


# Direct File Organization

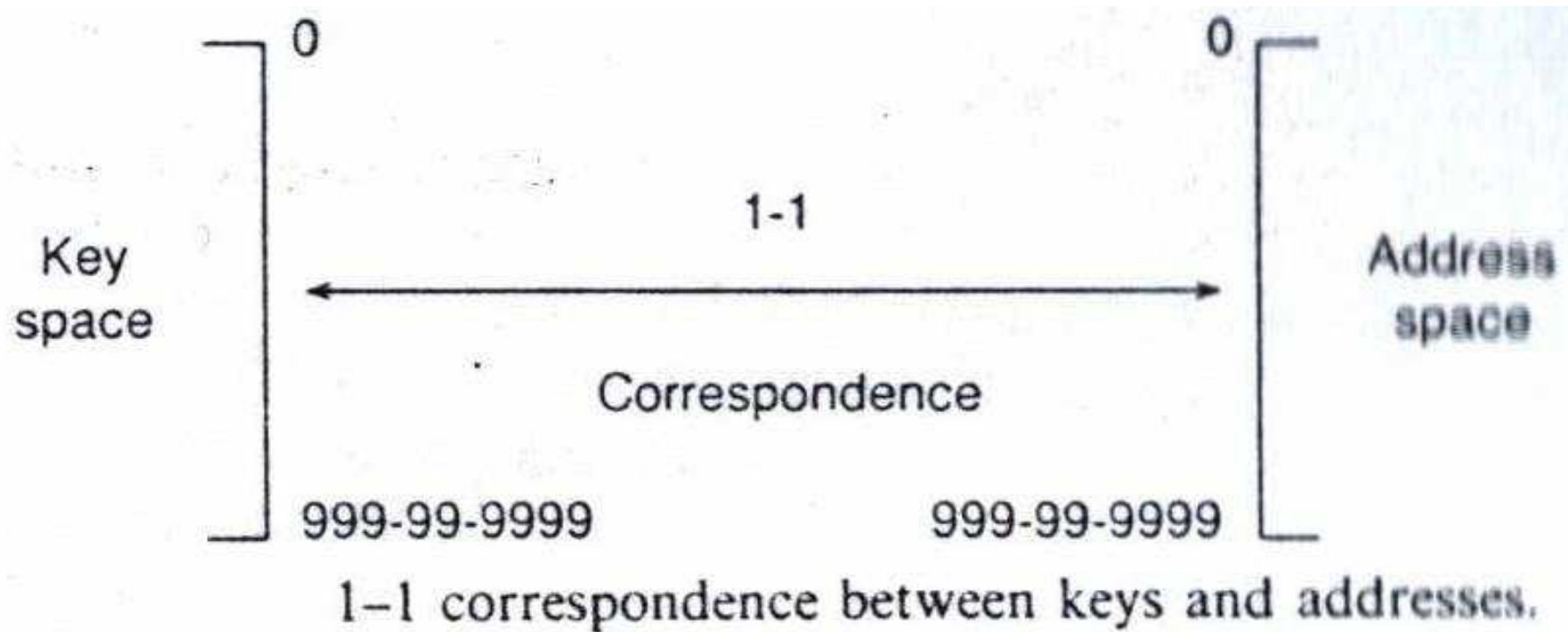
# Locating Information

- Ways to organize a file for direct access:
  - The key is a unique address.
  - The key converts to a unique address.
  - The key converts to a probable address (hashing).

# The Key Is a Unique Address

- We want to go *directly* to the address where the record is stored. This is possible if the key were also an address. 
- If the employee's nine-digit social security number were the record's primary key, we may have one probe per retrieval using a table of size  $10^9$ .
- The disadvantage is that a great deal of space must be reserved. One location for each *possible* social security number.
- But after employees began leaving the company, gaps would begin to appear.

# The Key Is a Unique Address



# The Key Converts to a Unique Address

- An algorithm to convert the primary key field (possibly a composite) into a unique storage address.
- An example of conversion:

$$\text{Location } A[i] = \alpha + m * s$$

where

$A$  = an array

$i$  = typical element

$\alpha$  = location of first element in array

$m$  = number of elements preceding  $i$ th element.

$s$  = size of an element

# The Key Converts to a Unique Address

- Another example of conversion:
  - Flights are numbered from 1 to 999.
  - Days are numbered from 1 to 366.
  - Flight number and day of the year could be *concatenated* to determine the location.
    - Location = flight number || day\_of\_the\_year
    - The addresses would range from 001001 to 999366.
  - We need only about one-third as much storage if:
    - Location = day\_of\_the\_year || flight number
  - It is preferable to place the wider range last so that we do not have large gaps in the key values.
  - Since the airline probably would not have 999 flights, many potential numbers would still go unused.

