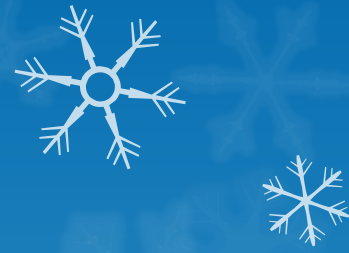


Course Name:
Database Management
Systems



Lecture 12

Topics to be covered

- ❑ *File Structures*
- ❑ **Sequential Files**
- ❑ **Indexed Sequential Files**



File

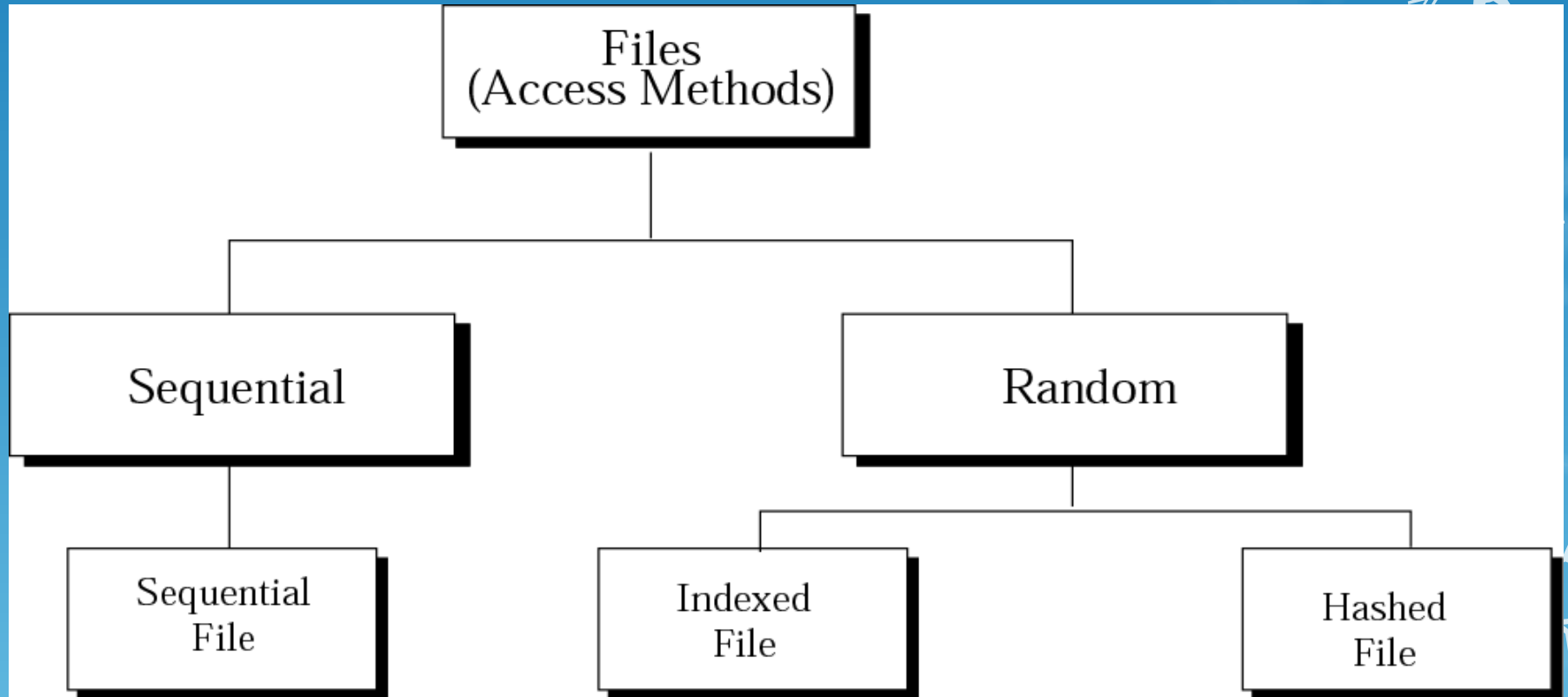
- A **file** is an external collection of related data treated as a unit.
- Files are stored in **auxiliary/secondary storage devices**.
 - Disk
 - Tapes
- A **file** is a collection of data records with each record consisting of one or more fields.



***ACCESS
METHODS***

Taxonomy of file structures

The **access method** determines how records can be retrieved: **sequentially** or **randomly**.



- One record after another, from beginning to end

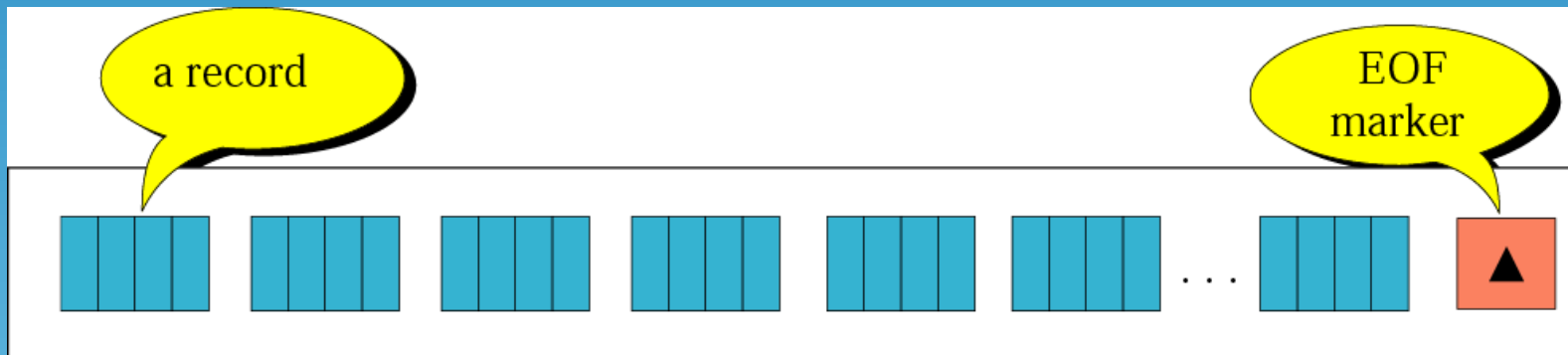
- Access one specific record without having to retrieve all records before it



***SEQUENTIAL
FILES***

Sequential file

Sequential file – records can only be accessed **sequentially**, one after another, from beginning to end.



