Lecture 10: Direct Files

Last Day: Direct Files

- Hashing
- Collisions

Today: Hashing:

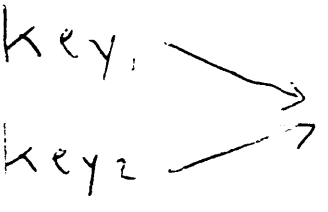
- Collision Resolution
- Open Addressing / Clustering
- Linear / Non-Linear Probing
- Double Hashing
- Buckets

Folk & Zoellick § 10.5 - 10.8

Hashing

- Given the key for a record, compute the record's address.

- When two records hash to the same address, we say they collide.



- The hash value (address) is called their home address, but one of them must be stored elsewhere.

- Finding this other storage location is called collision resolution

Collision Resolution

There are two basic approaches:

(1) Open Addressing (Progressive Overflow)

ie, Store the record at some other address in the same file.

(2) Separate Overflow.

ie, Store the record in another file, called the overflow area.

Open Addressing

- For each key, generate a sequence of addresses, called the probe sequence:

  $PA_0, PA_1, PA_2, PA_3, \cdots$

  When a collision occurs, store the new record at the first available probe address, ie, at the first PA_i that is not already storing a record.

- Note: PA_0 = home address = hash(key)

Linear Probing

A special case of open addressing in which

$$PA_i = [hash(key) + i \times step] \bmod N$$

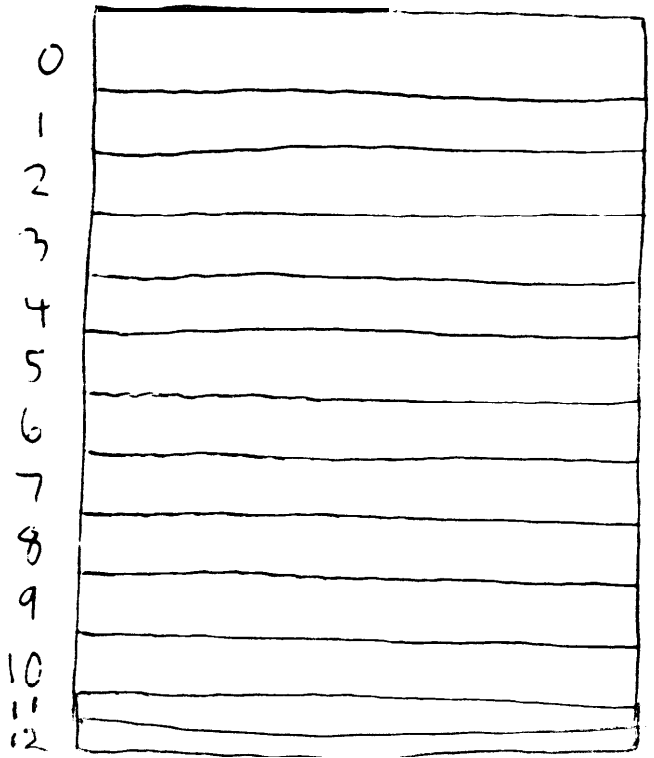
where N = file size (max. # records)
and step is a constant.

Note: $PA_{i+1} = (PA_i + step) \bmod N$

Example

key	hash(key)	PA_1	PA_2	PA_3	PA_4
Mozart	1	2	3	4	5
Tchaikovsky	8	9	10	11	12
Ravel	10	11	12	0	1
Beethoveen	5	6	7	8	9
Mendelssohn	5	6	7	8	9
Bach	10	11	12	0	1
Greig	3	4	5	6	7
Rachmaninoff	5	6	7	8	9
Vivaldi	6	7	8	9	10
Chopin	6	7	8	9	10

hash File



$N = 13$

$Step = 1$

$\therefore PA_i =$

$[i + hash(key)] \bmod 13$

Sample File accesses

- search for Ravel

  Cost = 1 File access

- Search for Chopin

  Cost = 7 File accesses.

- Search for Eisner (home address = 0)

  Stop searching when an empty address is reached (ie, at address 0)

  Cost = 4 file accesses

- Insert Eisner

  (home address = 10, stored at 0)

Deletions

Examples:

- Delete Chopin

- Search for Eisner (home address = 10).

  Note: Search stops at address 12, which is an empty record.

  But, it should not stop here!

Problem: Deletions can cause searches to end too soon.

Solution: When a record is deleted, we mark the address with a "tombstone."

When searching for records we pass over tombstones without stopping.

Examples:

- Delete Bach

  (Note: there are now tombstones at addresses 11 and 12.)