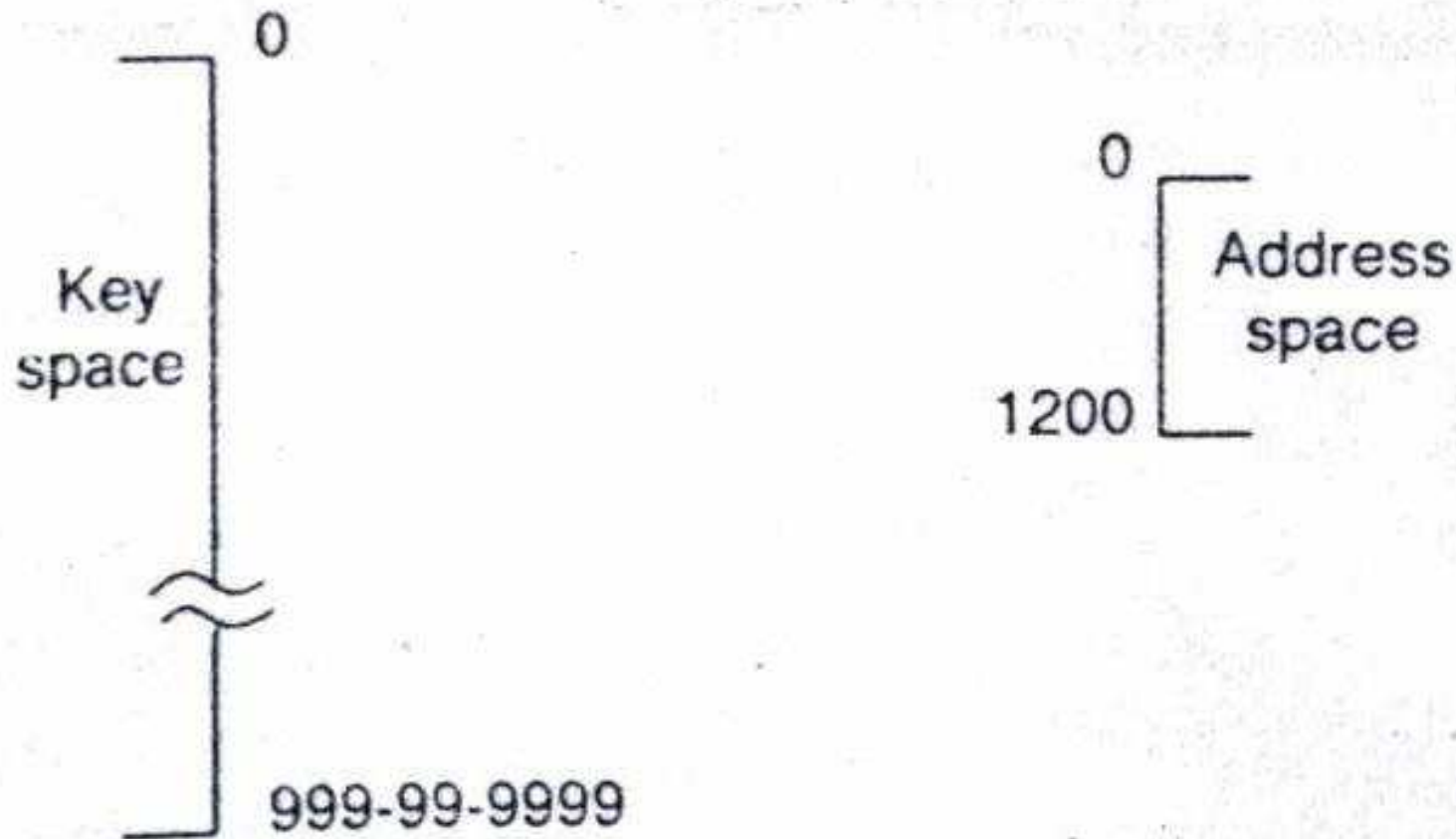
# The Key Converts to a Probable Address

- If we remove most of the empty spaces in the address space, we have a key space larger than the address space.
- We need a function for mapping. Such functions are referred to as **hashing functions**.

Hash(key)  $\rightarrow$  probable address

- This initial probable address is known as the **home address** for the record.

# The Key Converts to a Probable Address



Compressed address space.