Applications

- Applications – that need to access all records from beginning to end.
 - Personal information
- Because you have to process each record, sequential access is more **efficient** and **easier** than random access.
- Sequential file is not efficient for random access.

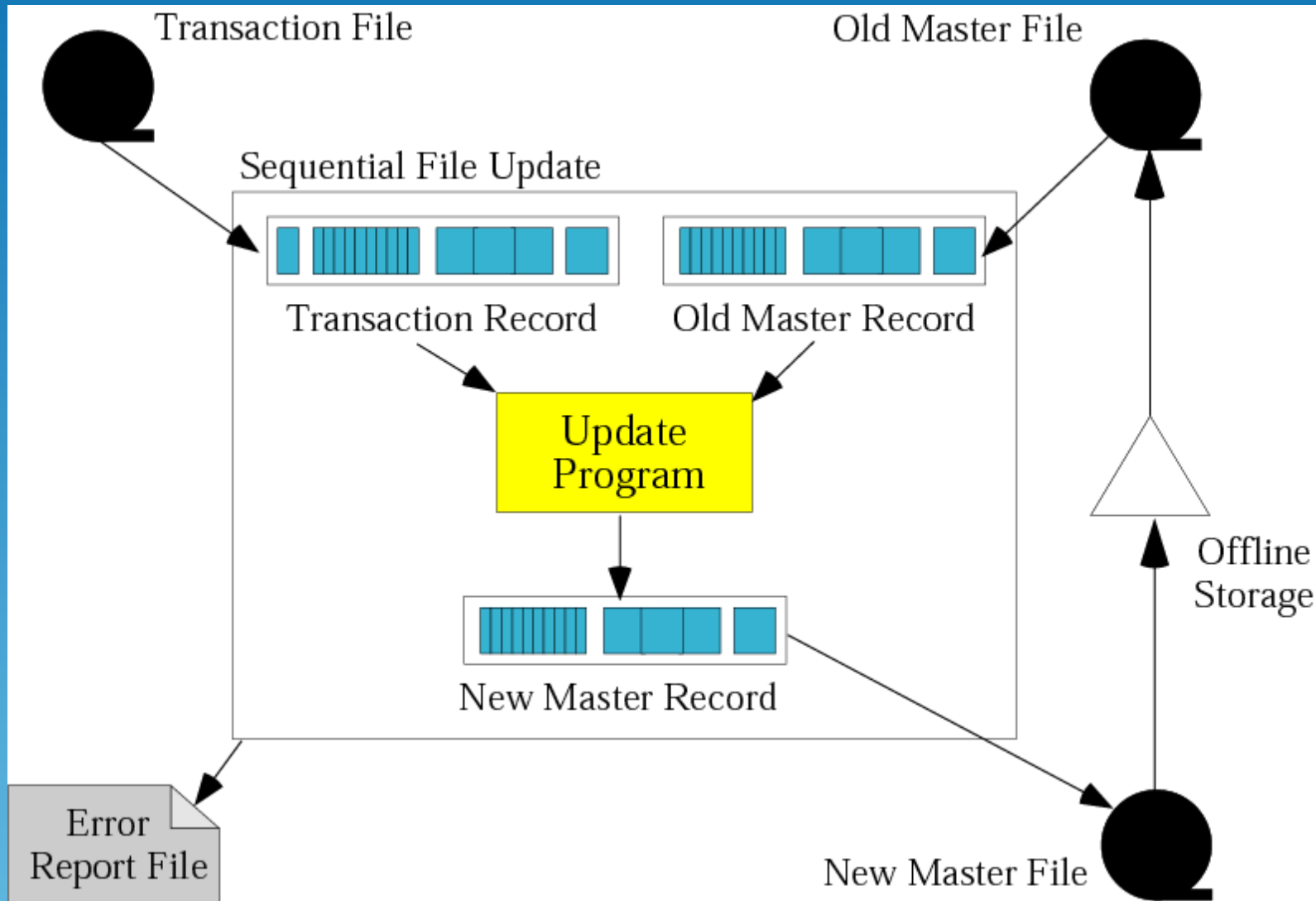
Updating sequential files



- sequential files must be updated periodically to reflect changes in information.
- The **updating process** – all of the records need to be checked and updated (if necessary) sequentially.
 - **New Master File**
 - **Old Master File**
 - **Transaction File** – contains changes to be applied to the master file.
 - **Add transaction**
 - **Delete transaction**
 - **Change transaction**
 - A **key** is one or more fields that uniquely identify the data in the file.
 - **Error Report File**



