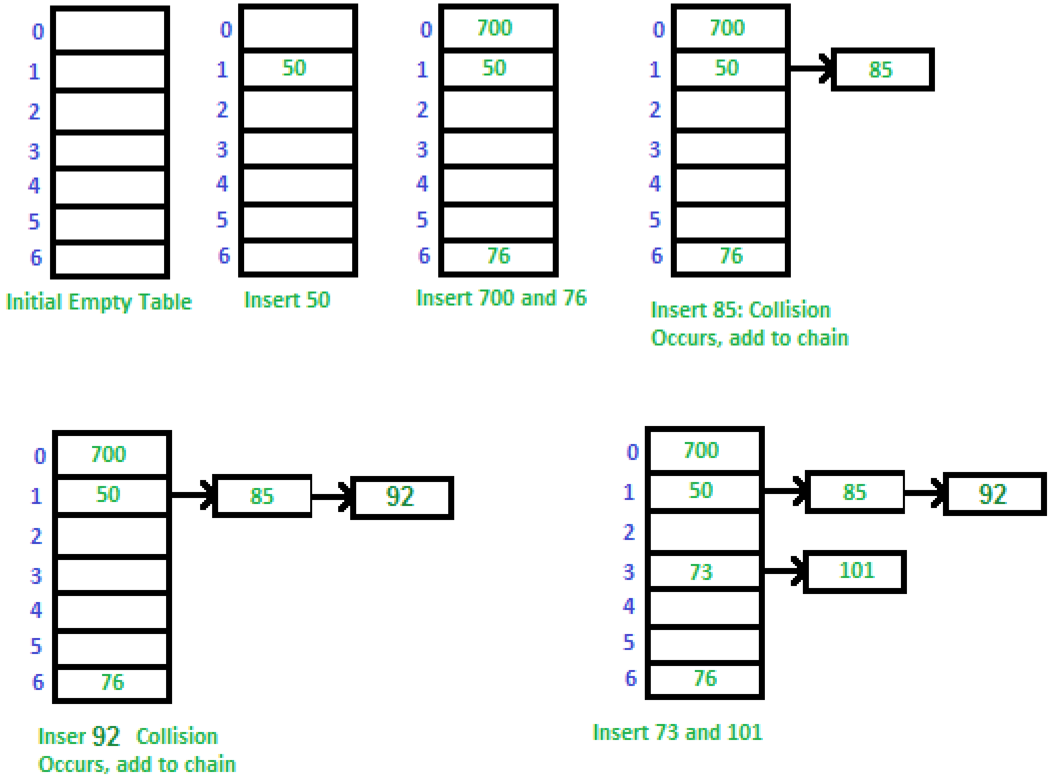
Dynamic Hashing Schemes

David Bui

Static Hashing

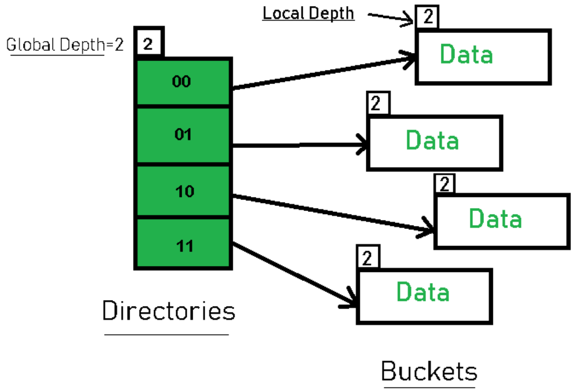


Separate Chaining

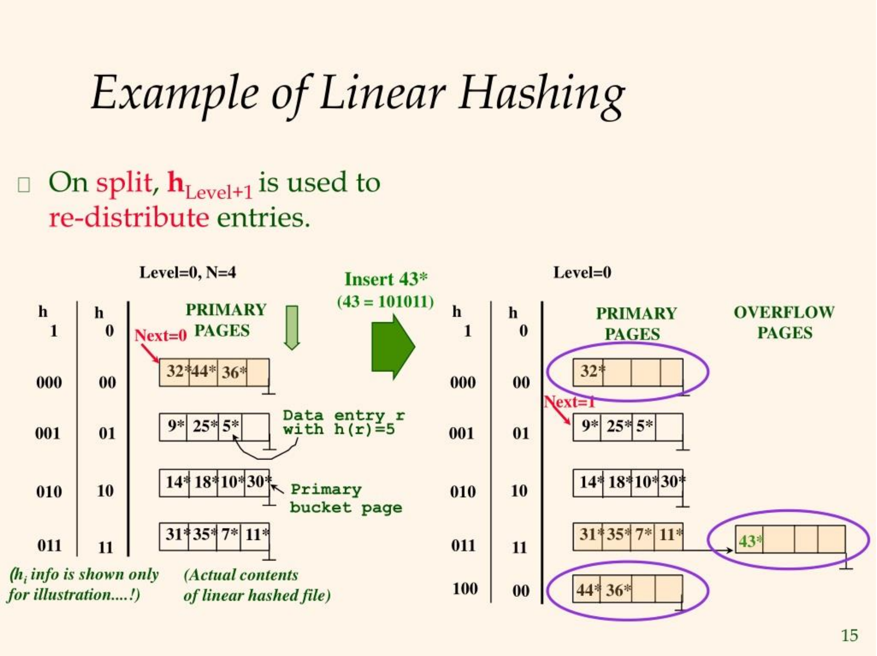
- Storage space allocated statically
- Issue 1: If file size exceeds allocated space the entire file needs to be moved to a larger space and rehashed
- Issue 2: Eventually access time will grow from O(1) to O(n) due to overflow.
- Static hash techniques: Linear Probing, Coalesced Chaining, Separate Chaining

Dynamic Hashing

- Hashing schemes that expand and contract when needed.
- Require hash functions to generate more key bits as file expands and less key bits as file shrinks.
- There are two types of dynamic hashing schemes those with directory schemes and directoryless schemes



