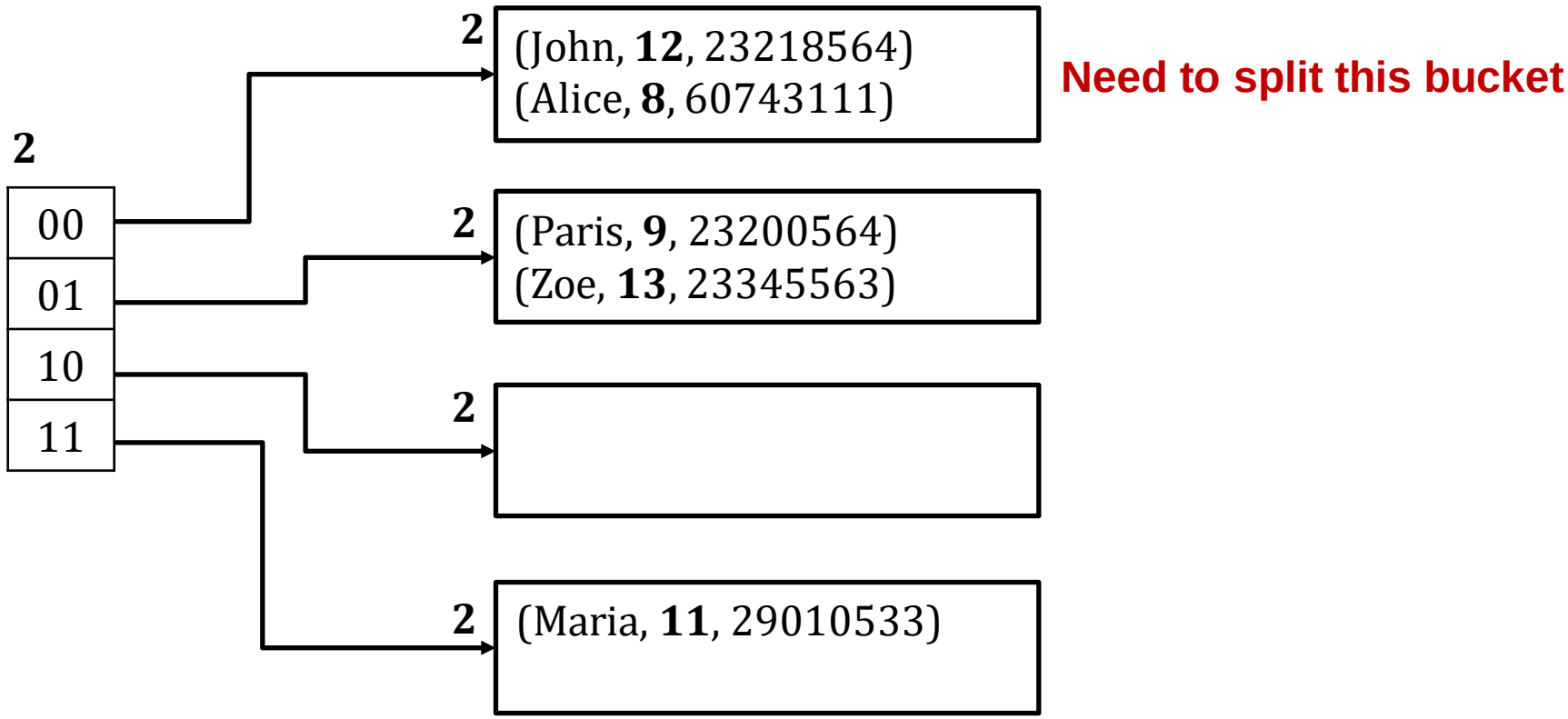
insert: (Zoe, 13, 23345563)



