

# Static Hashing — Definition (Notes)

**Static hashing** is a type of hashing where:

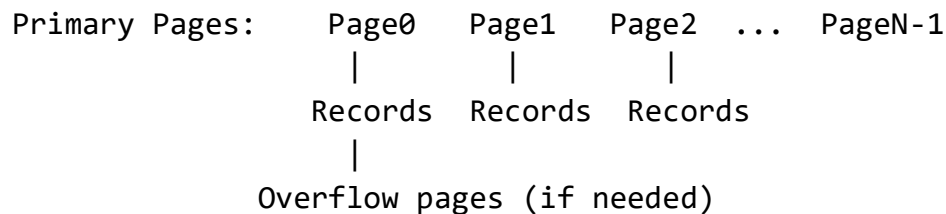
- The **hash function always generates the same bucket address** for a given key.
- The **number of buckets (or pages) in the hash file remains fixed** once chosen — it doesn't grow or shrink as records are inserted or deleted.

Example: If you use  $h(k) = k \bmod 5$ , then all keys will map only to buckets 0–4.

## Hash File Organization (Static)

- **Bucket/page:** A storage unit on disk or in memory where one or more records can be kept.
- In static hashing, the number of **primary bucket pages** is fixed.
- **Overflow pages** may be used if buckets get full (due to collisions).

**Diagram-style** (conceptual):



In this model, even if overflow pages are added, the number of *primary pages* stays fixed.

# Operations (Static Hashing)

## 1. Insertion

- a. Compute bucket address using hash function  $h(K)$ .
- b. Store the record in that bucket or in an *overflow page* if bucket full.

## 2. Search

- a. Apply the **same hash function** to compute address and check that bucket (and any overflow).

## 3. Deletion

- a. Find the record via hash, then delete it.

## 4. Update

- a. Locate record via hash and modify it.

# Properties (Static Hashing Notes)

## ✓ Fixed number of buckets/pages

- The hash table size defined at creation stays unchanged.

## ✓ Deterministic address computation

- Same key always maps to the same bucket.

## ✓ Hash function based on key

- Typically simple (e.g., modulo).

## ✓ Handles data in pages/buckets

- Records are placed into pages based on hashed addresses.

# Collision Handling

Since multiple keys can map to the same bucket:

**Two main strategies in static hashing:**

1. **Overflow chaining (closed hashing)**
  - a. When a bucket is full, **overflow pages** are chained to it.
2. **Open hashing (probing)**
  - a. On collision, a probing scheme (e.g., linear probing) finds another free slot.

## Advantages (Short Notes)

- Efficient for **small or stable datasets**.
- Fast direct access when the hash function distributes well.

## Disadvantages (Short Notes)

- **Doesn't scale well** if data grows beyond initial capacity; lots of overflow pages can degrade performance.
- Fixed number of buckets can waste space if database shrinks.
- Not suited to dynamic datasets — dynamic hashing (e.g., extendible/linear hashing) addresses this.

**Algorithm:** Static Hashing with Linear Probing

Step 1: Start

Step 2: Choose a hash function  $h(k)$  such that  $h(k) = k \bmod m$ , where  $m$  is the table size.

Step 3: Compute the hash address:

$\text{index} = h(k)$ .

Step 4: If the hash table at index is empty,

Insert the key at that location.

Step 5: If the hash table at index is already occupied (collision occurs),  
apply linear probing.

Step 6: Check the next index using:

$(\text{index} + 1) \bmod m$ .

Step 7: Repeat probing until an empty slot is found.

Step 8: Insert the key into the empty slot.

Step 9: For searching, compute  $h(k)$  and check sequentially until the key is found or an slot is encountered.

Step 10: Stop.