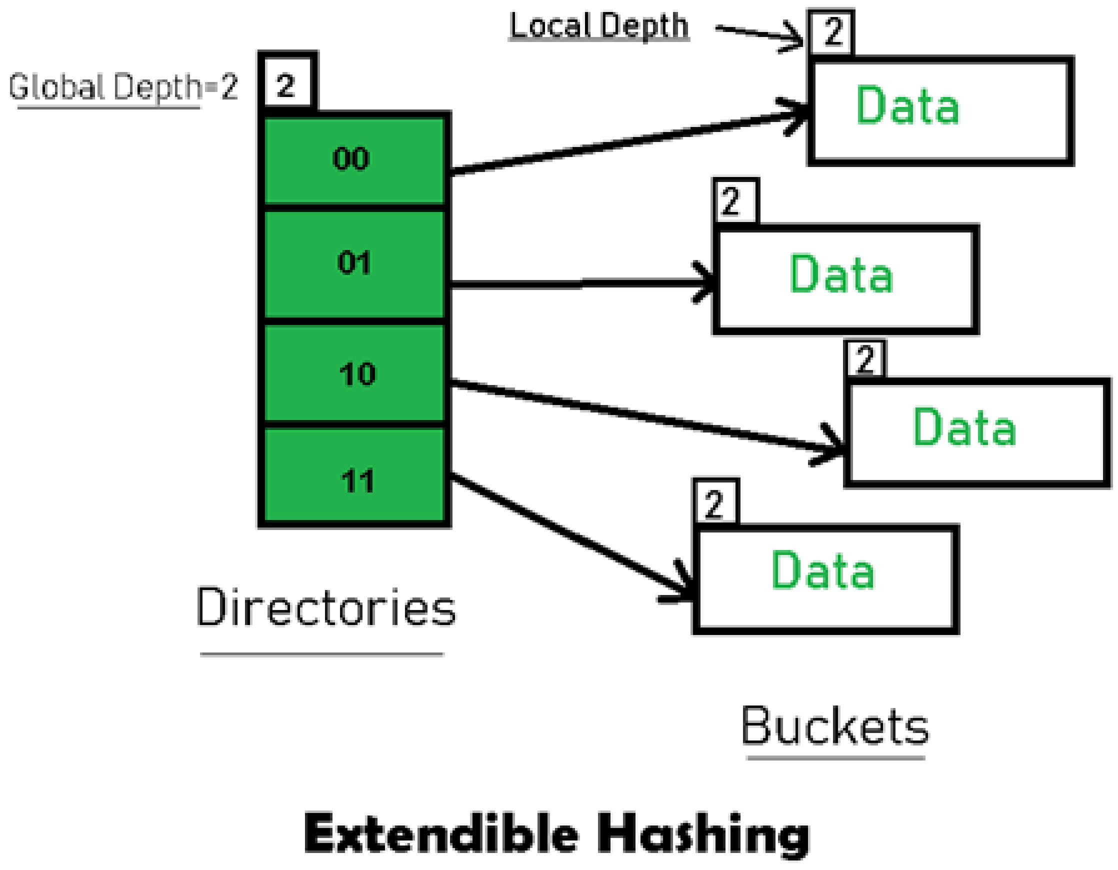
Extendible Hashing



Example of Linear Hashing

□ On split, $\mathbf{h}_{\text{Level+1}}$ is used to re-distribute entries.

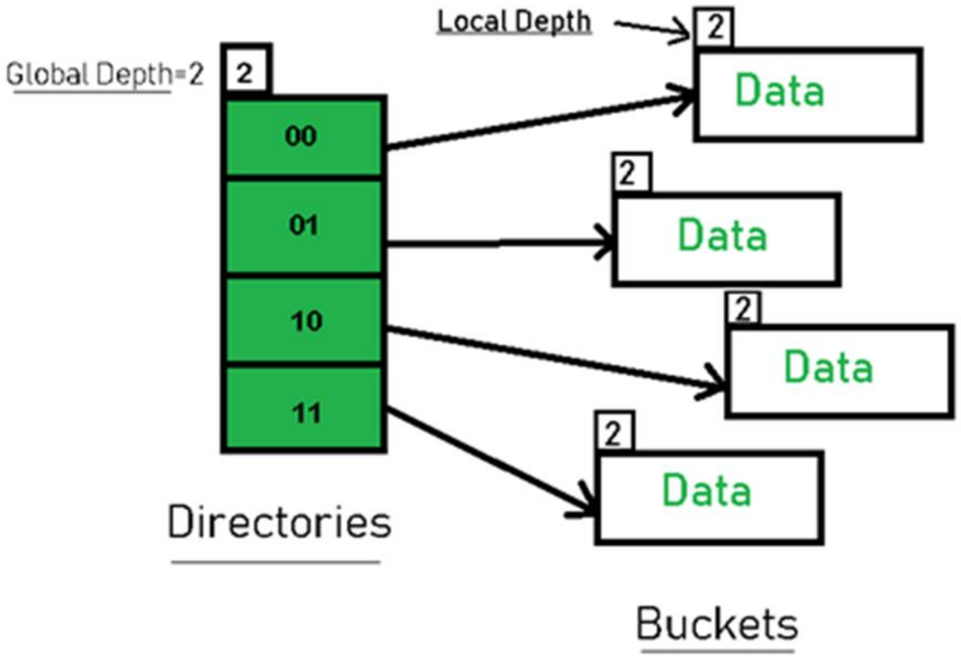
Extendible Hashing



- The dynamic hashing technique that uses directories.
- Directories store bucket addresses in pointers. Each directory has a dynamically changing id.
- Global Depth: Number of bits in directory id
- Local Depth: Number of bits in bucket id. Local Depth is always <= Global Depth