Extendible Hashing – Insert

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

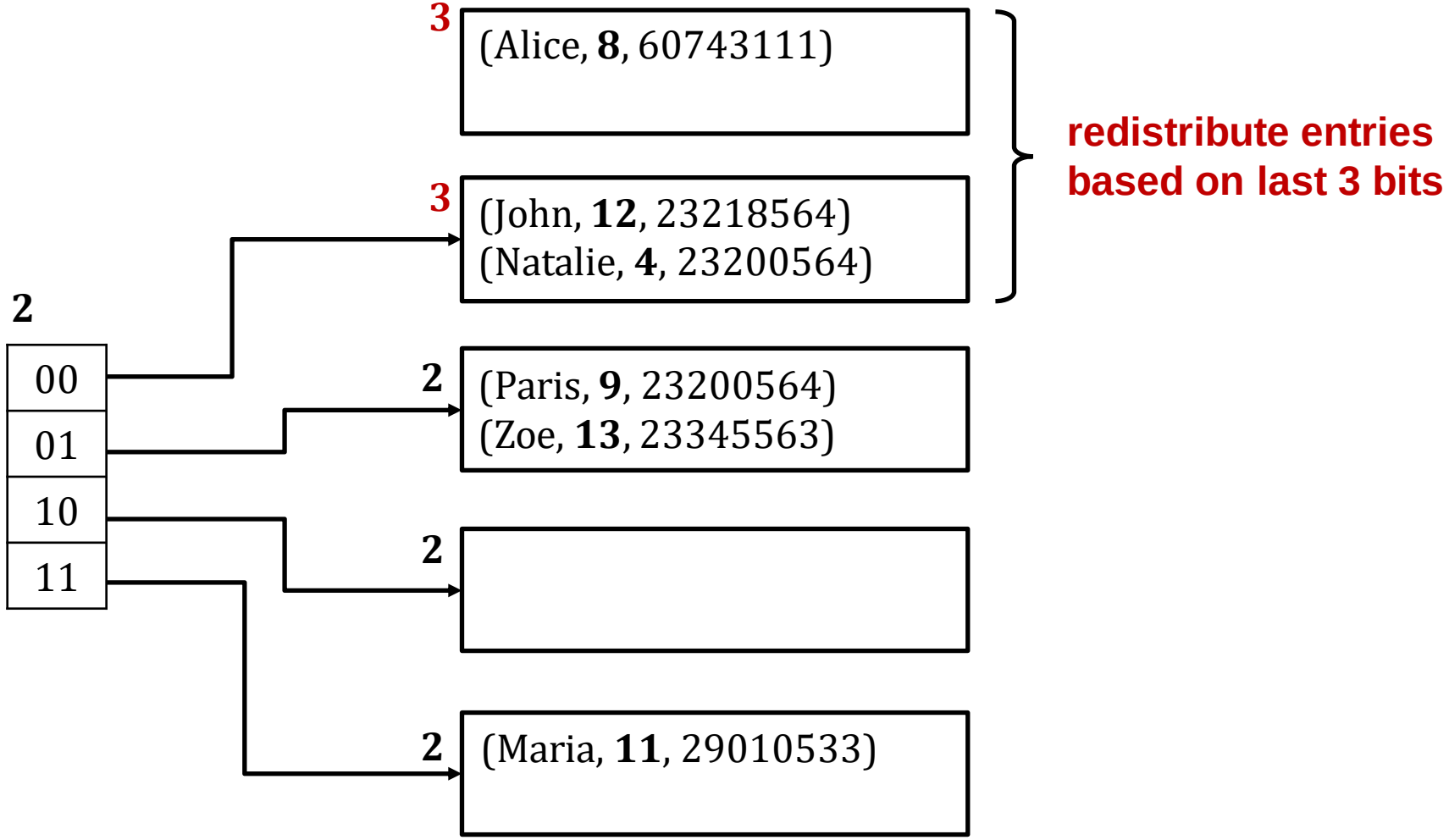
insert: (Natalie, 4, 23200564)



Bucket Split

insert: (Natalie, 4, 23200564)