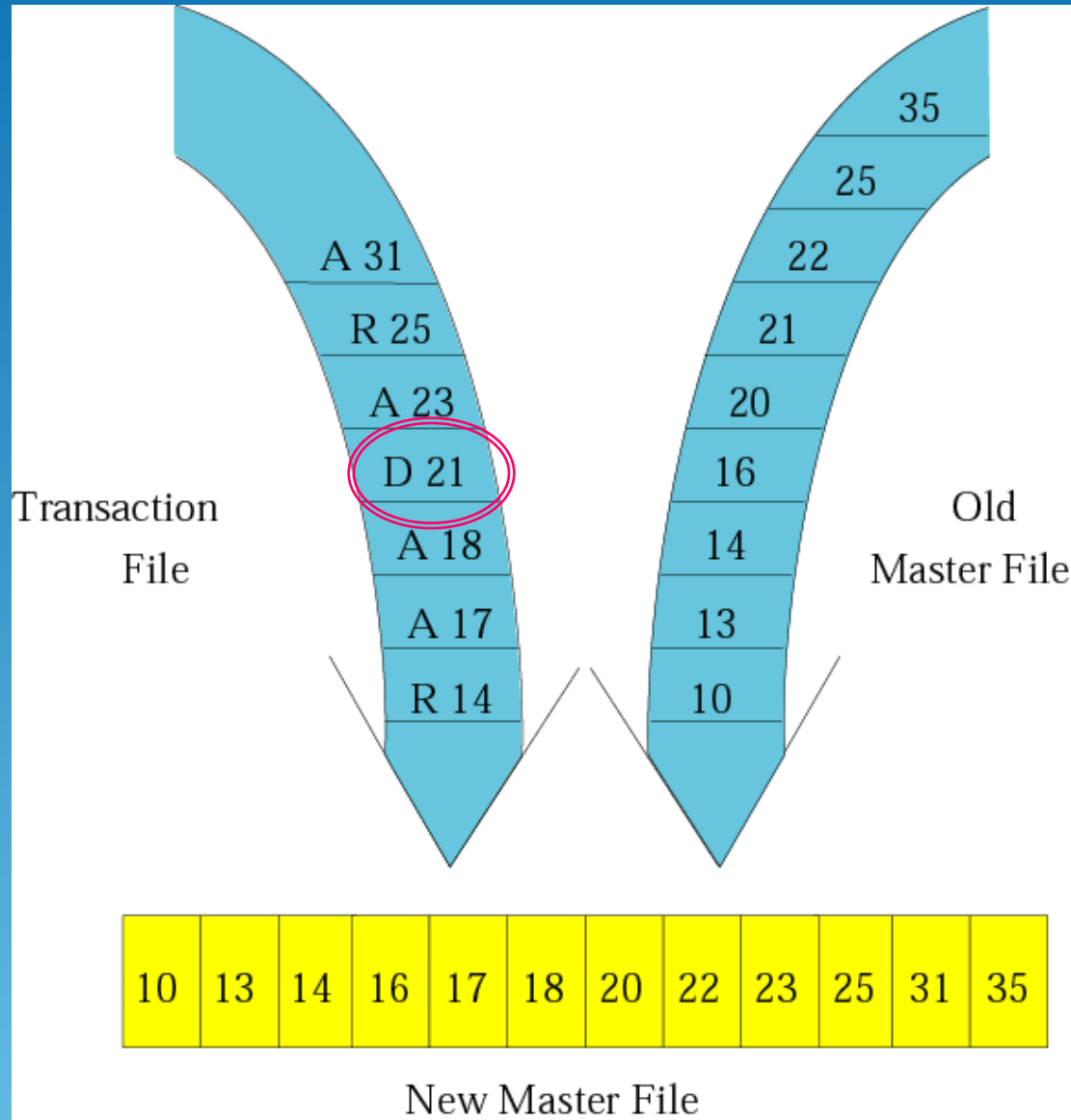
Updating a sequential file



Updating sequential files

- To make updating process efficient, all files are sorted on the same key.
- The update process requires that you compare :
[transaction file key] vs. [old master file key]
 - $<$: add transaction to new master
 - $=$:
 - Change content of master file data (transaction code = R(revise))
 - Remove data from master file (transaction code = D(delete))
 - $>$: write old master file record to new master file
(transaction code = A(add))

