Indexing: Hash Tables

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

Thanks to Doris Xi for her help with preparing the slides.

Learning Objectives

After this lecture, you should be able to:

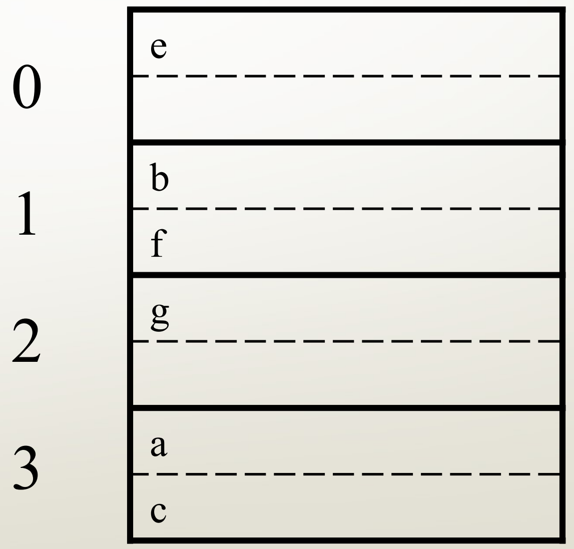
- Describe how to search, insert and delete keys from
  - Secondary storage Hash Table (HT)
  - Extensible HT
  - Linear HT

Hash Tables

- Secondary storage hash tables are much like main memory ones
- Recall basics:
  - There are B *buckets*
  - A hash function h(k) maps a key k to {0, 1, …, B-1}
  - Store in bucket h(k) a pointer to record with key k
- Secondary storage: bucket = block
  - Store in the block of bucket h(k) any record with key k
  - use overflow blocks when needed

Hash Table Example

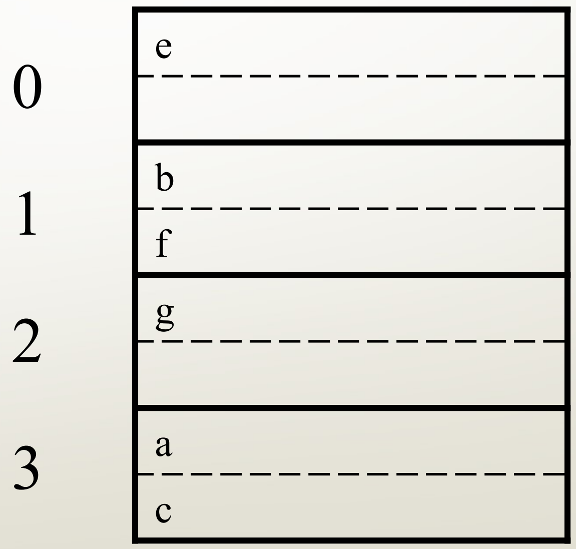
- Assume 1 bucket (block) stores 2 records
- h(e)=0
- h(b)=h(f)=1
- h(g)=2
- h(a)=h(c)=3



Searching in a Hash Table

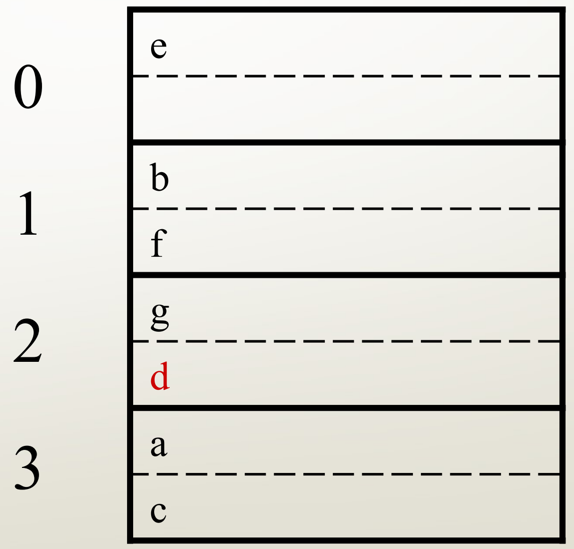
- Search for a:
- Compute h(a)=3
- Read bucket (block) 3
- 1 disk access

Main memory may have an array of pointers (to buckets) accessible by bucket number.