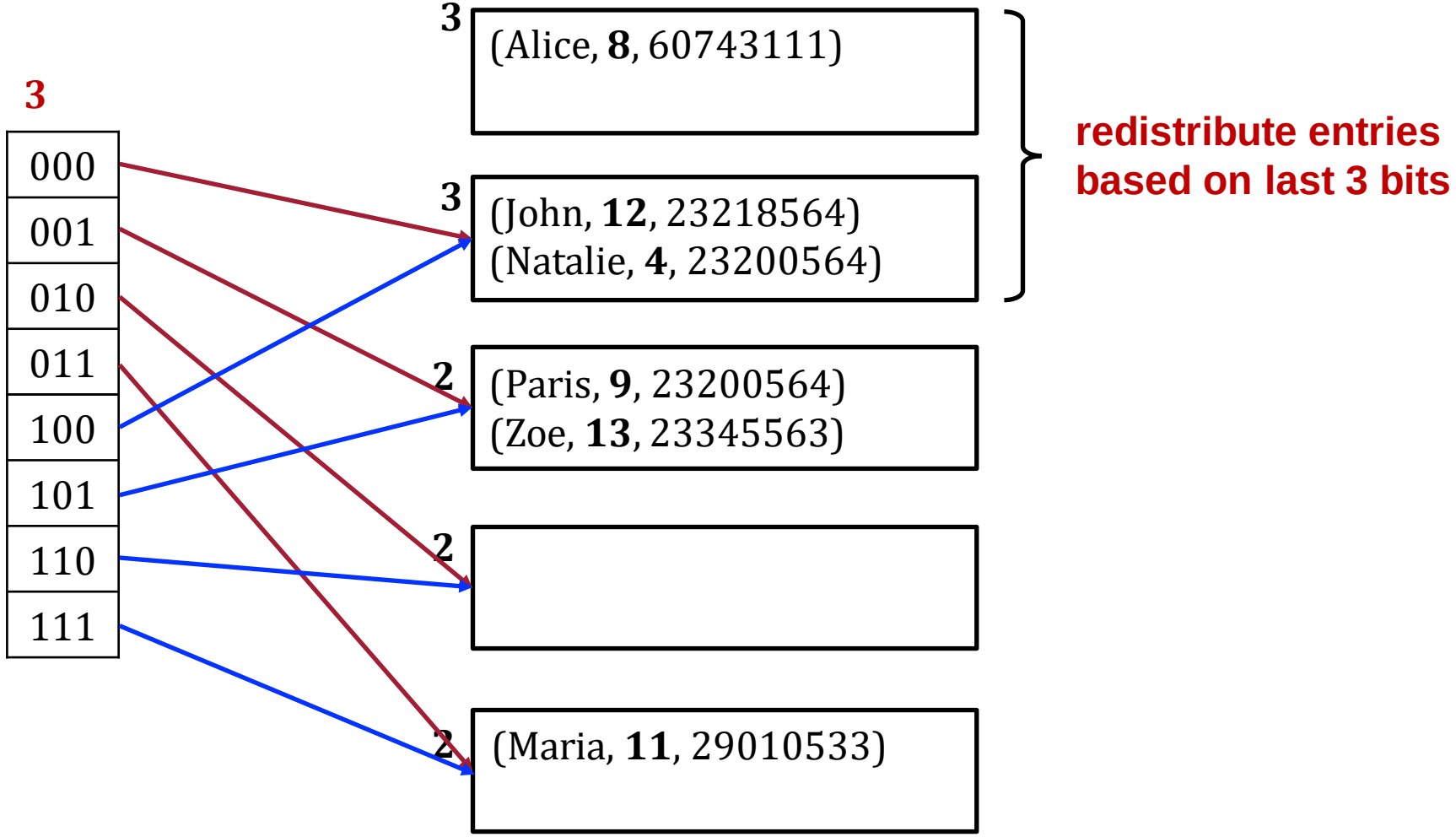
Step 1: allocate a new bucket and redistribute entries (consider the last 3 bits)



Bucket Split

insert: (Natalie, 4, 23200564)