- Search for Eisner (home address = 10)

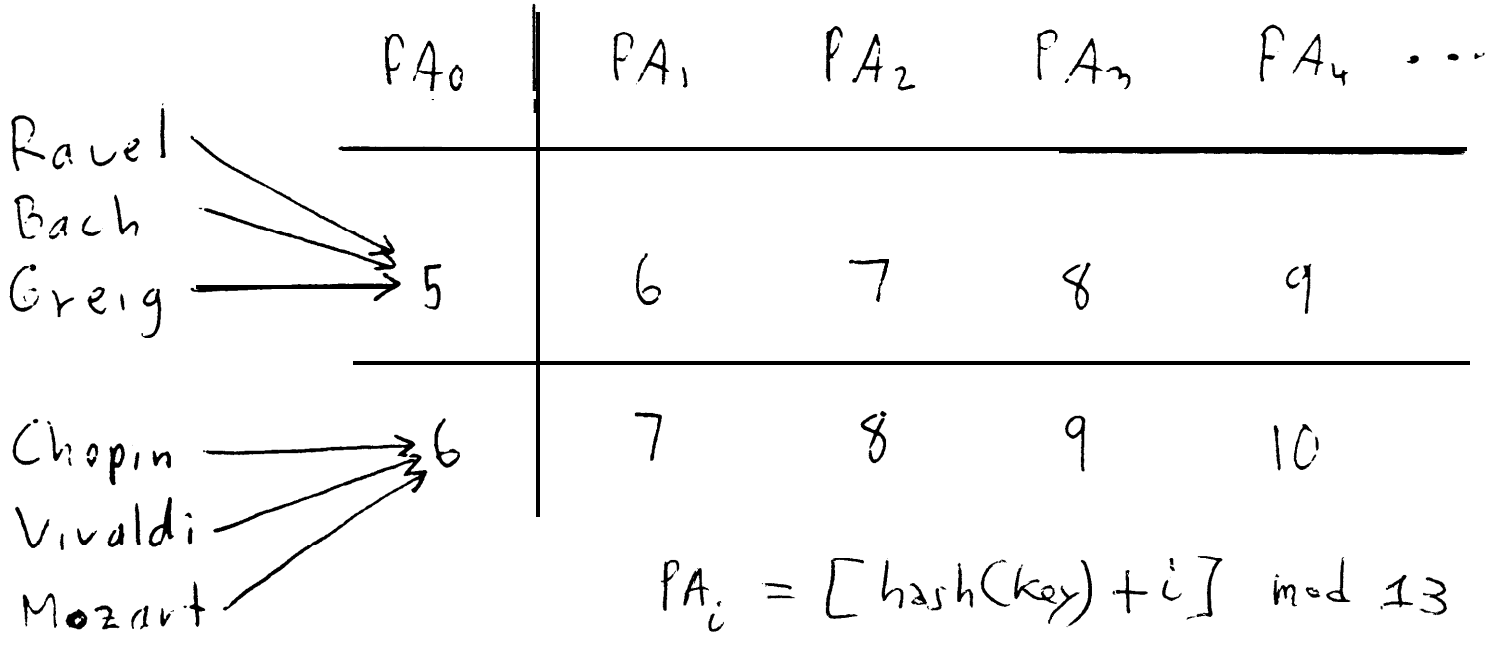
  Cost = 4 file accesses

  (Note: Without tombstones, we could not delete Eisner.)