Updating process

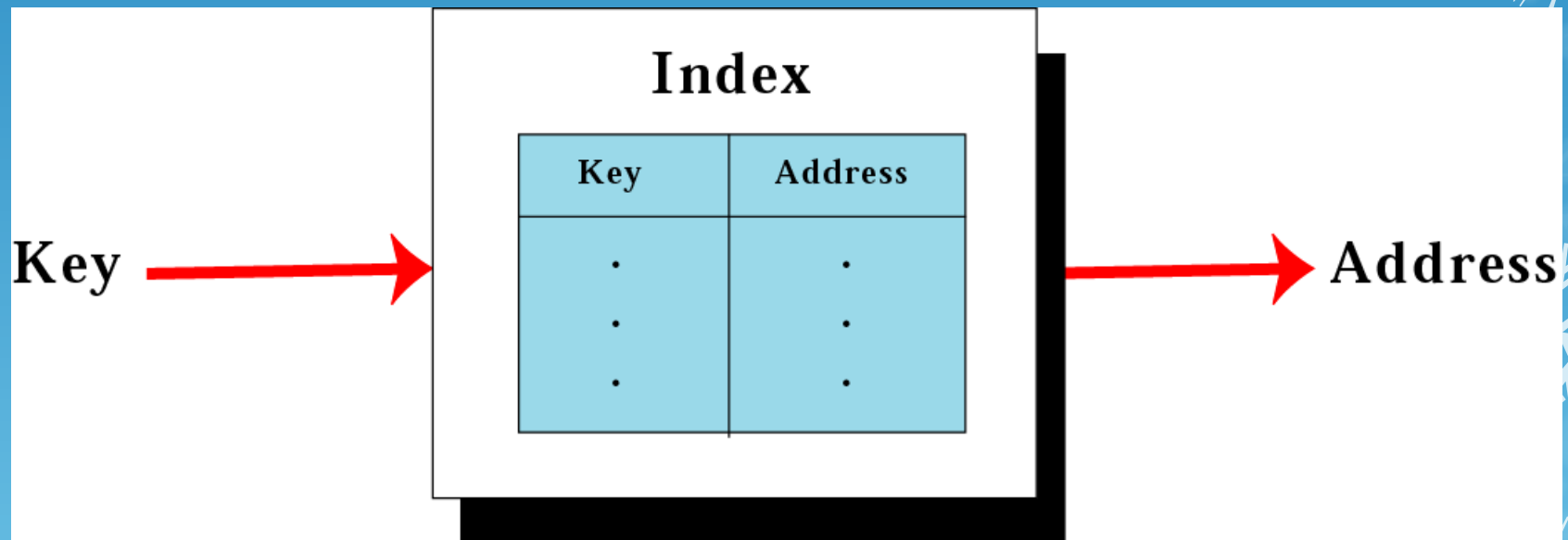




***INDEXED
FILES***

Mapping in an **indexed file**

- To access a record in a file **randomly**, you need to know the **address** of the record.
- An **index file** can relate the **key** to the **record address**.



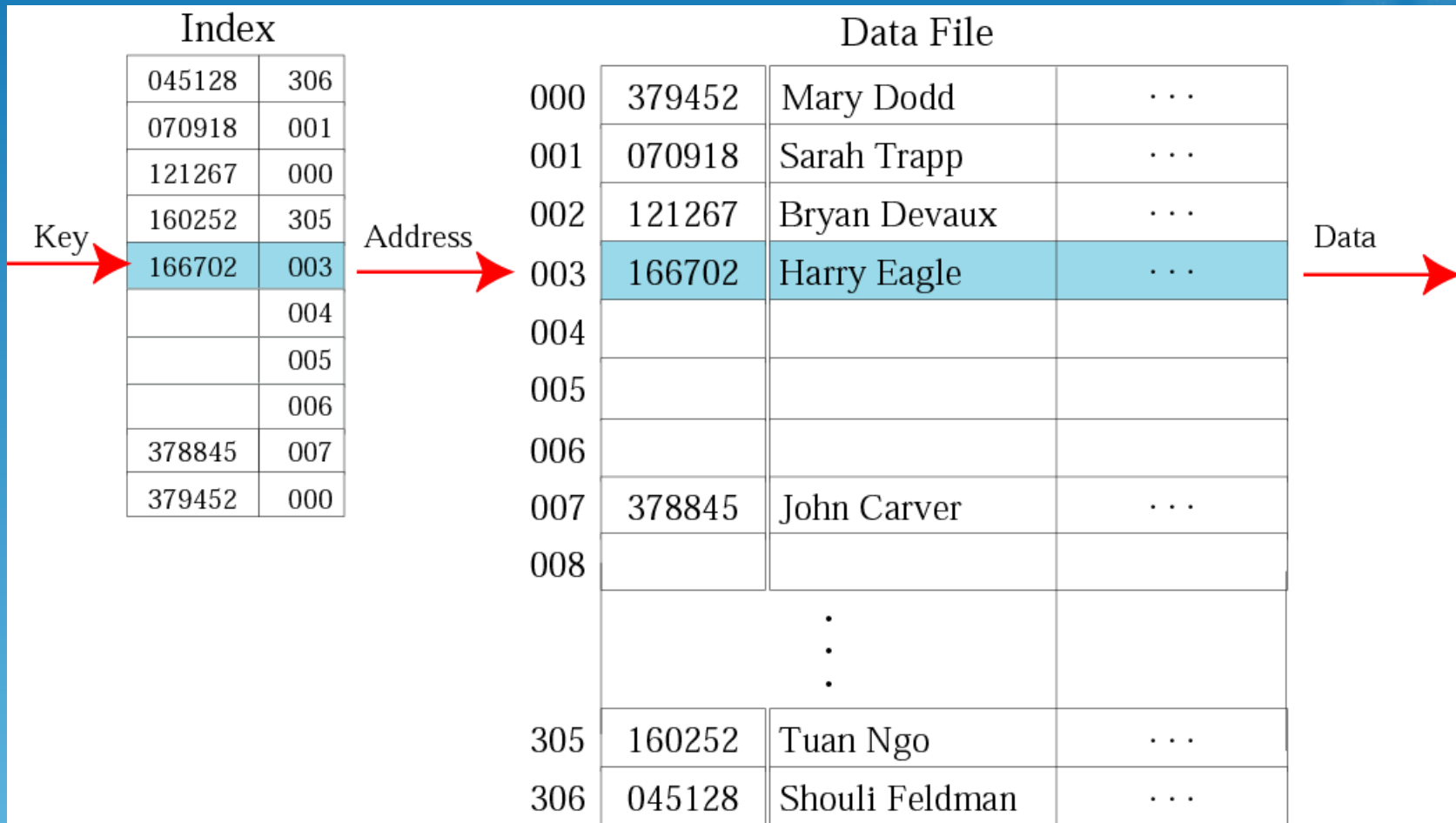
Indexed files

- An **index file** is made of a **data file**, which is a sequential file, and an **index**.
- **Index** – a small file with only two fields:
 - The **key** of the sequential file
 - The **address** of the corresponding record on the disk.
- To access a record in the file :
 1. **Load** the entire index file into main memory.
 2. **Search** the index file to find the desired key.
 3. **Retrieve** the address the record.
 4. **Retrieve** the data record. (using the address)
- **Inverted file** – you can have more than one **index**, each with a different key.

