CS5800: External Hashing

Ricardo Baeza-Yates & Anurag Bhardwaj

Northeastern University 2018

Agenda

- External Hashing
  - Introduction
  - Collision Avoidance
  - Bucketed Hashing
  - Static vs Dynamic Hashing
- Dynamic Hashing
- Linear Hashing
- Extendible Hashing

Introduction

- External Hashing: Hashing for disk files
- Key Ideas:
  - Target Address Space consists of buckets
  - Each bucket is either a single or multiple contiguous blocks
  - Hash function maps a key to a bucket number which points to an absolute block address on disk
- External Hashing Vs. B+-Trees:
  - Faster than B+-Trees for searches
  - Cannot support range queries

Collision Avoidance

- Collision is not a big problem for external hashing
  - Each bucket can hold as many records as possible
  - When bucket is full, chaining can be used
  - Record pointers chain the bucket to an Overflow Bucket
  - Hashing performance depends on number of overflow buckets

Bucketed Hashing

- Hash Function H generates a bucket address
- Store more than one key at the bucket address
- N Hash addresses split into B buckets
- Each bucket gets N/B slots
- Overflow record handling

Bucketed Hashing

- Insertion:
  - Given key k, generated bucket address b = h(k)
  - Find the first empty slot at b from (1,2,…,N/B) and store key there
  - If all N/B slots are full, store key in an overflow bucket
- Search:
  - Given a search key sk, generate bucket address b = h(sk)
  - Search for key sk at bucket b, return if found, return NF if free slots
  - If sk is not found at b, search in overflow bucket, return if found
  - If sk is not found in overflow bucket, return NF

Bucketed Hashing

- Deletion:
  - An important step to ensure correct search behavior
  - e.g. Insert 4 keys in sequence {A, J, M, S} to a given bucket b
  - Hash Assignment: 1 -> A, 2 -> J, 3 -> M, 4 -> S
  - Delete M, new assignment: 1 -> A, 2 -> J, 3 -> empty, 4 -> S
  - Search S, will return empty since search terminated at 3
  - Solution: mark 3 as TOMBSTONE and continue search
  - Correct assignment: 1 -> A, 2 -> J, 3 -> TOMBSTONE, 4 -> S
  - Modify insertion to add record at TOMBSTONE as well