# The Key Converts to a Probable Address

- A good function:
  - Evenly distributes the keys among the addresses
    - To reduce the number of collisions.
  - Executes efficiently
    - To keep the retrieval time to a minimum.
- A collision occurs when two distinct keys map to the same address. Hashing is composed of two aspects:
  - The function
  - The collision resolution method

# Hashing Functions

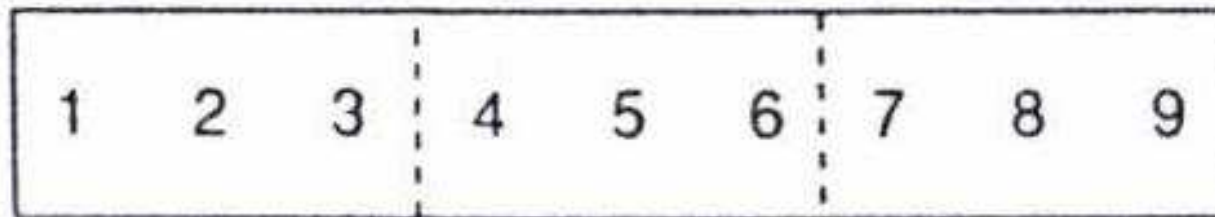
- **Key mod  $N$**  (where  $N$  is the table size).
- **Key mod  $P$**  (where  $P$  is the smallest prime number  $\geq N$ )
- **Truncation** (or substringing): If we have social security numbers as the primary keys such as 123-45-6789, the digits may come from *any* part of the key.

# Hashing Functions

- Truncation: Would it be reasonable to truncate the social security number key after the first three digits?
  - Depends on the application. If we are applying this function to employees who work at a single plant, it would not be a good choice. Numbers are assigned by geographic regions, most of the workers would have the same leftmost two or three digits.
  - A better choice would be to use the three rightmost digits.

# Hashing Functions

- **Folding:** Two kinds of folding exist:  
(1) Folding by boundary, (2) Folding by shifting
  - If we have the nine-digit key



# Hashing Functions

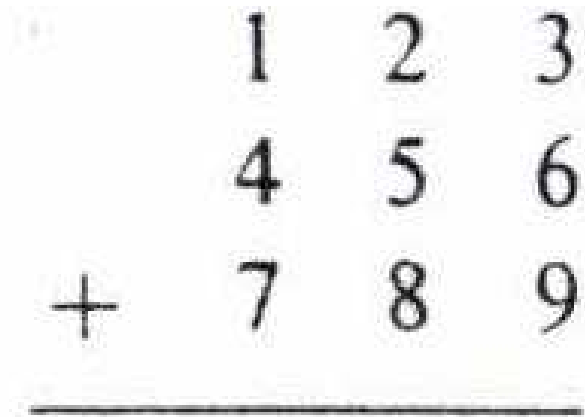
- Folding
  - Fold the number on the dashed boundaries into three parts. The digits would then be superimposed as

$$\begin{array}{r} 321 \\ 456 \\ + 987 \\ \hline \end{array}$$

Codes were added together (without carry) to obtain the probable address of 654.

# Hashing Functions

- Folding
  - Folding by shifting slides the parts one over another until all are superimposed.



The diagram illustrates the folding process by shifting. It shows three rows of numbers: 1 2 3, 4 5 6, and 7 8 9. A plus sign (+) is positioned to the left of the third row. A horizontal line is drawn under the third row, indicating the result of the folding operation.

Which would yield the different address of 258.

- We could also have added with carry. The results would be (1)764 and (1)368 respectively.

# Hashing Functions

- **Squaring** (a key and then substringing or truncating a portion of the result is another way).
- **Radix Conversion** (the key is considered to be in a base other than 10 and is then converted into a number in base 10. For example the key 1234 in base 11 would become 1610 in base 10. Substringing or truncation could then be used).

# Hashing Functions

- **Polynomial Hashing**

- The function

- $f(\text{information area}) \rightarrow$  cyclic check bytes

- is in fact a hashing function. The key is divided by a polynomial.

- **Alphabetic Keys**

- Alphabetic or alphanumeric key values can be input to a hashing function if the values are interpreted as integers. All information is represented internally in a computer as bits. Variant records or “unions” may be used.



# Collisions

- One hashing function may distribute the keys more evenly than another. One strategy for finding such a hashing function would be to combine simpler functions.
- A hashing function that has a large number of collisions is said to exhibit **primary clustering**.
- Secondary memory is slower. It is better to have a slightly more expensive hashing function if it reduces the number of collisions.