Inverted file

- A file that reorganizes the structure of an existing data file to enable a rapid search to be made for all records having one field falling within set limits.
- For example, a file used by an estate agent might store records on each house for sale, using a reference number as the key field for sorting. One field in each record would be the asking price of the house. To speed up the process of drawing up lists of houses falling within certain price ranges, an inverted file might be created in which the records are rearranged according to price. Each record would consist of an asking price, followed by the reference numbers of all the houses offered for sale at this approximate price.

Logical view of an indexed file





***HASHED
FILES***

Mapping in a **hashed file**



A **hashed file** uses a **hash function** to map the key to the address.

- Eliminates the need for an extra file (**index**).
- There is no need for an index and all of the overhead associated with it.



Hashing methods



Direct Hashing –

the key is the address without any algorithmic manipulation.

- **Modulo Division Hashing** – (Division remainder hashing)

divides the key by the file size and use the remainder plus 1 for the address.

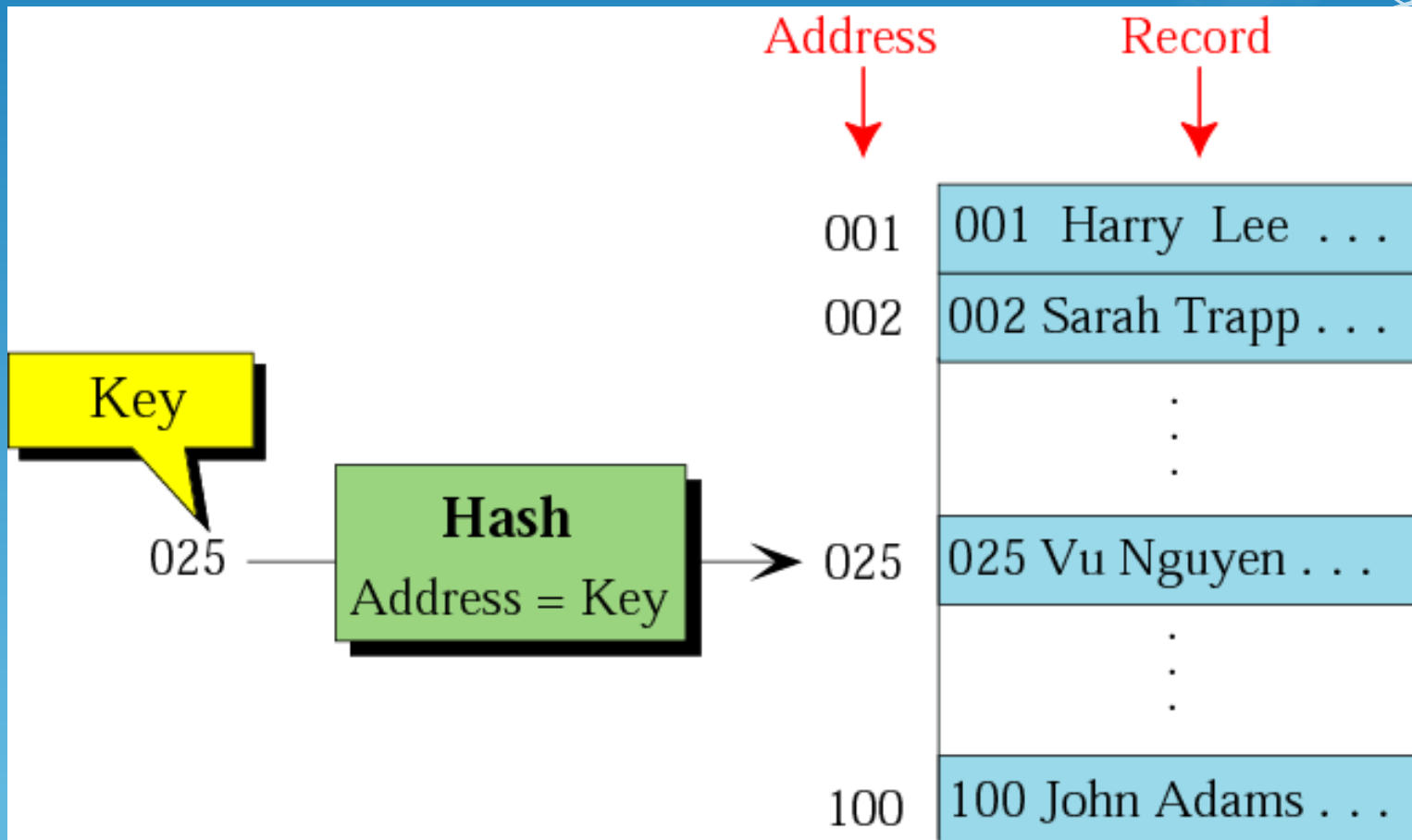
- **Digit Extraction Hashing** –

selected digits are extracted from the key and used as the address.

Direct hashing



Direct Hashing –
the key is the address without any algorithmic manipulation.



Direct Hashing



the file must contain a record for every possible key.

Adv. – no **collision**.

Disadv. – space is wasted.