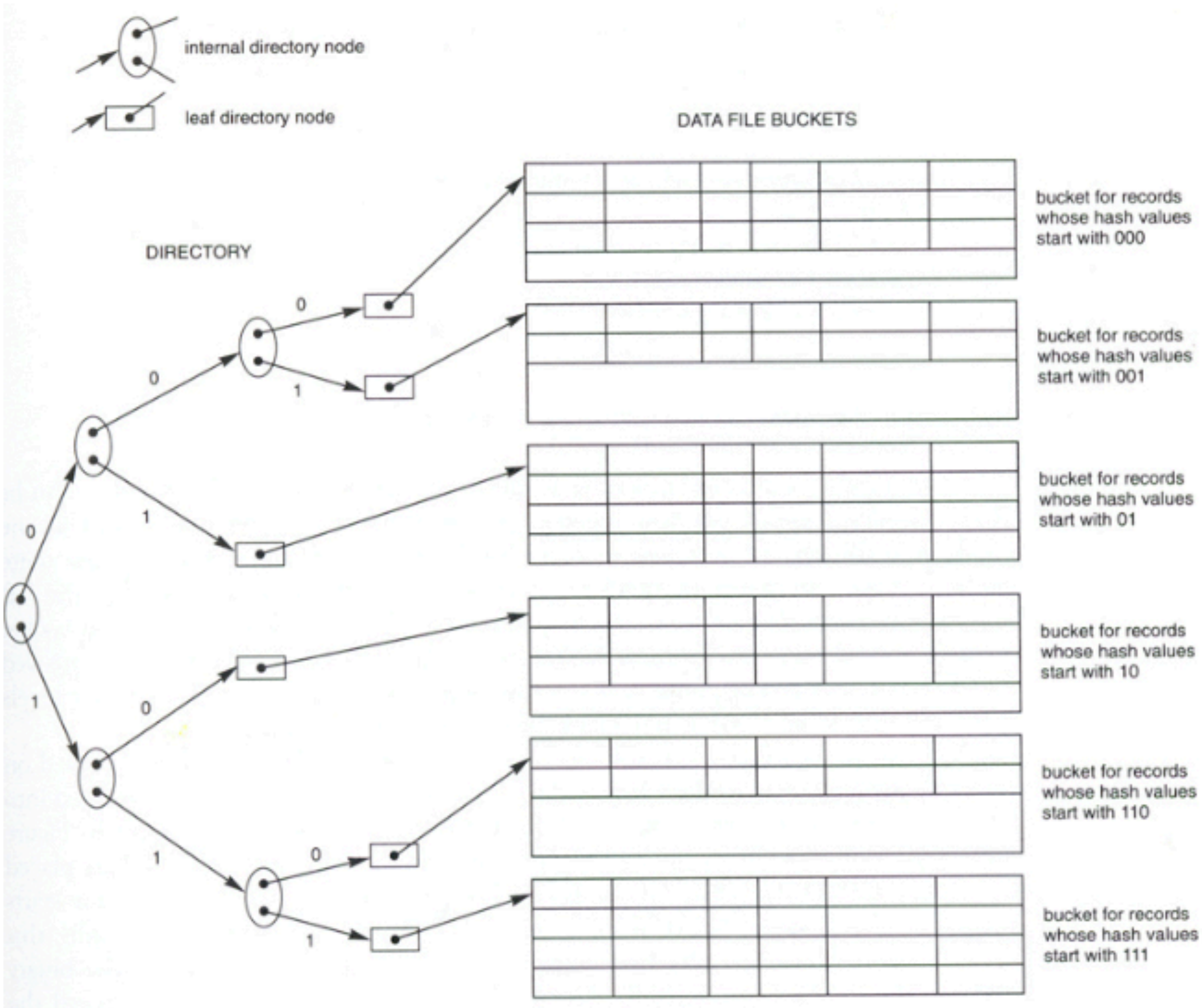
Static vs. Dynamic Hashing

- Static Hashing
  - Number of buckets is fixed
  - Expensive to re-organize or re-hash the whole file
- Dynamic Hashing
  - Number of buckets grows/shrinks dynamically
  - Low cost

Dynamic Hashing (1/2)

- Core Idea
  - Start with 1 bucket to store the records
  - Continue till bucket is full
  - If new record is added, split 1 bucket into 2 buckets
  - Records are distributed based on 1-MSB of binary hash value
- Directory
  - Internal node to guide the search
  - Leaf node to point to bucket

Dynamic Hashing (2/2)