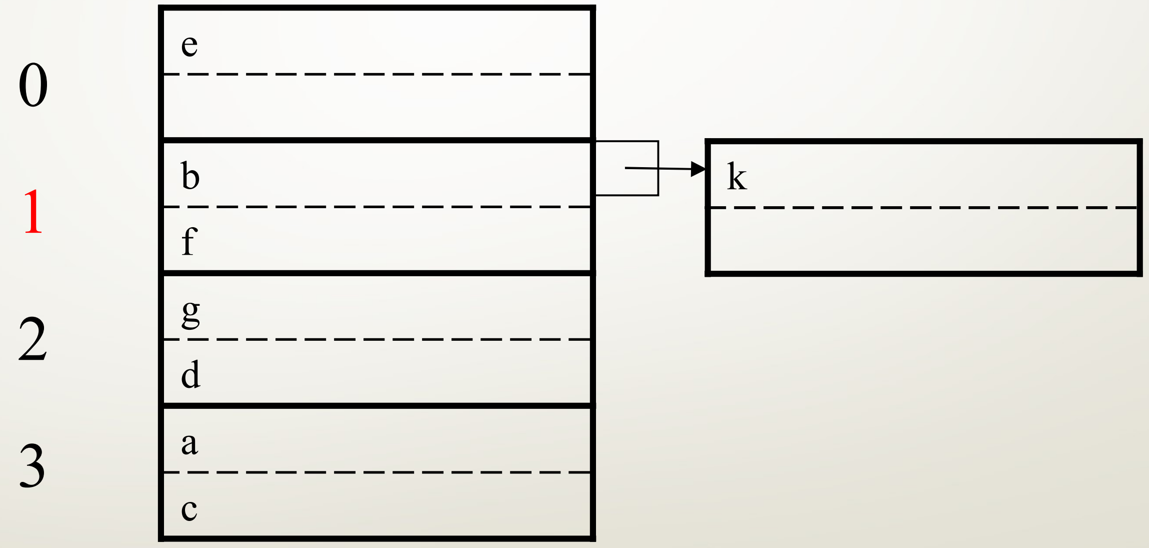
Insertion in Hash Table

- Place in right bucket (block), if space
- E.g. h(d)=2



Insertion in Hash Table

- Create overflow block, if no space
- E.g. h(k)=1



More over-flow blocks may be needed

Hash Table Performance

- Fixed number of buckets
- Excellent, if no overflow blocks
- Degrades considerably when there are many overflow blocks.
  - Might need to go through a chain of overflow blocks

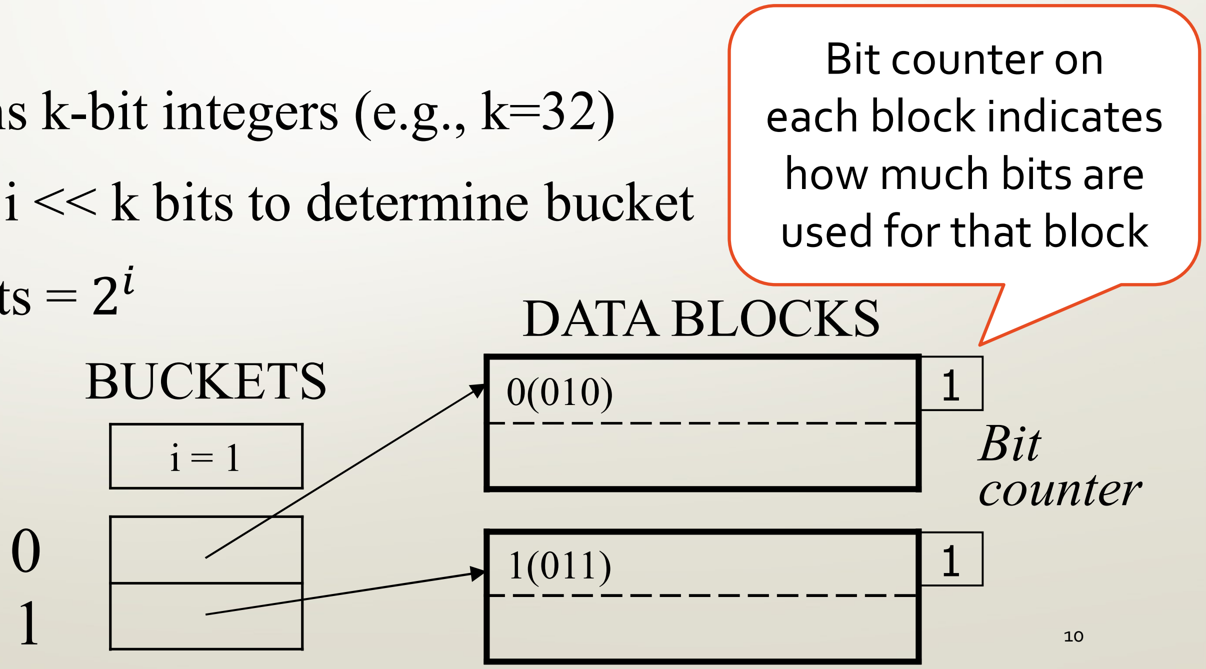
Can improve this by allowing the number of buckets to grow