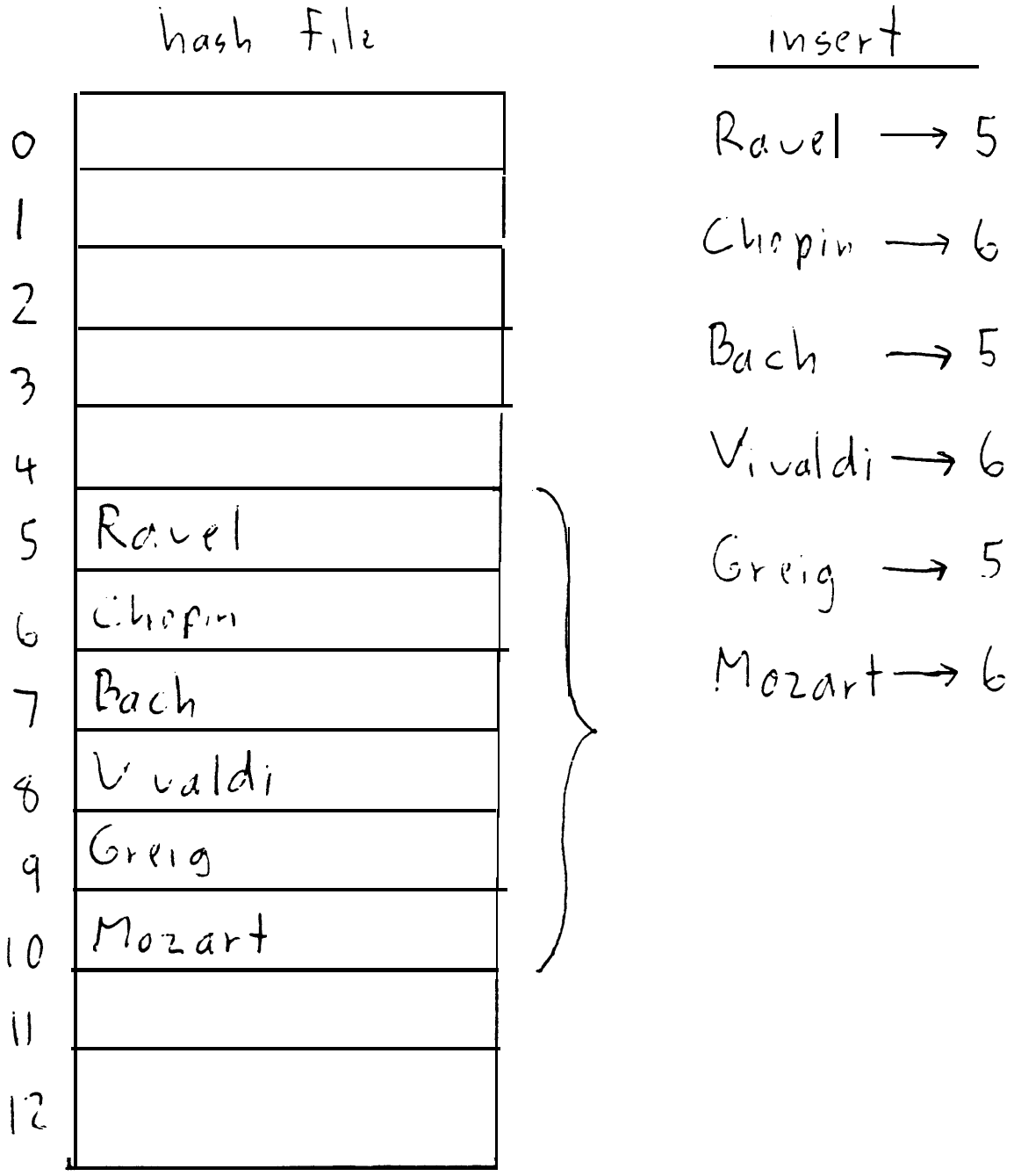
Clustering

- If many keys hash to the same vicinity, a dense cluster of records can form.
- This increases search time, since many probes may be needed to get through the cluster.

Example



$PA_i = [\text{hash}(\text{key}) + i] \bmod 13$



insert

Ravel → 5

Chopin → 6

Bach → 5

Vivaldi → 6

Greig → 5

Mozart → 6

A primary cluster

Examples: Access Cost

- Retrieving Bach or Vivaldi now takes 3 probes (ie, file accesses)

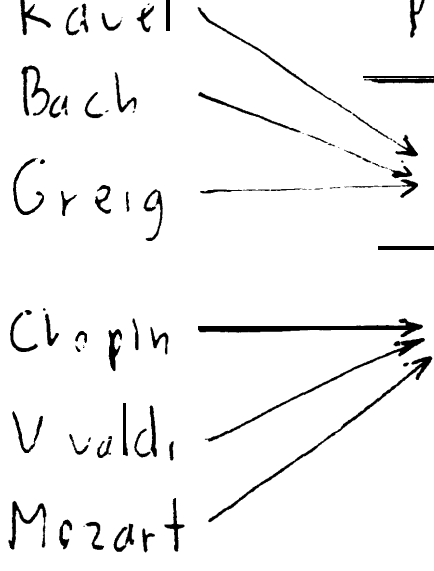
- Retrieving Greig or Mozart now takes 5 probes.

These accesses slow down retrieval (and updating) significantly.