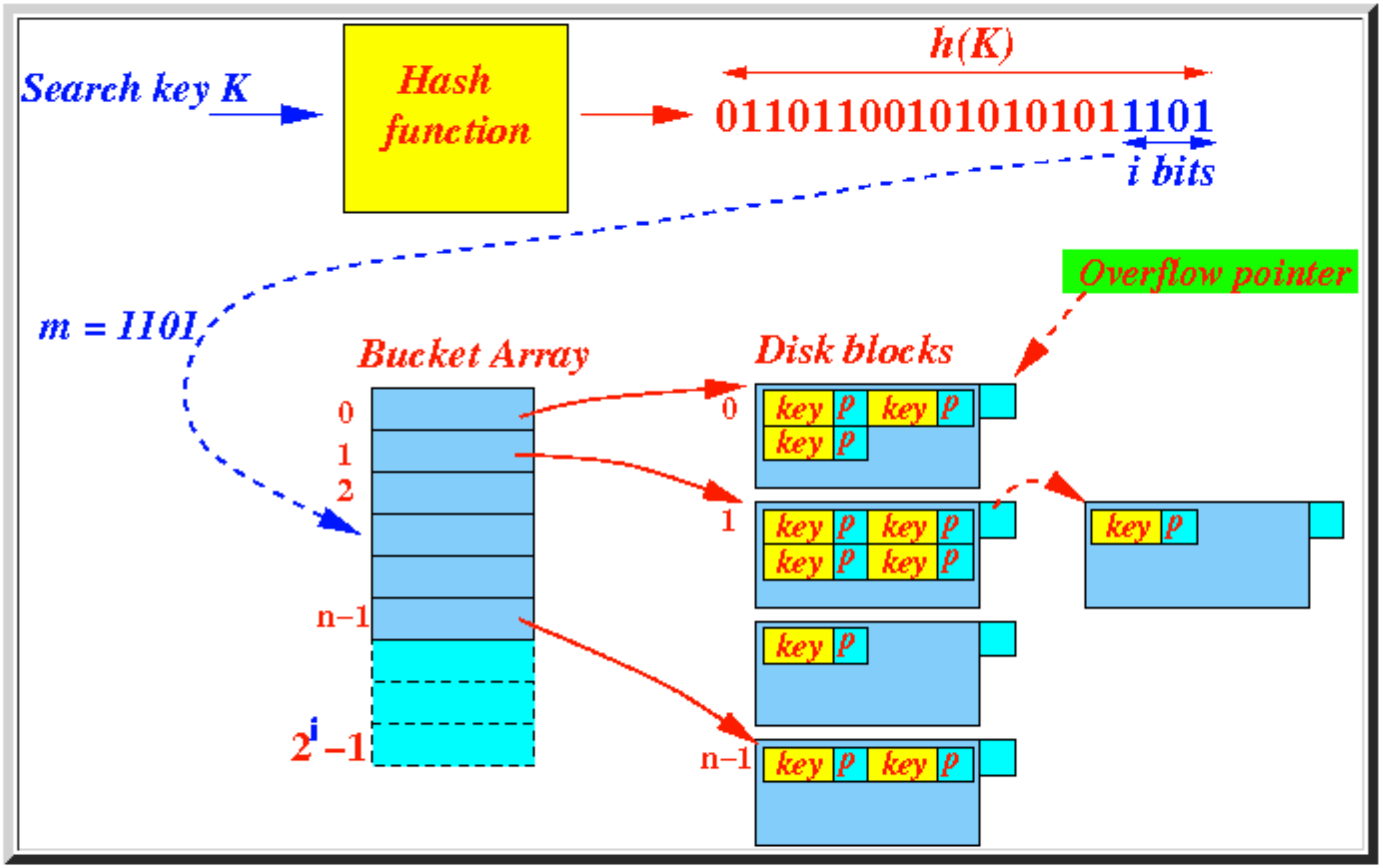


Fundamentals of Database Systems by Elmasri and Navathe 2nd edition

Linear Hashing

- Core Idea
  - Allows dynamic shrinking/growing of buckets without directory
  - When avg. occupancy per bucket > threshold, split happens
  - Growth rate of bucket is linear
  - Initial hash function: $h_0 = h(k) = k \mod M$
  - Uses a family of hash functions
  - Example shown next