Outline

- Hash Tables
  - ✓ Secondary storage HT
  - Extensible HT
  - Linear HT

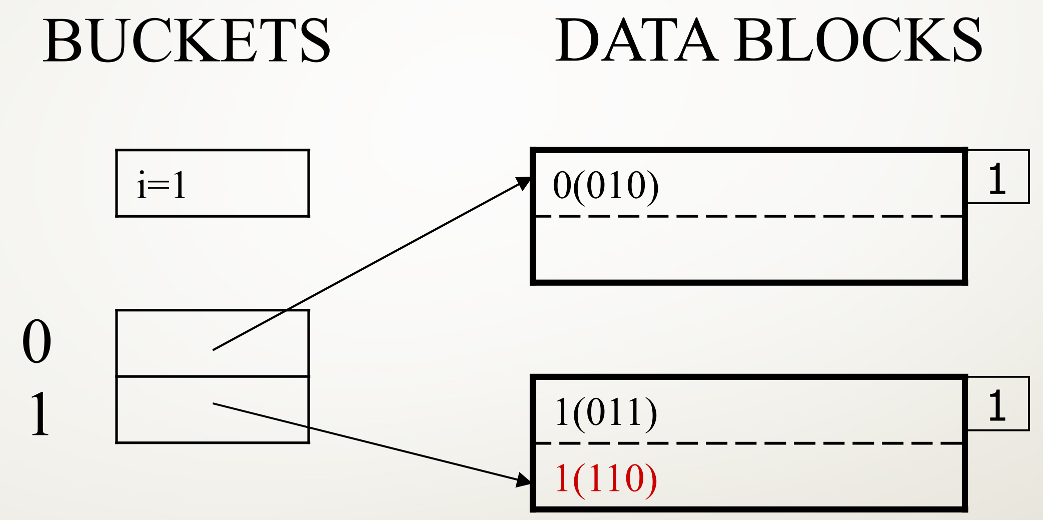
Extensible Hash Table

- Array of pointers to blocks instead of array of blocks
- Size of array is allowed to grow. 2x size when it grows
- Don't need a block per bucket. Sparse buckets share a block
- Hash function returns k-bit integers (e.g., k=32)
  - Only use the first i << k bits to determine bucket
  - Number of buckets = 2^i


Bit counter on each block indicates how much bits are used for that block
DATA BLOCKS
BUCKETS
i = 1
0
1
0(010)
1(011)
1
1
Bit counter