A Partial Solution

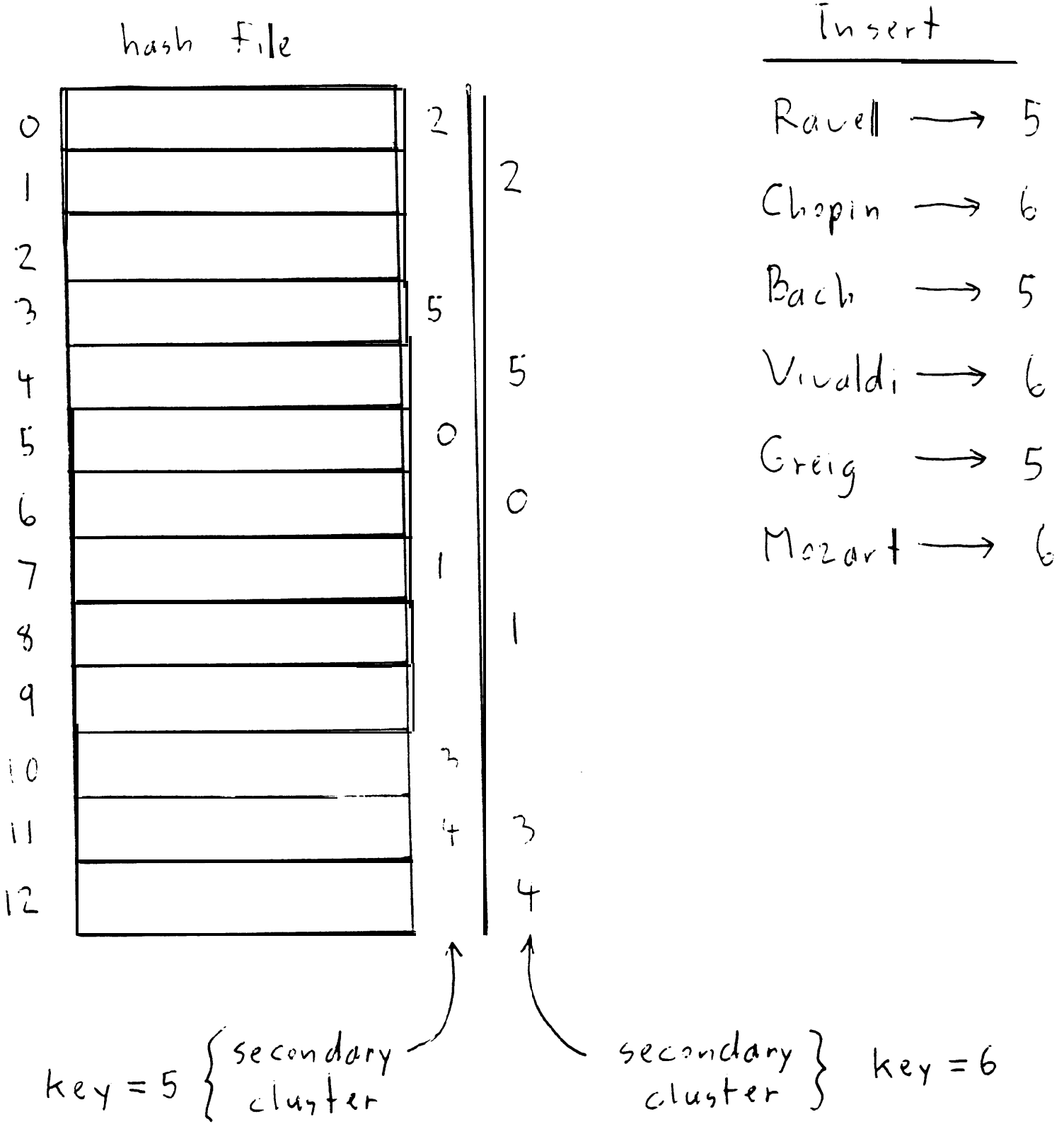
Use a non-linear probing function

eg. $PA_i = [\text{hash}(\text{key}) + 2i^2] \bmod 13$



	PA_0	PA_1	PA_2	PA_3	PA_4	PA_5
Ravel, Bach, Greig	5	7	0	10	11	3
Chopin, Vivaldi, Mozart	6	8	1	11	12	4

i	1	2	3	4	5
$2i^2$	2	8	18	32	50
$5+2i^2$	7	13	23	37	55
$6+2i^2$	8	14	24	38	56



The primary cluster has been broken up into two secondary clusters

Secondary Clustering

- All records with the same home address follow the same sequence of probe addresses.
- This sequence is a secondary cluster.
- Solution: Double Hashing