# Collisions

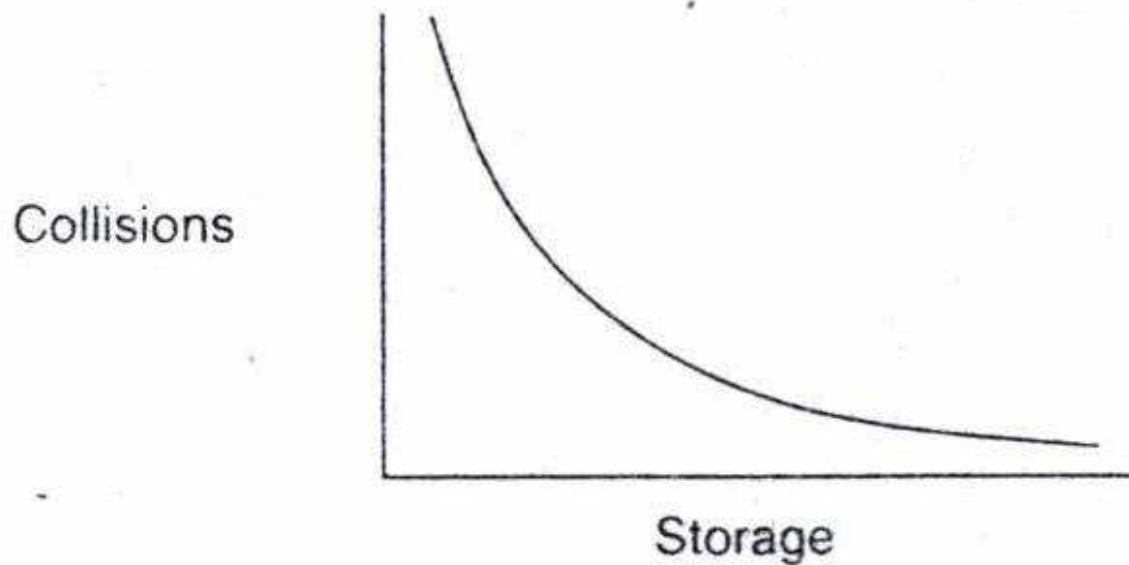
- One method for reducing collisions is to change hashing functions.
- Another method is to reduce the **packing factor**.

$$\text{Packing factor} = \frac{\text{number of records stored}}{\text{total number of storage locations}}$$

- Other names for it: packing density and load factor

# Collisions

- As the packing factor increases, the likelihood of a collision increases. The disadvantage of decreasing the packing factor to reduce the number of collisions is that a lower packing factor takes more space.



# Collision Resolution

- Changing the hashing function or decreasing the packing factor may reduce the number of collisions but not eliminate them.
- We need a procedure to position a synonym when a collision occurs. The method requires a minimum number of additional probes from its home address.

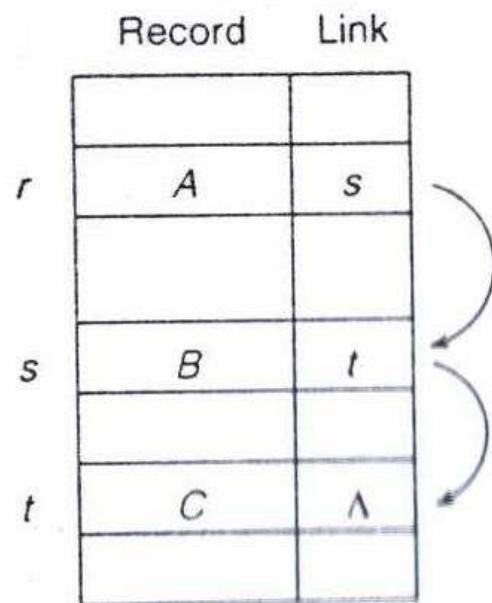
# Collision Resolution

- One way is to *point* to the location of the synonym record.
- We form a chain of synonym records. The NULL pointer indicates the end of the synonym chain.
- A disadvantage is that storage is needed for the link fields.

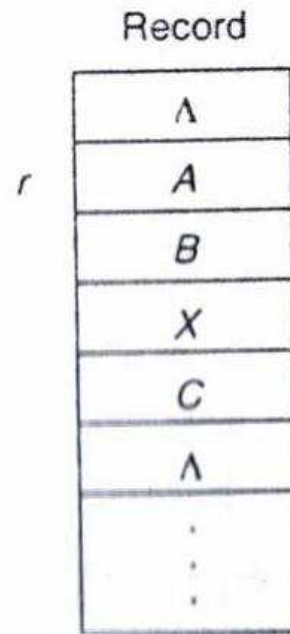
# Collision Resolution

- We need a mechanism that does not involve the use of actual links.
- We can use *implied* links by applying a convention, or set of rules. We compute the next search address by applying the set of rules.
- A simple convention is to look at the next location. If it is occupied, we then examine the following location.
- The disadvantage is that we may need to make more probes.

# Collision Resolution



(a)  
with links



(b)  
without links

Synonym chains: with and with-

out links.