Hashing techniques –

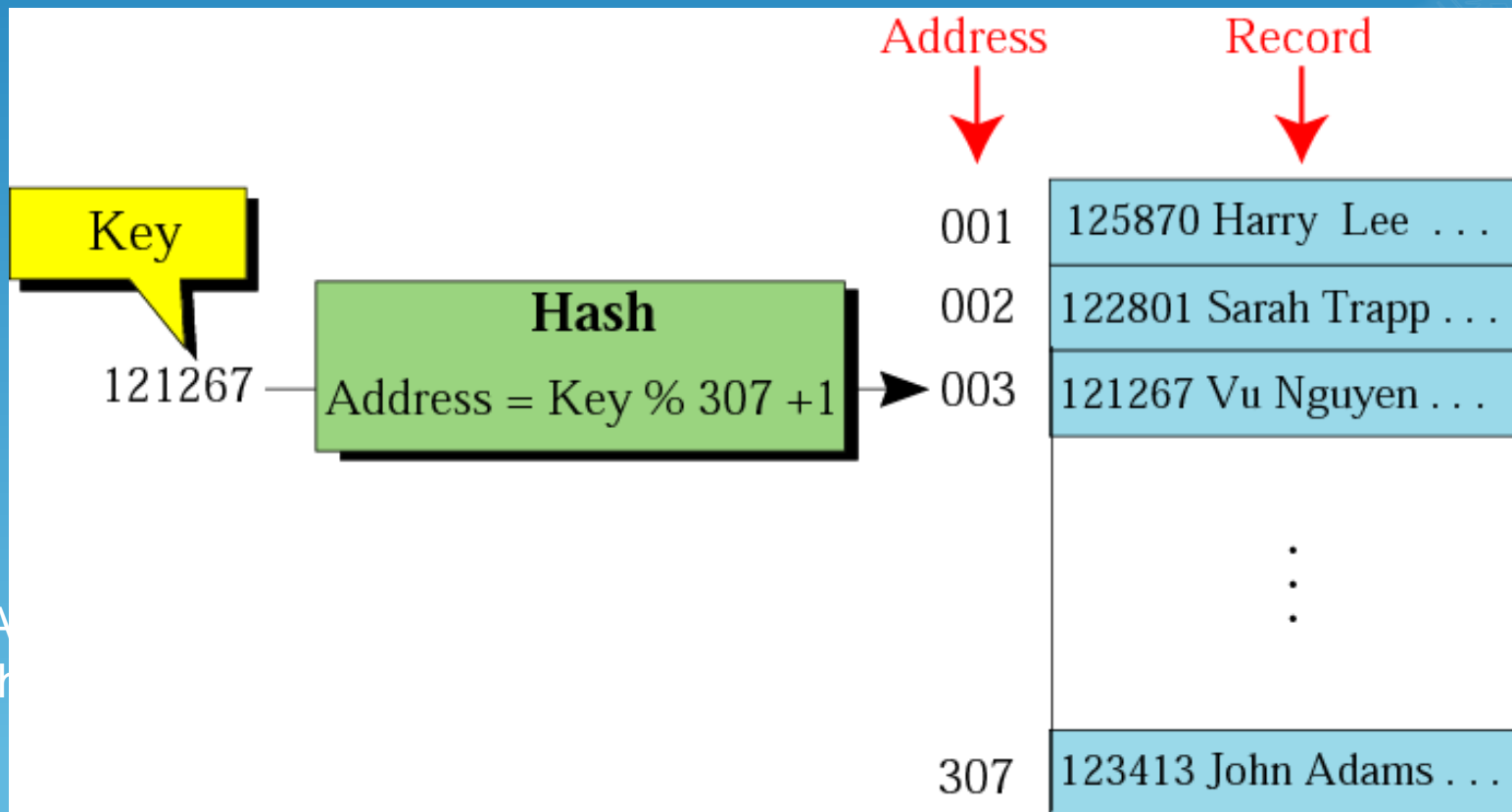
map a large population of possible keys into
a small address space.

Modulo division



$$\text{address} = \text{key} \% \text{list_size} + 1$$

list_size : a prime number produces fewer collisions



A
th



Digit Extraction Hashing



selected digits are extracted from the key and used as the address.



■ For example :

1,3,4

6-digit employee number → → → 3-digit address

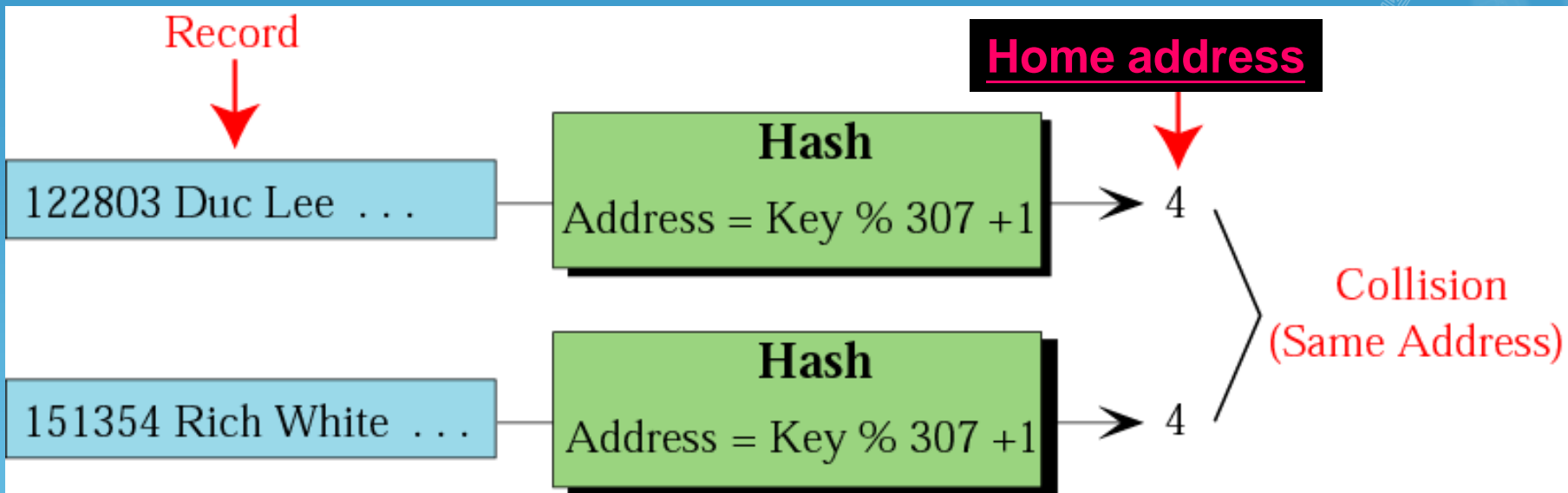
- 125870 → 158
- 122801 → 128
- 121267 → 112
- ...
- 123413 → 134



Collision



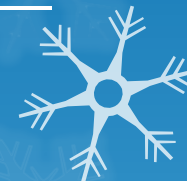
- Because there are many keys for each address in the file, there is a possibility that more than one key will hash to the same address in the file.
- **Synonyms** – the set of keys that hash to the same address.
- **Collision** – a hashing algorithm produces an address for an insertion key, and that address is already occupied.
- **Prime area** – the part of the file that contains all of the home addresses.