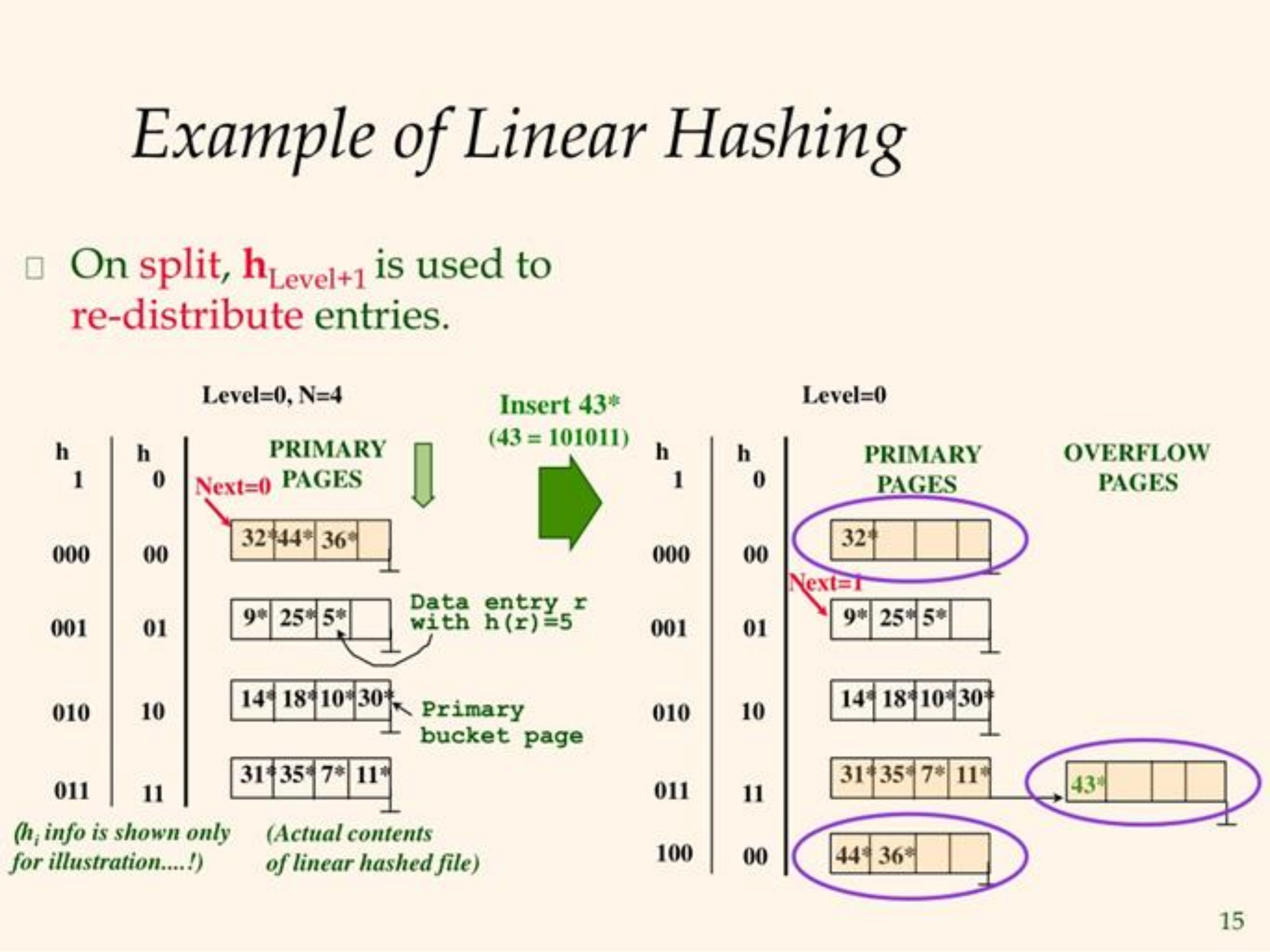
Extendible hashing steps

1. Hash the data
2. Match “global depth” number lower significant bits of the hashed data to the corresponding directory id
3. Go to bucket pointed by directory and insert if there is no overflow.
4. If bucket overflows and local depth = global depth, expand directory, split bucket, and then increment local and global depth number.
5. If bucket overflows and and local depth < global depth just split the bucket and increment local depth by 1
6. All split buckets must be rehashed



Extendible Hashing

Linear Hashing



Example of Linear Hashing

- On split, $h_{Level+1}$ is used to re-distribute entries.