# Collision Resolution

- There are several mechanisms for resolving collisions grouped according to the mechanism used to locate the next address to search and the attribute of whether a record once stored can be relocated;
  - Collision resolution with links
  - Collision resolution without links
    - Static positioning of records.
    - Dynamic positioning of records
  - Collision resolution with pseudolinks



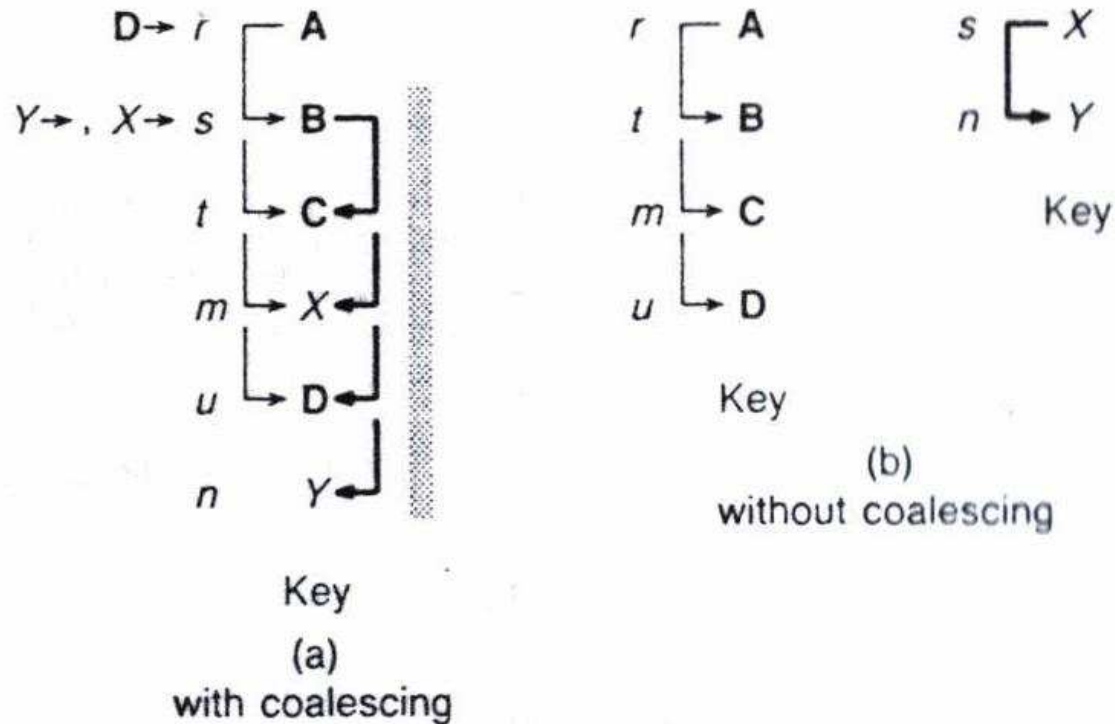
# Coalesced Hashing

- Uses pointers to connect the elements of a synonym chain.
- The insertion of a synonym is the same as the insertion into a linked list.
- On retrieval, the elements of the synonym chain are searched until the desired record is located or until the end indicated by NULL is reached.

# Coalesced Hashing

- “Coalescing” occurs when we attempt to insert a record with a home address that is already occupied by a record from a chain with a *different* home address.
- The two chains with records having different home addresses **coalesce** or grow together.
- Coalesced hashing is also referred to as *direct chaining*.

# Coalesced Hashing



Two synonym chains: with and without coa-

lescing.

A, B, C, D form one set of synonyms and X, Y form another set.

# Coalesced Hashing

- In (a), when we attempt to insert the record  $X$ , it collides with the record  $B$  which is *not* a synonym of  $X$  since  $B$  has a home address of  $r$  and  $X$  has a home address of  $s$ .
- The two chains coalesce where the bar on the right indicates the portion of the chain in which coalescing has occurred,  $|$  represents insertions on the synonym chain with  $r$  as its home address, and  $|$  represents those with  $s$  as its home address.
- This will result in more probes than would be required for noncoalesced chains.
- When  $X$  is inserted, it is inserted at the *end* of the chain. Instead of one, three probes are needed for  $X$ .

# Coalesced Hashing

- In (a), 19 probes in total is needed to retrieve each record once, in (b) where two synonym chains were separated, 13 probes.
- Insertion of a new record is made at the bottommost (highest address) empty location. An available space pointer is continually decremented until either an empty location is found or table is full.
- The available space pointer is initially set to a location one record length beyond the last storage location in the table.

# Coalesced Hashing

## Algorithm 3.1 COALESCED HASHING

---

- I. Hash the key of the record to be inserted to obtain the home address, or the probable address for storing the record.
  - II. If the home address is empty, insert the record at that location, else if the record is a duplicate, terminate with a “duplicate record” message, else
    - A. Locate the last record in the probe chain by following the link fields until encountering a  $\Lambda$  link that signifies the end of the chain.
    - B. Find the bottommost empty location in the table. If none is found, terminate with a “full table” message.
    - C. Insert the record into the identified empty location and set the link field of the record at the end of the chain to point to the location of the newly inserted record.
-

# Coalesced Hashing - Example

- For the collision resolution examples of this chapter we use

$$\text{Hash}(\text{key}) = \text{key} \bmod 11$$

Where 11 is the prime number table size.