Linear Hashing



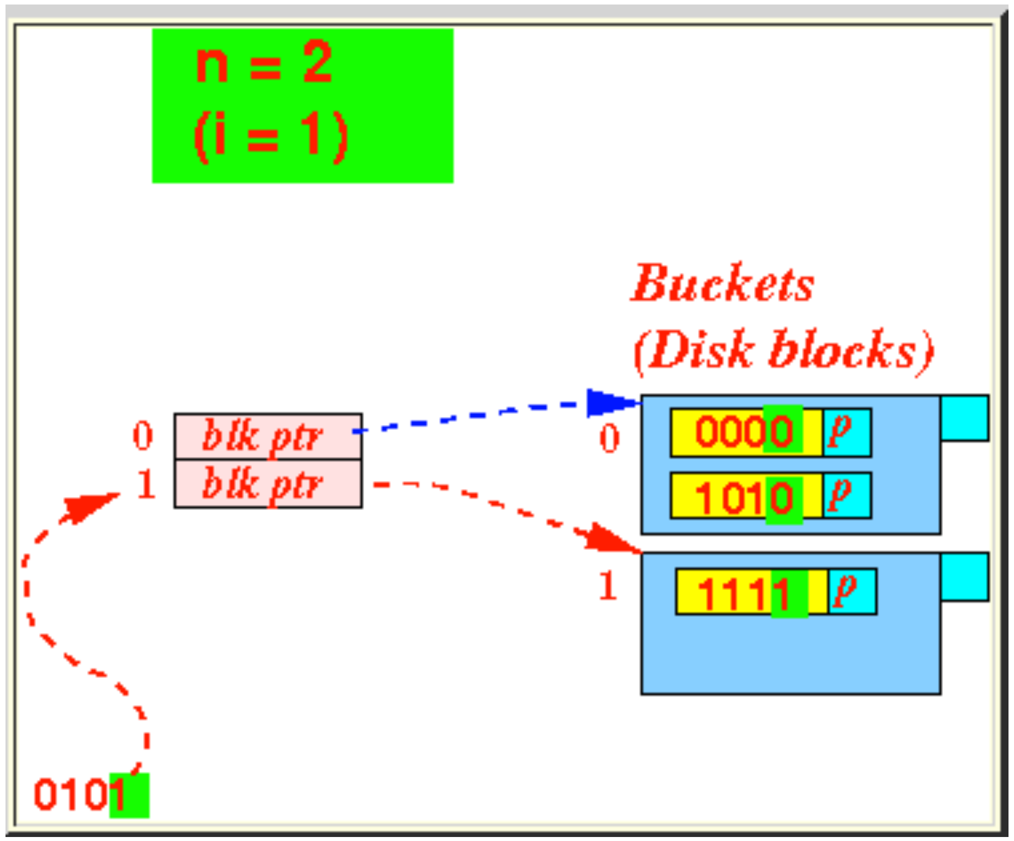
- Core Idea
  - "n" real buckets, last "i" bits from hash address are used
  - "i" bits can address 2^i buckets which can be > n
  - Bucket numbers > (n-1) are called virtual buckets and have their first bit as 1, 1XXXXXX

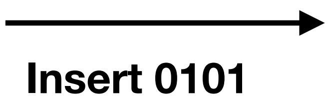
http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/3-index/linear-hashing.html