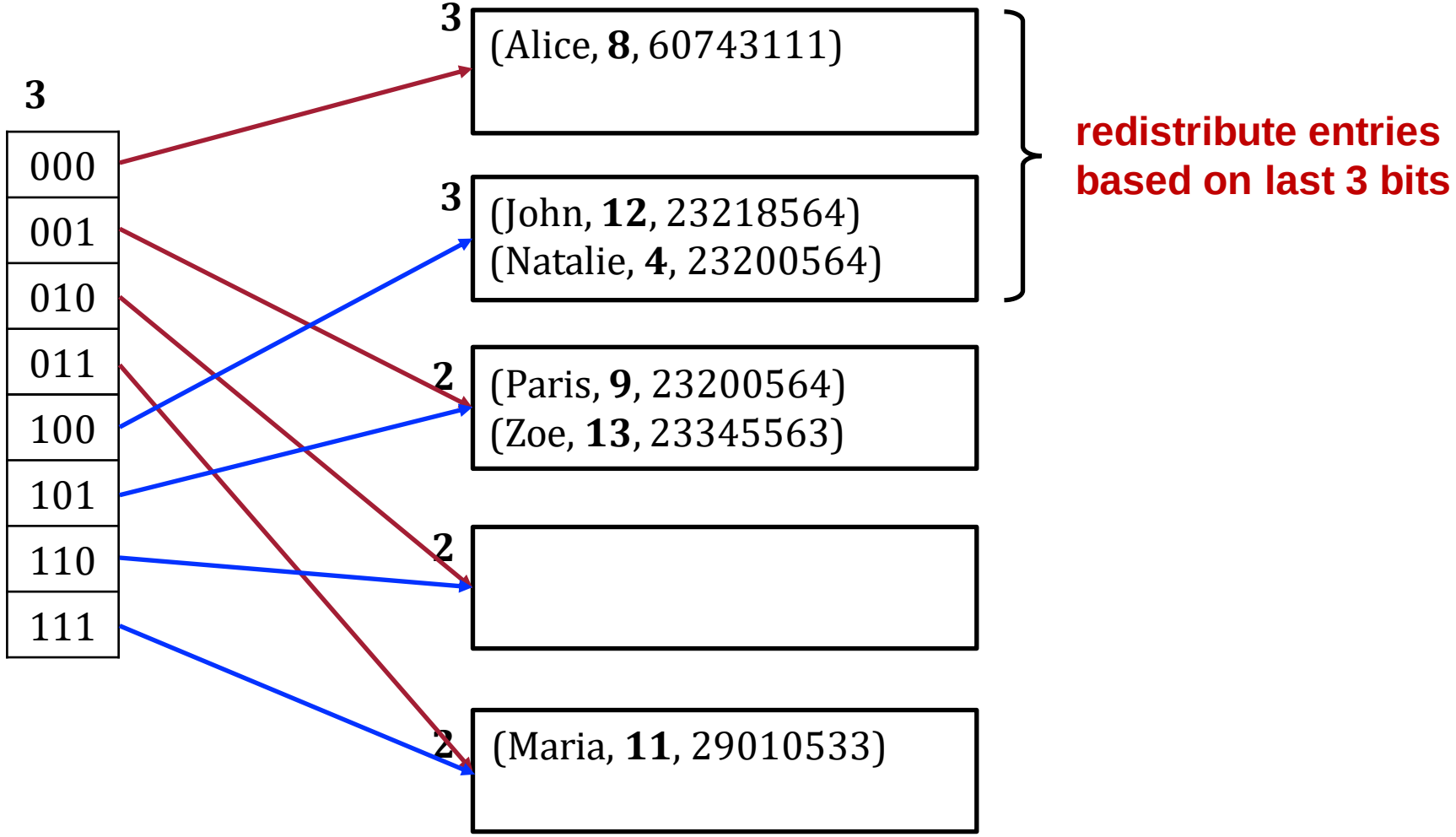
Step 2: double the directory