Collision Resolution



With the exception of the **directed hashing**, **none** of the methods we discussed creates one-to-one mapping.



- Several **collision resolution** methods :
 - Open addressing resolution
 - Linked list resolution
 - Bucket hashing resolution



Open addressing resolution

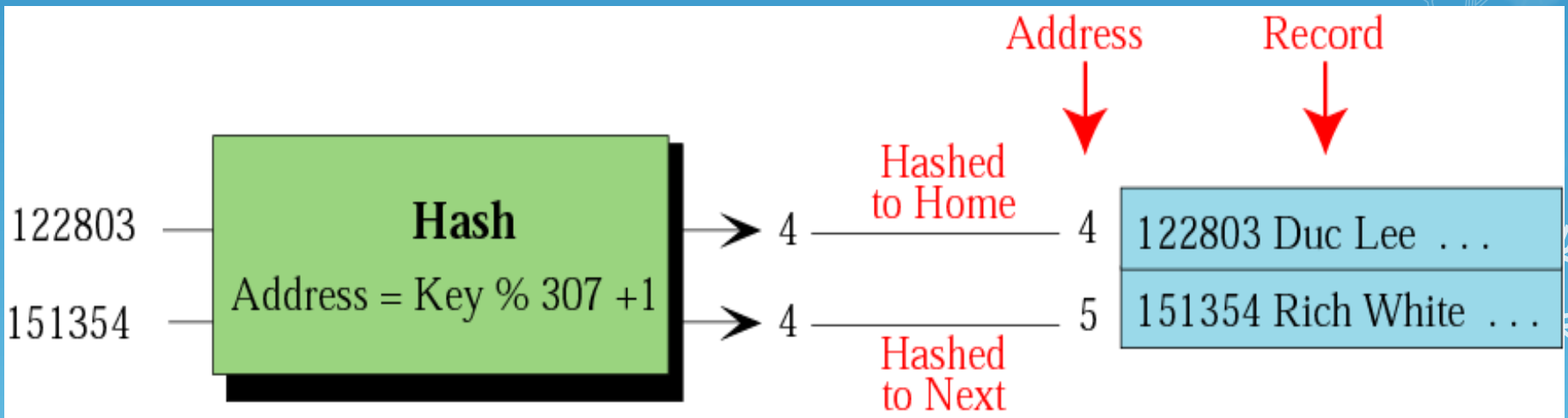


Resolve collisions in the prime area.

The prime area addresses are searched for an open or unoccupied record where the new data can be placed.

One simplest strategy – the next address (home address + 1)

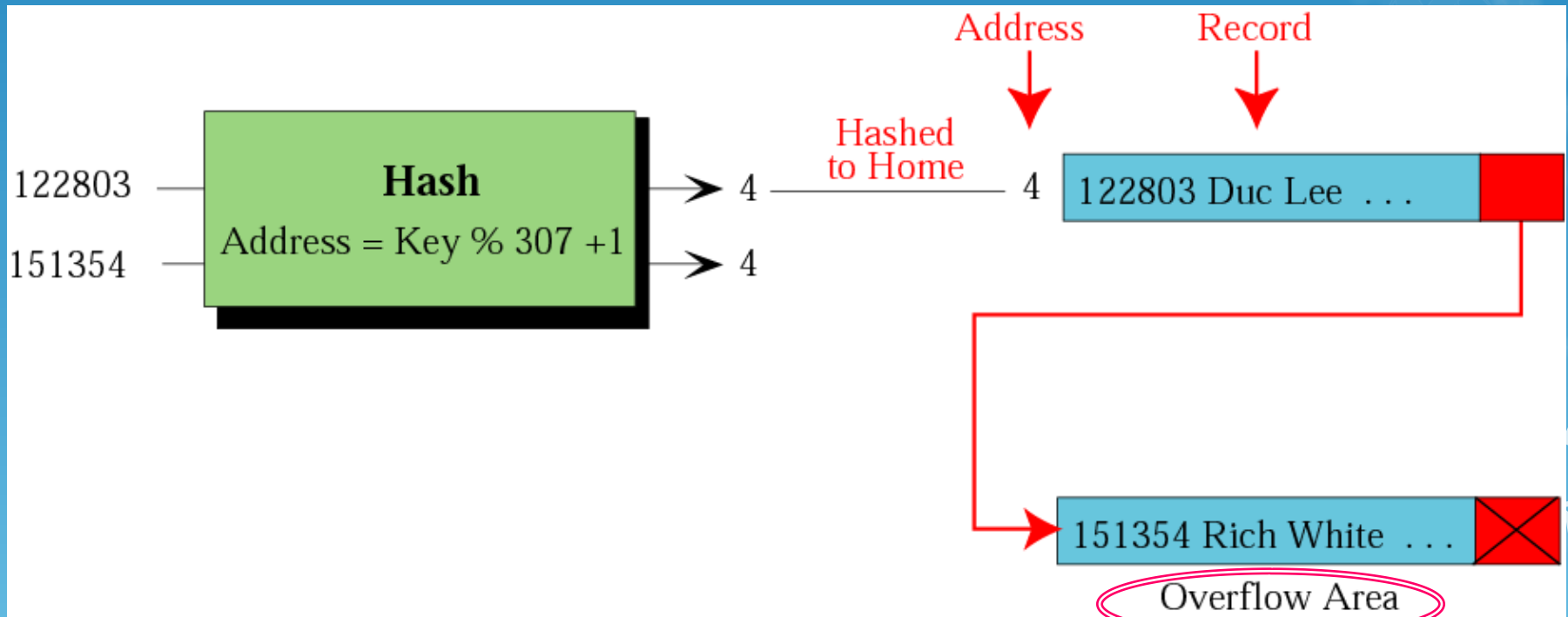
Disadv. – each collision resolution increases the possibility of future collisions.