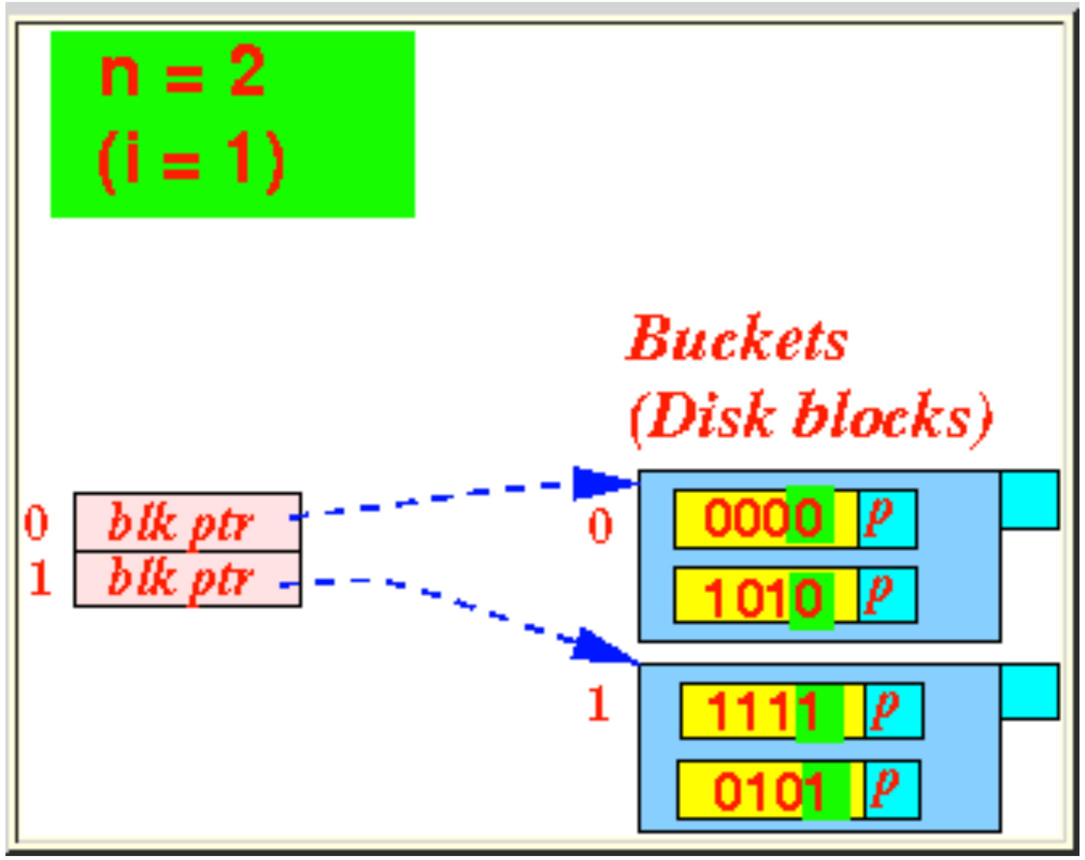
Linear Hashing

- Split Criteria
  - r is the number of records
  - n is the number of buckets
  - b is the bucket size or number of keys that can be stored in bucket
  - Avg occupancy = $\dfrac{r}{n * b}$
  - Split when avg. occupancy > some threshold
  - Increase criteria: n = n + 1

Linear Hashing Example



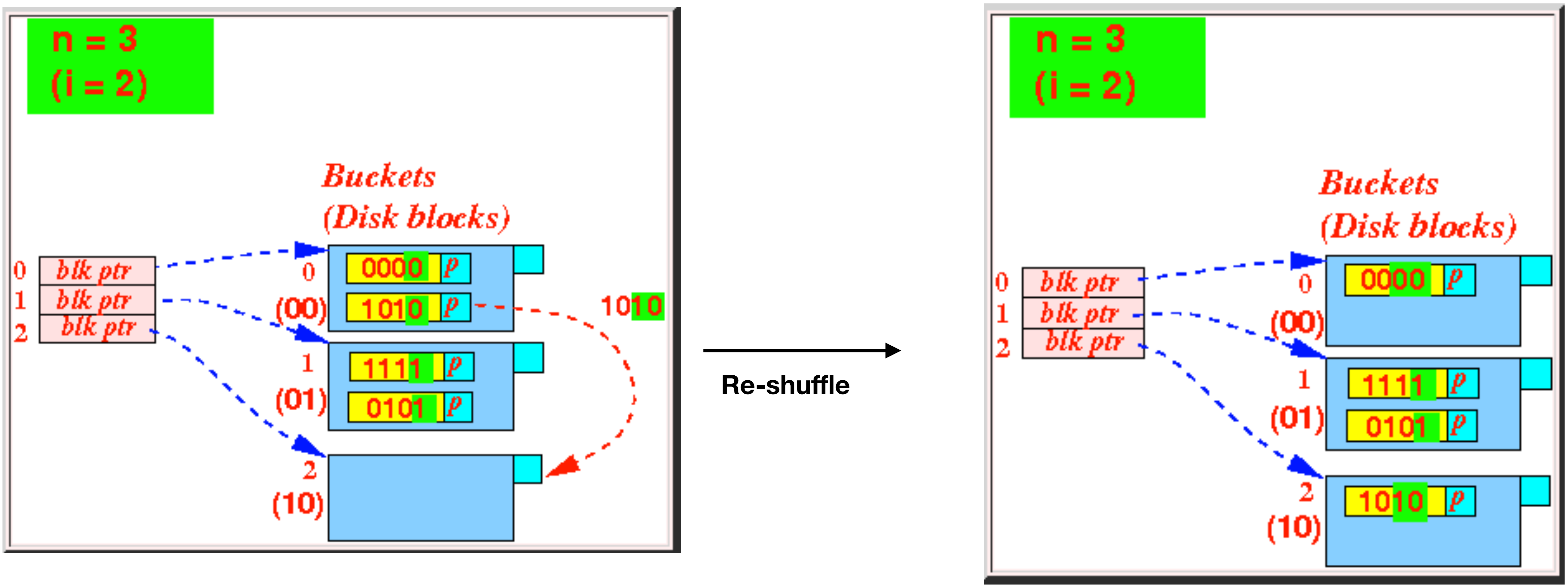




- Avg Occupancy before Insertion = 3 / (2*2) = 0.75
- Avg Occupancy after insertion = 4 / (2*2) = 1
- Let's assume split threshold = 0.85
- Since 1 > 0.85, we need to split and add a block

http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/3-index/linear-hashing.html