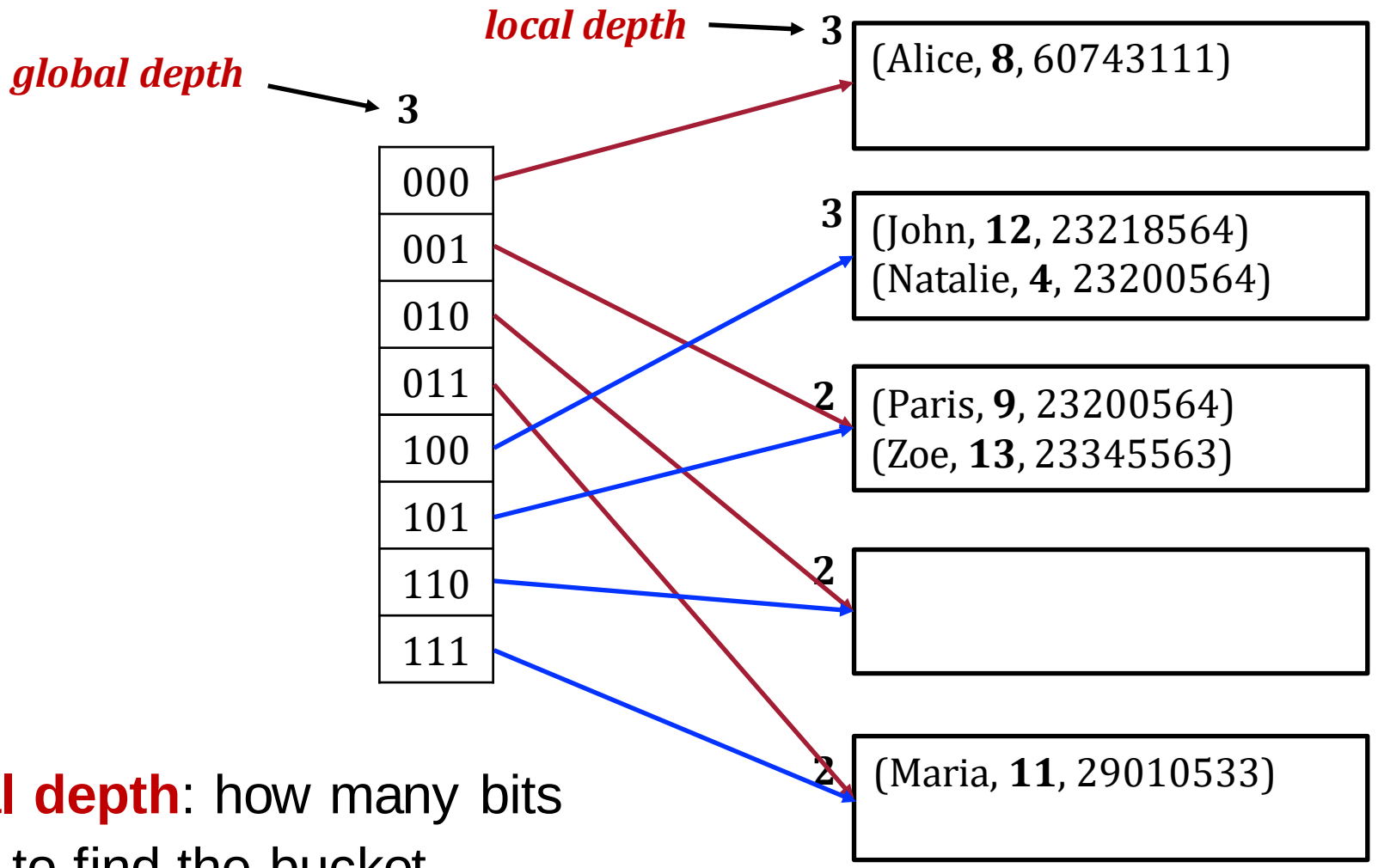
Bucket Split

insert: (Natalie, 4, 23200564)