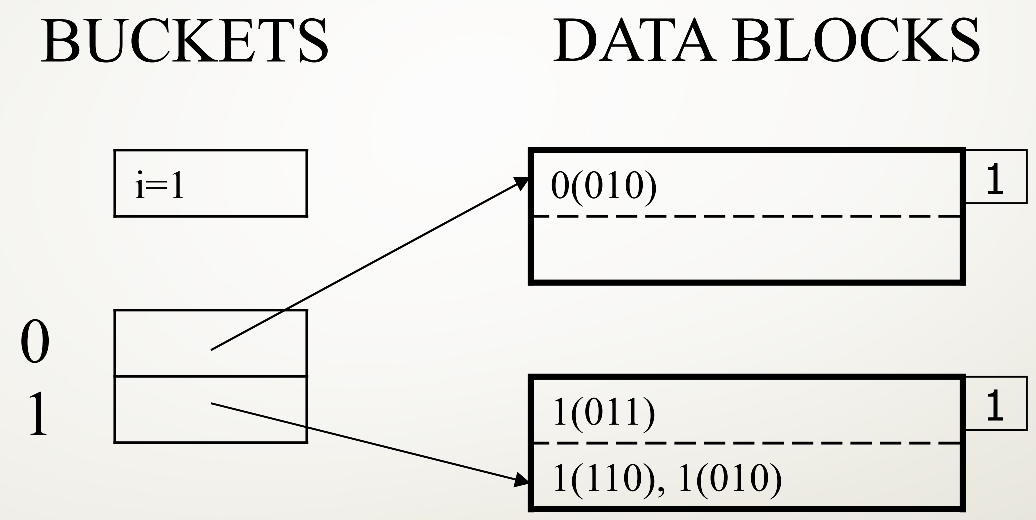
Insertion in Extensible Hash Table

- Insert 1110



Insertion in Extensible Hash Table

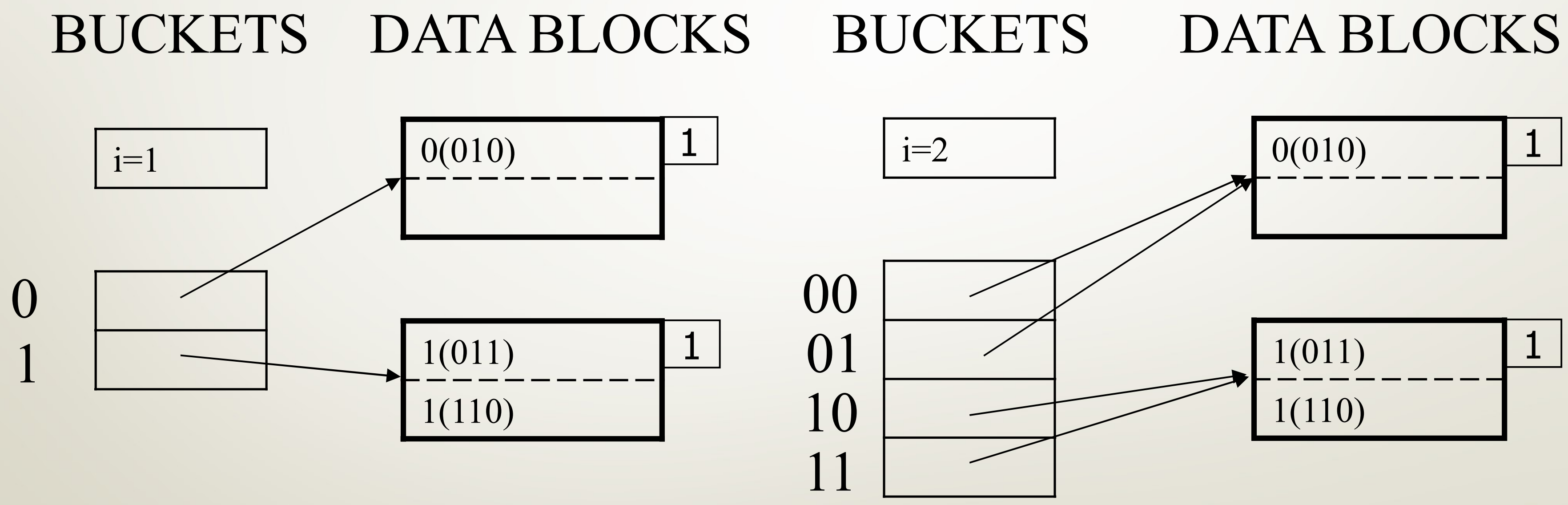
- Now insert 1010



- Need to split block and extend bucket array
- i becomes 2: done in two steps

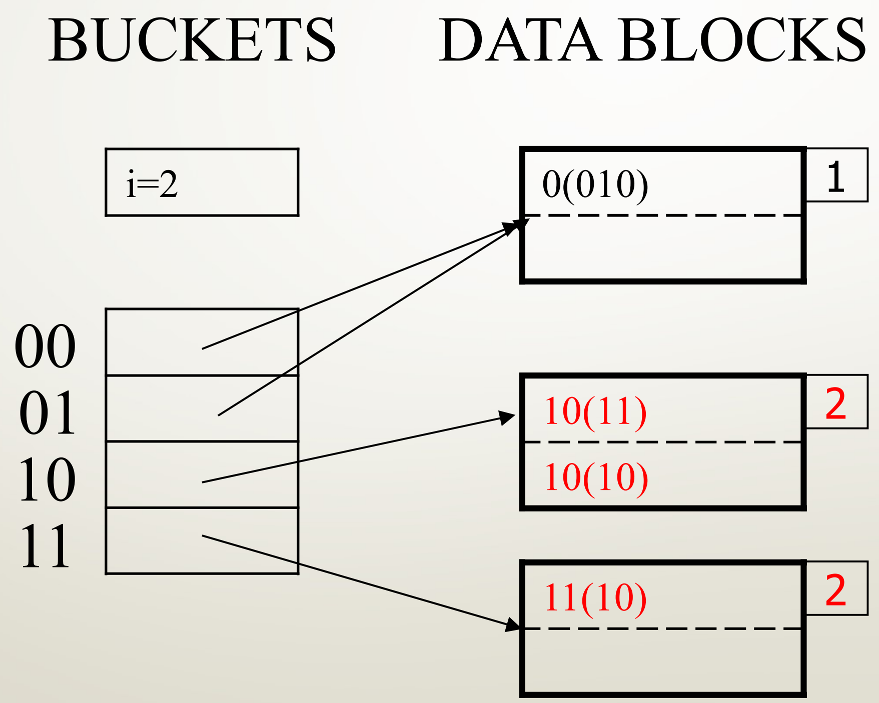
Insertion in Extensible Hash Table

Step 1: Extend the buckets



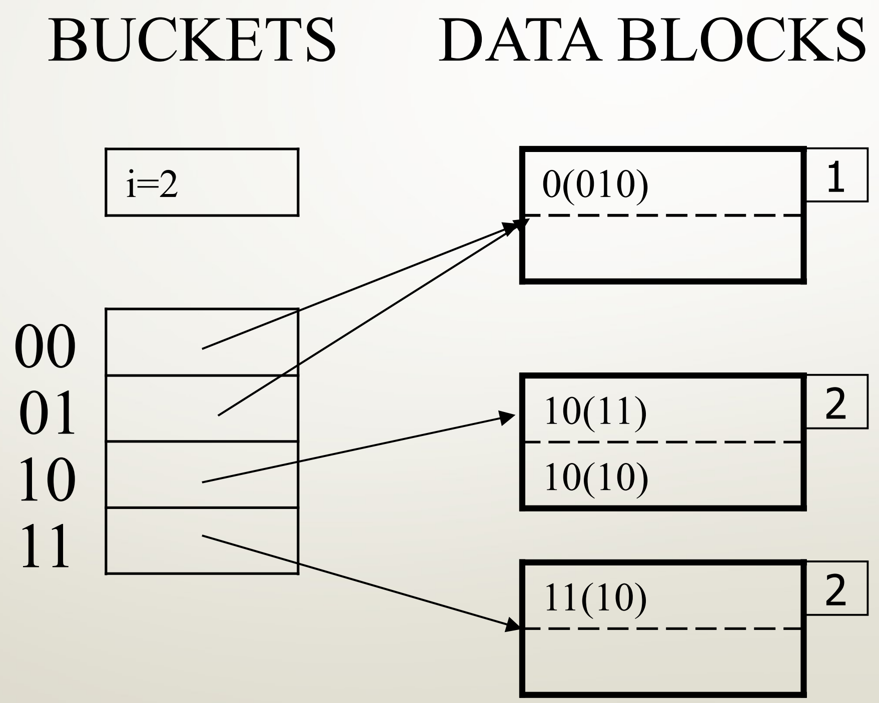
Insertion in Extensible Hash Table

Step 2: Now try to insert 1010