(h_i info is shown only for illustration....!)

(Actual contents of linear hashed file)

15

- The dynamic hashing technique that uses no directories.
- Instead, keys are hashed directly to a bucket.

Linear Hashing Terms

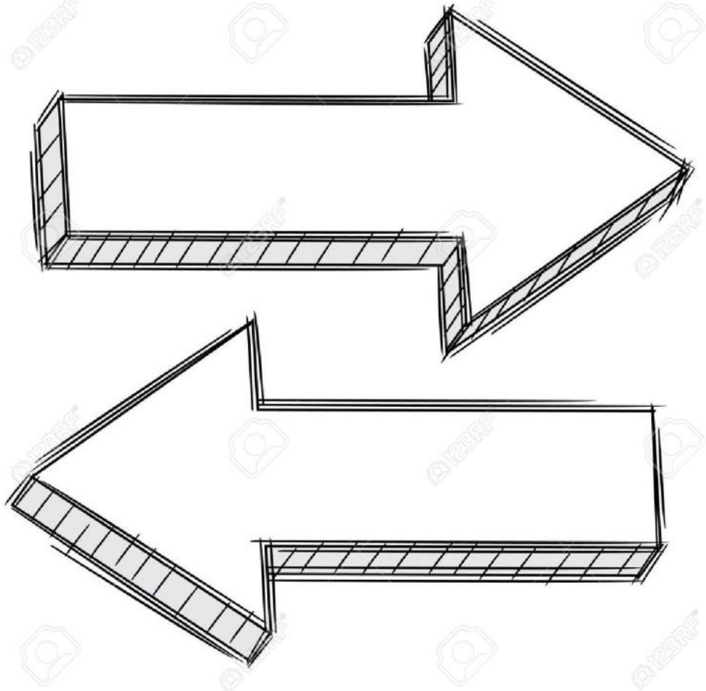
- N = number of buckets (initial number always a power of 2)
- S = index of bucket to be split
- I= number of bits needed to address N BUCKETS
- Load factor = number used a threshold to determine if we expand or contract the table

Linear Hashing Steps

- A hash function will give typically give some number of bits. Let’s say our hash function gives 32-bit output from some key. However, in Linear Hashing we will only use the first I bits since we only start with N buckets.
- If we start with N= 2 buckets, then I = 1 bits. So, we will only use the first bit of the hash function’s 32-bit output to map to a bucket.
- Once number of insertions exceed the load factor add 1 bucket to N. If N >(2^I -1) we need to increment I to address to the new bucket.
- When any bucket is added we split the bucket at index S’s keys with the new bucket, rehash if I is incremented, and then increment S. Once N has doubled from where it was initially, we reset S to 0.

Comparison

- Extendible Hashing
- Advantages
  1. Since Buckets are a fixed size in directory hashing schemes possible to set a upper bound access times.
- Disadvantages
  1. Wasted memory when global and local depth difference becomes large.
  2. Directory can become unbalanced due to too many hashed records.



- Linear Hashing
- Advantages
  1. Lower on average access time due to no directory.
  2. Partial expansion more graceful than doubling directory
- Disadvantages
  1. Unable to set an upper bound like directory schemes
  2. Physical implementation performance is unclear.

Reference

- Enbody, R. J., & Du, H. C. (1988). Dynamic hashing schemes. ACM Computing Surveys, 20(2), 850-113