- For comparison purposes, we use the same initial subset of data with the keys of  
27, 18, 29, 28, 39, 13, and 16

# Coalesced Hashing - Example

	Record	Link
0		$\Lambda$
1		$\Lambda$
2		$\Lambda$
3		$\Lambda$
4		$\Lambda$
5	27	$\Lambda$
6		$\Lambda$
7	18	10
8		$\Lambda$
9		$\Lambda$
$R \rightarrow 10$	29	$\Lambda$

- When we insert 29, there is a collision.
- The available space pointer, R is decremented from the bottom to check if location 10 is available.
- We set the link field in location 7 to point to location 10.



# Coalesced Hashing - Example

	Record	Link		Record	Link
0		$\Lambda$	0		$\Lambda$
1		$\Lambda$	1		$\Lambda$
2		$\Lambda$	2	13	$\Lambda$
3		$\Lambda$	3		$\Lambda$
4		$\Lambda$	4		$\Lambda$
5	27	$\Lambda$	5	27	8
6	28	9	6	28	9
7	18	10	7	18	10
8		$\Lambda$	8	16	$\Lambda$
$R \rightarrow$ 9	39	$\Lambda$	9	39	$\Lambda$
10	29	$\Lambda$	10	29	$\Lambda$

# Coalesced Hashing - Example

	Record	Link
0		Λ
1		Λ
2	13	Λ
<i>R</i> → 3	17/6	Λ
4	42/7	3
5	27	8
6	28/6	9
7	18	10
8	16	Λ
9	39/6	4
10	29	Λ

ing.

- If we add two additional records with keys of 42 and 17 and home addresses of 9 and 6 respectively, we will have coalescing of the chains with these two home addresses.
- If we add records to both of the probe chains the performance of each chain would be degraded.

Insertion using coalesced hash-

# Coalesced Hashing - Discussion

- One method of reducing the coalescing is to reduce the packing factor. The lower it is, the less chance that there will be a collision from one chain with that of another.
- Also there are variants of coalesced hashing to reduce coalescing.
- An unsuccessful search takes more time since we need to examine more records.

# Coalesced Hashing - Discussion

- The number of retrieval probes is a function of the *order of insertion*. A record inserted early in the process will be placed near the beginning.
- So if the frequencies are known, it is good to place the most frequently accessed records early in the insertion process.

# Coalesced Hashing - Deletion

- We cannot merely remove the record as if we were deleting an element from a singly linked list.
- We might lose the remainder of a probe chain that had coalesced as a result of being its home address.

# Coalesced Hashing - Deletion

- A simple deletion procedure is to move a record later in the chain into the position of the deleted record. The relocated record should be the last element on the chain having the home address of the location of the deleted record.
- This relocation process is repeated at the vacated location until no other records need to be moved.
- Then, prior-to-moving predecessor of the most recently moved record is relinked to the moved record's prior-to-moving successor.

# Coalesced Hashing - Deletion

- A more complex scheme could be used to reposition records in the coalesced chains to reduce the amount of coalescing.
- Determining if coalescing has occurred in a probe chain may be time-consuming.
- If we know that a coalescing has not occurred, we just remove the deleted record from a singly linked list.
- Deleted record is reinitialized to null, the available space pointer,  $R$ , is reset to the maximum of (1) its current address and (2) the vacated address plus one.

# Coalesced Hashing – Deletion Example

	Record	Link
0		$\Lambda$
1		$\Lambda$
2	13	$\Lambda$
<i>R</i> → 3	17/6	$\Lambda$
4	42/7	3
5	27	8
6	28/6	9
7	18	10
8	16	$\Lambda$
9	39/6	4
10	29	$\Lambda$

To delete the record with key 39 requires that

- Because coalescing has occurred, we move the last element in the chain with a home address of 9 (record 42) into location 9 and adjust the links and table entries.

- *R* is set to location 5 (the vacated address plus one).

Insertion using coalesced hash-

ing.