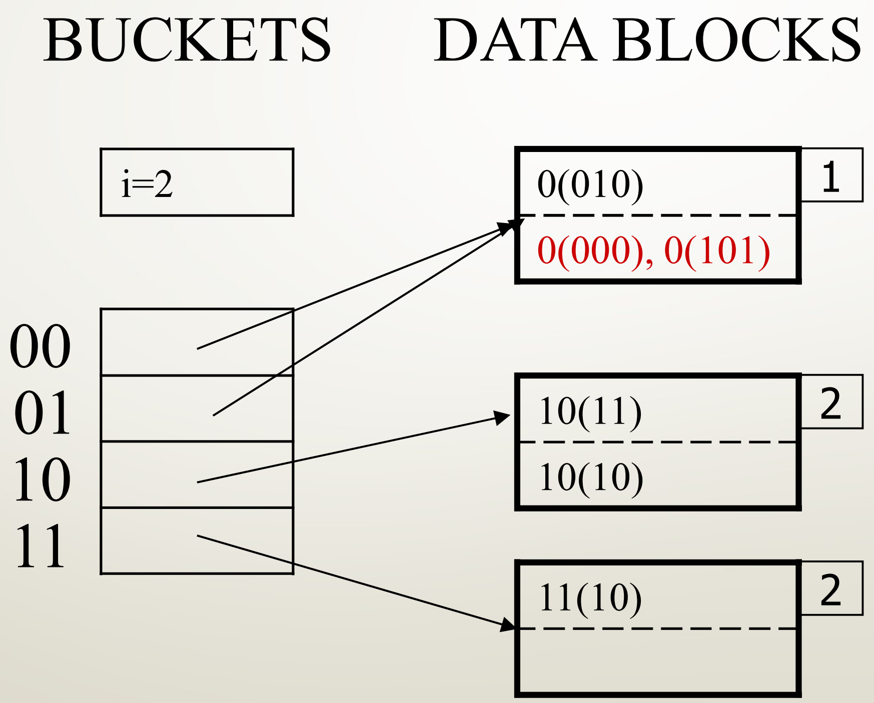
Insertion in Extensible Hash Table

- Now insert 0000: where would it go? Then 0101?
- Need to split block, but not bucket array



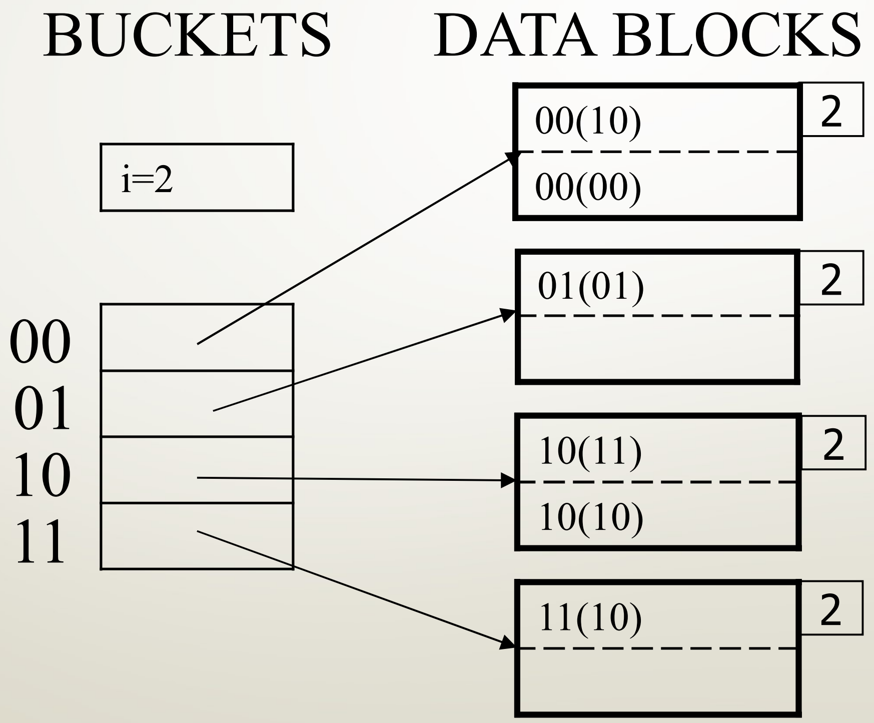
Insertion in Extensible Hash Table

- Now insert 0000: where would it go? Then 0101?
- Need to split block, but not bucket array



Insertion in Extensible Hash Table

- Now insert 0000: where would it go? Then 0101?
- Need to split block, but not bucket array



Performance: Extensible Hash Table

- No overflow blocks: access always one read for distinct keys
- BUT:
  - Extensions can be costly and disruptive
  - After an extension bucket table may no longer fit in memory
  - Imagine three records whose keys share the first 20 bits. These three records cannot be in same block (assume two records per block). But a block split would require setting i = 20, i.e., accommodating for 2^20 = 1 million buckets, even though there may be only a few hundred records.