Linked list resolution



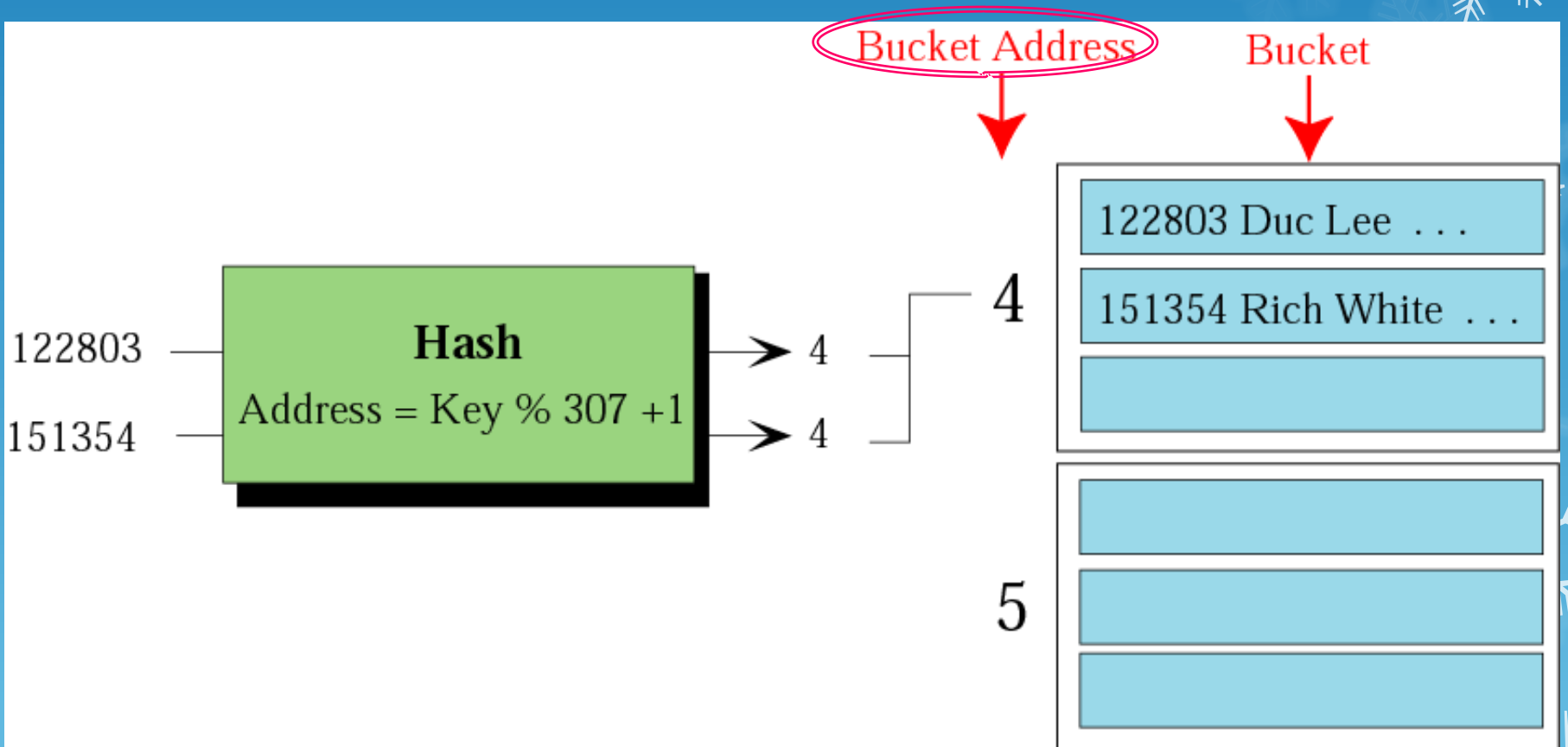
The first record is stored in the home address (**prime area**), but it contains a pointer to the second record. (**overflow area**)



Bucket hashing resolution



Bucket –
a node that can accommodate more than one record.



TEXT
VERSUS
BINARY

Text and binary interpretations of a file

A **file** stored on a storage device is a sequence of bits that can be interpreted by an application program as a **text file** or a binary file.

Interpreted as a text file

01000001 01000010

Two bytes represent
two characters
(A and B)

Interpreted as a binary file

01000001 01000010

Two bytes represent
one number
(16706)

Text vs. Binary



Text files –

- ❑ A file of **characters**.
- ❑ Cannot contain integers, floating-point numbers, or any other **data structures** in their internal memory format.
- ❑ Encoding system – **ASCII** or **EBCDIC** ...



■ Binary files –

- ❑ A collection of data stored in the **internal format** of the computer.
- ❑ Contain data that are **meaningful** only if they are properly **interpreted** by a program.