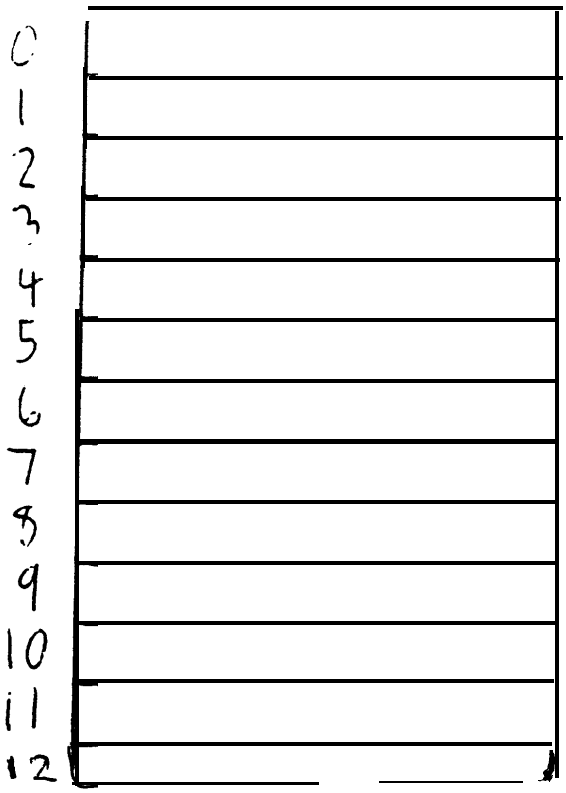
$$PA_i = [hash_1(key) + i \times hash_2(key)] \bmod N$$

↑ variable step

- Each probe sequence is linear, but the step size now depends on the key.

Example

key	$hash_1$	$hash_2$	PA_1	PA_2	PA_3	PA_4	PA_5
Ravel	5	3	8	11	1	4	7
Chopin	6	4	10	1	5	9	0
Bach	5	7	12	6	0	7	1
Vivaldi	6	8	1	9	4	12	7
Greig	5	5	10	2	7	12	4
Mozart	6	10	3	0	10	7	4
	↑	↑					
	home address	step size					



$$PA_i = (hash_1 + i \times hash_2) \bmod 13$$

Double Hashing: Observations

- Records are now scattered throughout the file.
- No clusters (primary or secondary).
- Records can now be accessed with fewer probes.
- ie, More records are stored at or near their home addresses.
- Thus, retrieval is now <u>faster.</u>

Buckets

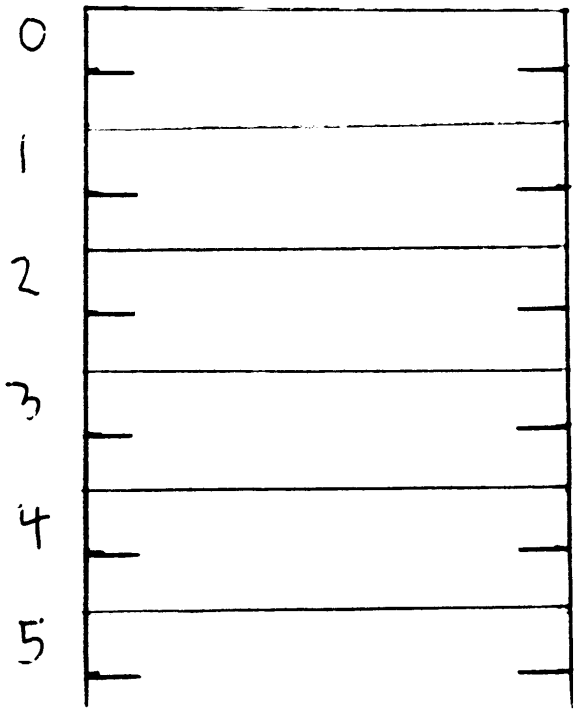
- Recall that a disk reads and writes not one record at a time, but an entire block of data at a time, which may hold <u>many</u> records.
- ∴ It makes sense for hash functions to compute <u>block</u> addresses, not record addresses
- This way, several records can be stored at the same address
- Each such block is called a <u>bucket</u>.

- Now addresses refer to buckets, not records.
- We say that a hash function identifies a bucket, into which the record is placed.
- Note: Collisions are not a problem unless the bucket is full
- Such collisions are said to cause an overflow in the bucket.

Example

key	hash(key)
Mozart	1
Tchaikovsky	2
Ravel	4
Beethoven	5
Mendelssohn	5
Bach	4
Greig	3
Rachmaninoff	5
Vivaldi	0
Chopin	0

- Linear probing
- Step size = 1
- Bucket size = 2



Note:

- Fewer overflows
- Records stored much closer to home
- ∴ Faster retrieval

Hashing so far:

- Open addressing (Progressive overflow)
- All records are stored in one file.

Paradox (Tradeoff) of Open Addressing:

- Clustering increases access time.
- To avoid clustering, records with the same home address must be scattered throughout the file.
- But, this leads to more disk-head movement, which also increases access time.

Alternative: Separate Overflow.