Step 2: double the directory



Global and Local Depth



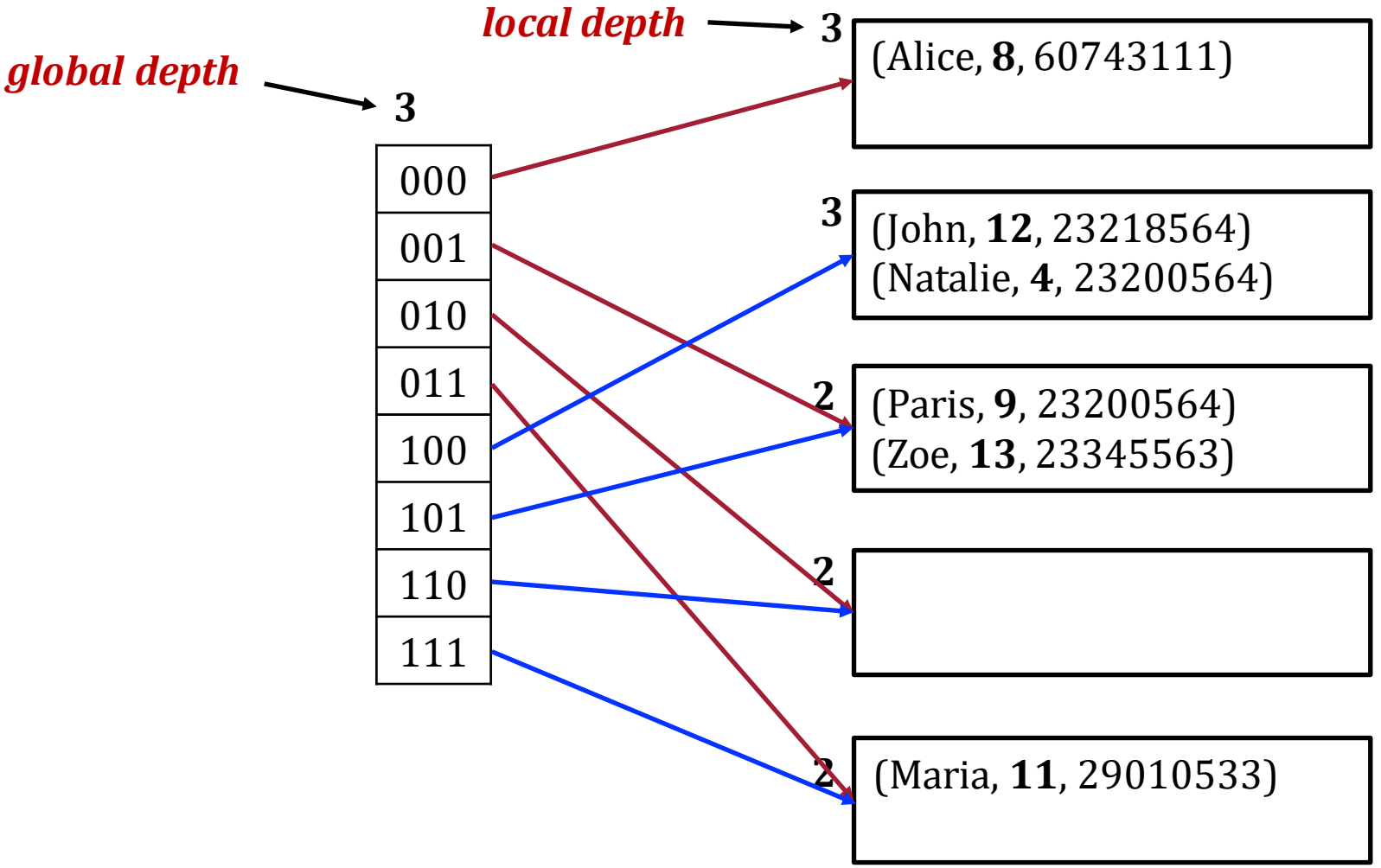
Local depth

- Incremented upon bucket split
- Double directory if local depth = global depth and bucket is full