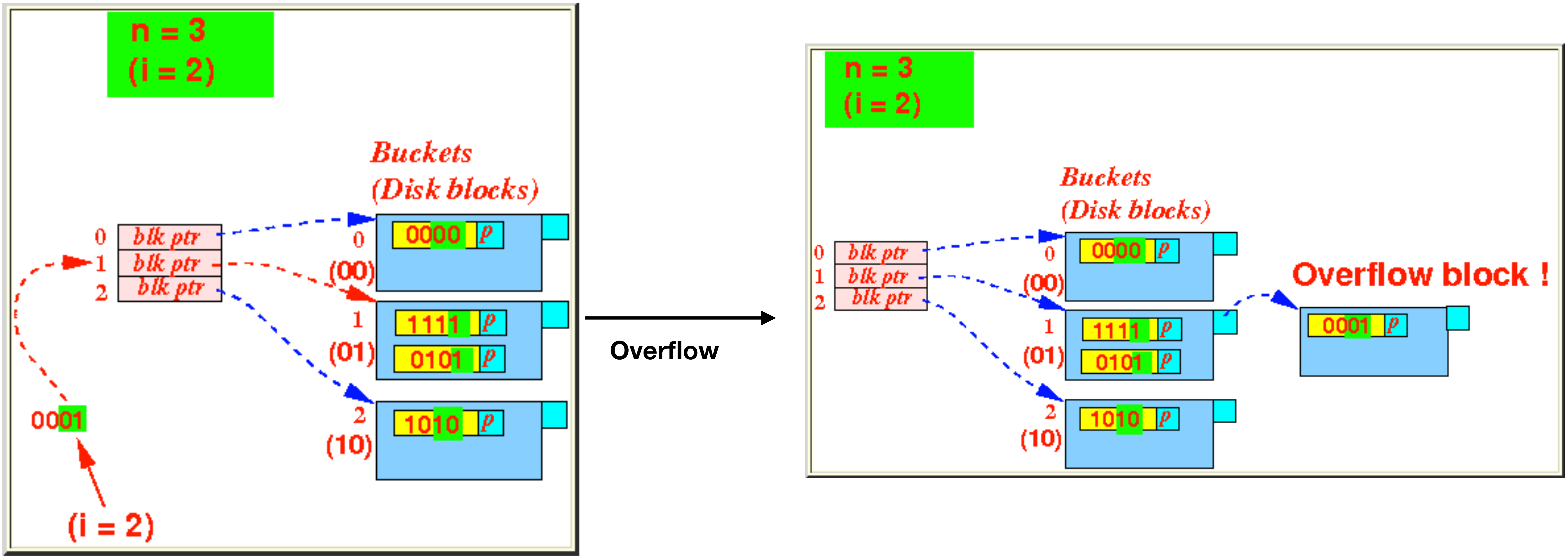
Linear Hashing Example



- Add bucket 2 with id 01, notice i changes from 1 to 2 since we now need 2 bits to address buckets
- Transfer key from bucket 00 to bucket 10
- Avg. occupancy after the split and shuffle: 4 / (3*2) = 0.67 < 0.85
- How to search key 1111 ?
  - i = 2, bucket address 11 does not exist or is a virtual bucket, Flip the MSB from 1 to 0, 11 -> 01

http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/3-index/linear-hashing.html