# Coalesced Hashing – Deletion Example

	Record	Link
0		$\Lambda$
1		$\Lambda$
2	13	$\Lambda$
3	17	$\Lambda$
4		$\Lambda$
$R \rightarrow$ 5	27	8
6	28	9
7	18	10
8	16	$\Lambda$
9	42	3
10	29	$\Lambda$

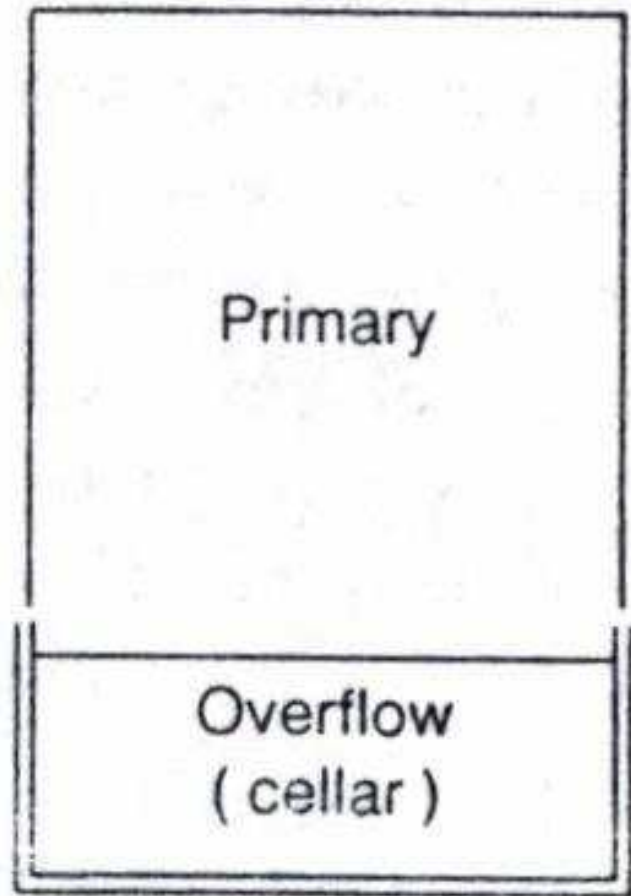
This is the resulting table.

# Coalesced Hashing – Variants

- To reduce coalescing, the variants may be classified in three ways:
  - The table organization (whether or not a separate overflow area is used)
  - The manner of linking a colliding item into a chain.
  - The manner of choosing onoccupied locations.

# Coalesced Hashing – Variants

- Table organization: Table is divided as follows:



primary area

$$\text{Address factor} = \frac{\text{primary area}}{\text{total table size}}$$

As the address factor decreases, the cellar size increases, reduces coalescing but decreases primary area increasing the collisions.

An address factor of 0.86 yields nearly optimal performance for most load factors.

# Coalesced Hashing – Variants

- The previous algorithm 3.1 is called late insertion standard coalesced hashing (LISCH) since new records are inserted at the *end* of the chain.
- The variant using a cellar is called LICH, late insertion coalesced hashing.

# Coalesced Hashing – LICH

- If we insert 27, 18, 29, 28, 39, 13, 16, 42, 17

	Record	Link
0	28	8
1	29	Λ
2	16	Λ
3	17	Λ
4	18	10
5		Λ
6	27	9
7		Λ
<i>R</i> → 8	42	Λ
9	13	Λ
10	39	Λ