Global depth: how many bits to use to find the bucket

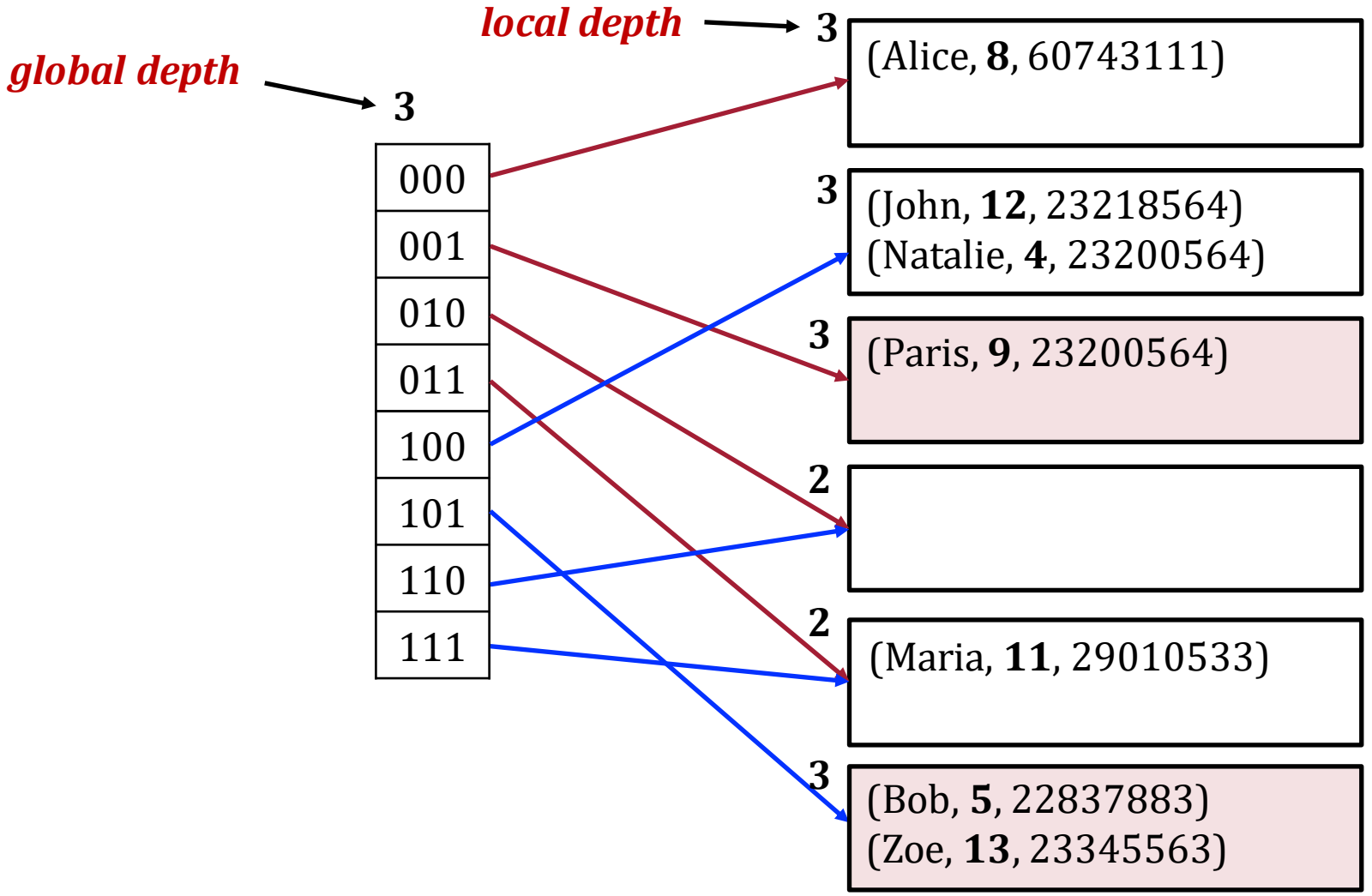
Insert – Example

insert: (Bob, 5, 22837883)



Insert – Example

insert: (Bob, 5, 22837883)



Outline

Static hashing
- Equality search
- Insert and deletes
- Hash functions
- Limitations of static hashing

Extendible hashing
- Search
- Insertion and split
- **Discussion**

Cost of Doubling Directory

The directory is much smaller than the buckets and can typically fit in memory

For example, 100MB file where each bucket has size of 4KB

- How many entries in the directory? 100MB/4KB = 25,000

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

Summary

Static hashing
- Equality search
- Insert and deletes
- Hash functions
- Limitations of static hashing

Extendible hashing
- Search
- Insertion and split
- Discussion