Linear Hashing Example



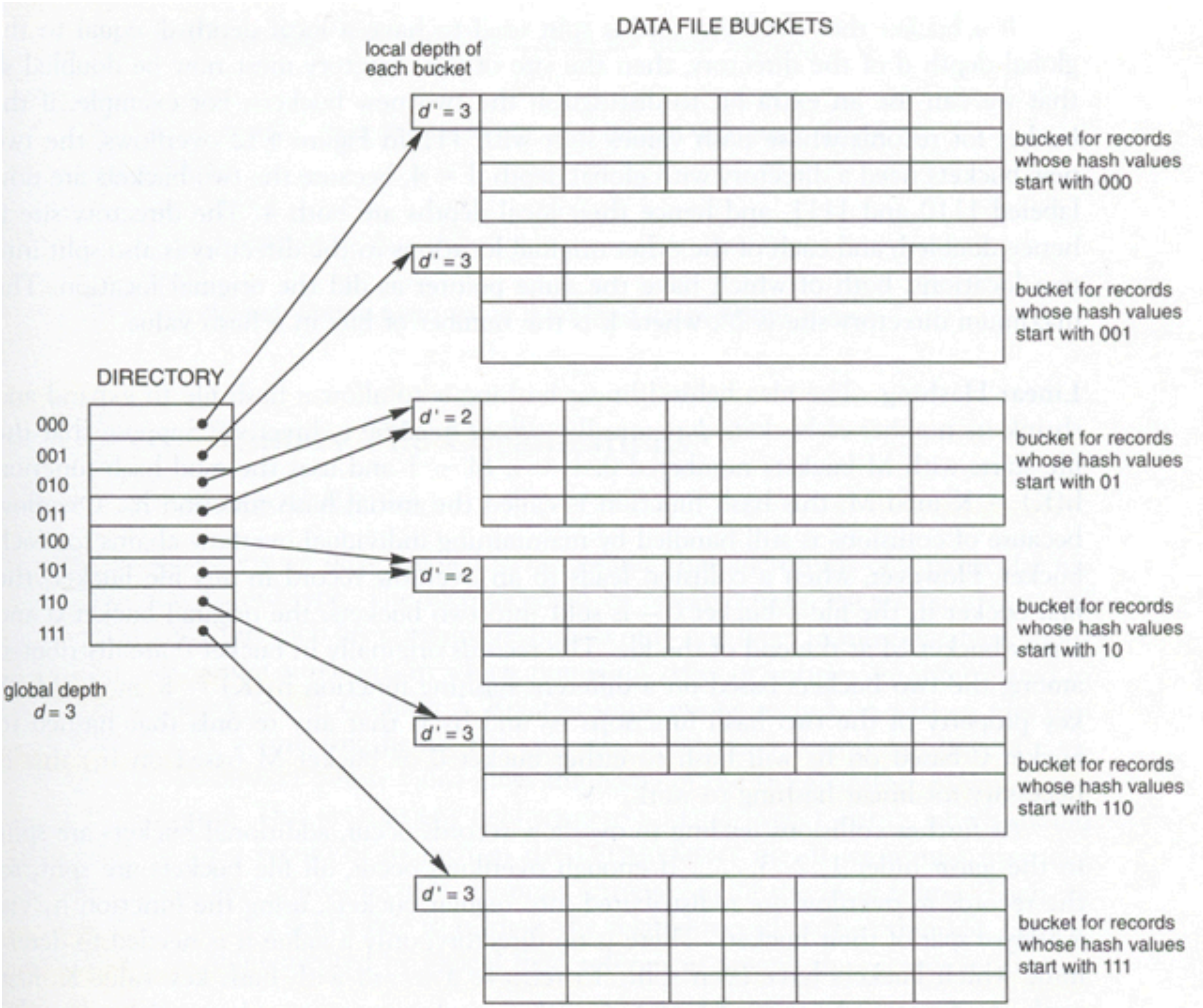
- Avg. occupancy after overflow: 5 / (3*2) = 0.83 < 0.85

http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/3-index/linear-hashing.html

Extendible Hashing (1/2)

- Core Idea
  - Variant of dynamic hashing with different directory structure
  - Directory contains 2^d addresses; d = Global depth
  - Local depth $= d'$
  - Local depth determines number of bits to use for hash values
  - If a bucket fills up, a split happens
  - Example shown next
  - Issue: exponential increase in size of hash table

Extendible Hashing (2/2)



When this bucket fills up, it splits into 2 buckets of local depth = 3 containing hash values 010 and 011

Fundamentals of Database Systems by Elmasri and Navathe 2nd edition