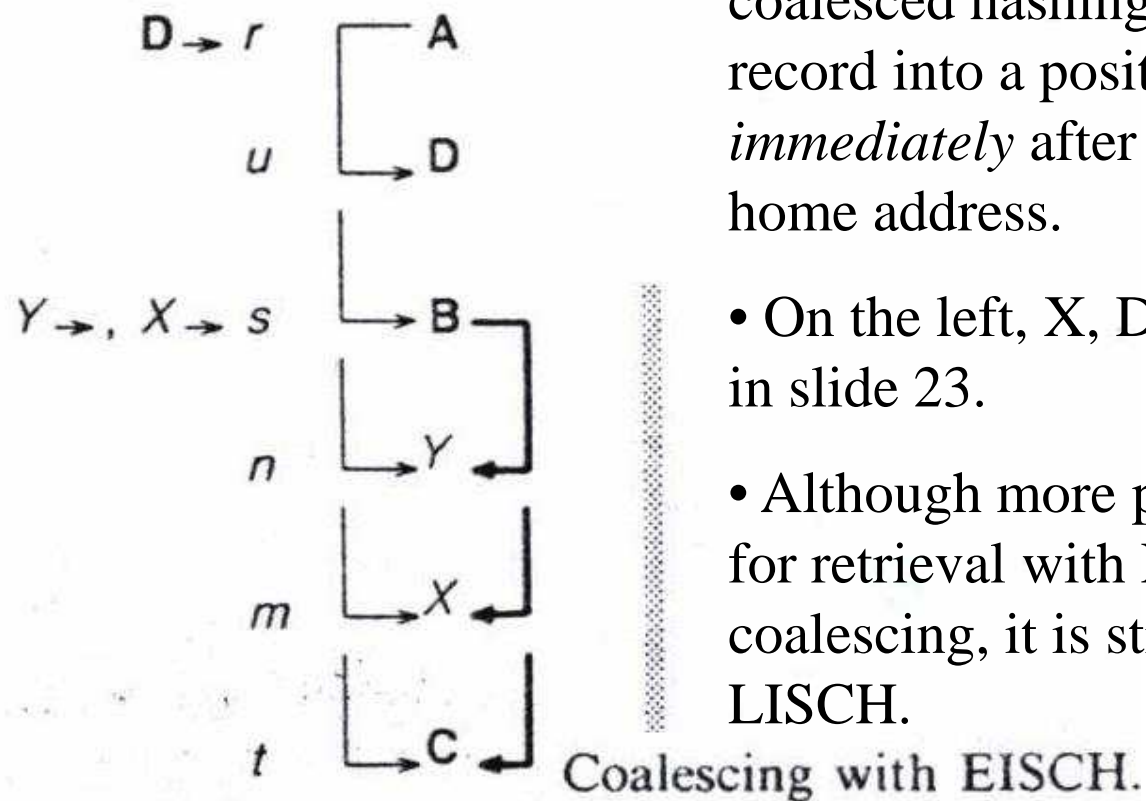
Cellar

7 primary locations and 4 cellar locations using key mod 7.

The average number of probes needed to retrieve each record in this table once is 1.3 versus 1.8 of previous (LISCH) example.

# Coalesced Hashing – Variants

- Second technique is to vary the position in which new records are inserted.



- A variant called early insertion standard coalesced hashing (EISCH) inserts a new record into a position on the chain *immediately* after the record stored at its home address.

- On the left, X, D, Y are inserted to the table in slide 23.

- Although more probes are still necessary for retrieval with EISCH than with no coalescing, it is still an improvement over LISCH.

# Coalesced Hashing – EISCH Example

	Record	Link
0		Λ
1		Λ
2	13	Λ
3		Λ
<i>R</i> ⇒ 4	42	Λ
5	27	8
6	28	9
7	18	10
8	16	Λ
9	39	4
10	29	Λ

- Here, until a chain is greater than 2 elements, LISCH and EISCH give the same result.
- We begin with the file appearing on the left.

# Coalesced Hashing – EISCH Example

	Record	Link
0		$\Lambda$
1		$\Lambda$
2	13	$\Lambda$
$R \rightarrow$ 3	17	9
4	42	$\Lambda$
5	27	8
6	28	3
7	18	10
8	16	$\Lambda$
9	39	4
10	29	$\Lambda$

- When we insert 17, instead of inserting it at the end of the chain (after 42), it is inserted in the position after the record at the home address( after 28)
- Early insertion reduces the amount of coalescing. The earlier insertion postpones the coalescing to a later position.
- The algorithm using early insertion with a cellar is called EICH.



# Coalesced Hashing – Variants

- The third technique is to change the way in which we choose an unoccupied location. As specified in the algorithm, the unoccupied locations are always chosen from the *bottom* of the storage area.
- By concatenating *all* the overflow items in one area, we *increase* the number of collisions. We can choose a *random* unoccupied location for the new insertion. So, overflow records would be more evenly distributed.
- But, for a %90 packing factor, REISCH (R = Random) gives only %1 improvement over EISCH

# Coalesced Hashing – Variants

- Another method is to alternate the selection between the top and bottom of the table, called BLISCH (B = Bidirectional).
- A means of eliminating coalescing is to *move* a record not stored at its home address. In moving, we would not increase the number of probes for the item moved since we are merely inserting it into a new location but it retains its same position on the probe chain. This variation of LISCH is called direct chaining without coalescing (DCWC).

# Coalesced Hashing

**TABLE 3.1** MEAN NUMBER OF PROBES FOR SUCCESSFUL LOOKUP ( $n = 997$ ) FOR VARIANTS OF COALESCED HASHING

$\alpha$ method	0.2	0.4	0.6	0.8	0.9	0.95	0.99
EISCH	1.1065	1.2277	1.3684	1.5290	1.6182	1.6653	1.7033
LISCH	1.1063	1.2316	1.3789	1.5657	1.6737	1.7337	1.7827
BEISCH	1.1055	1.2286	1.3721	1.5336	1.6236	1.6728	1.7107
BLISCH	1.1055	1.2341	1.3836	1.5703	1.6818	1.7423	1.7898
REISCH	1.1063	1.2322	1.3693	1.5257	1.6124	1.6614	1.7014
RLISCH	1.1085	1.2384	1.3876	1.5653	1.6723	1.7296	1.7790
EICH	1.1116	1.2256	1.3408	1.4942	1.5867	1.6347	1.6762
LICH	1.1116	1.2256	1.3406	1.4888	1.5801	1.6281	1.6695

*Source:* Hsiao, Yeong-Shiou, and Alan L. Tharp, "Analysis of Other New Variants of Coalesced Hashing," Technical Report TR-87-2, Computer Science Department, North Carolina State University, 1987.