Outline

- Hash Tables
  - ✓ Secondary storage HT
  - ✓ Extensible HT
  - Linear HT

Linear Hash Table

- Idea 1: add only one bucket at a time

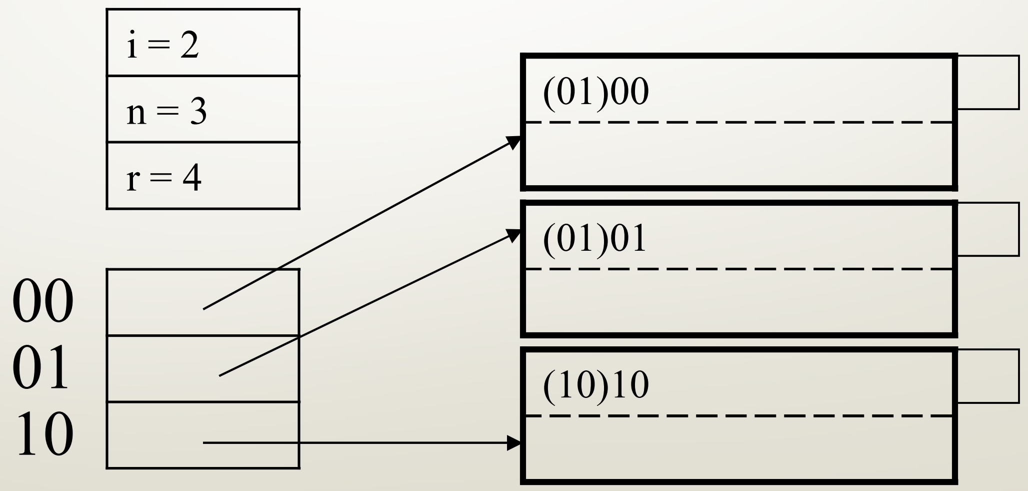
  Problem: n = no longer a power of 2
- Let i be # bits necessary to address n buckets.
  - i = ceil(log_2 n)
- After computing h(k), use ***last*** i bits:
  - If last i bits represent a number (say m) < n, store the key in bucket m
  - If m >= n, change msb from 1 to 0 (get a number < n)
- Idea 2: allow overflow blocks (not expensive to overflow)
- Convention: Read from the right (as opposed to the left)

Linear Hash Table Example

- N=3 <= 2^2 = 4
  - Therefore, only buckets until 10

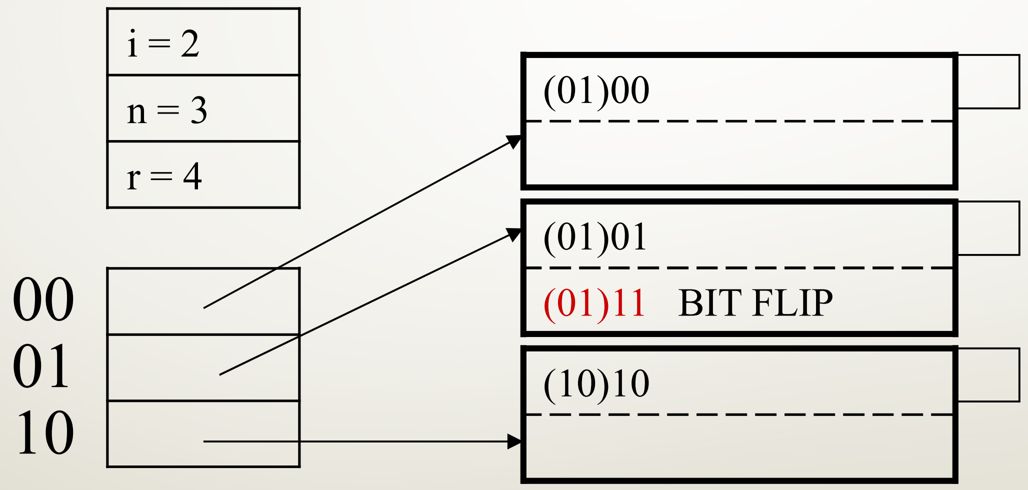
Try to insert 01<u>11</u>

11 is flipped => 01



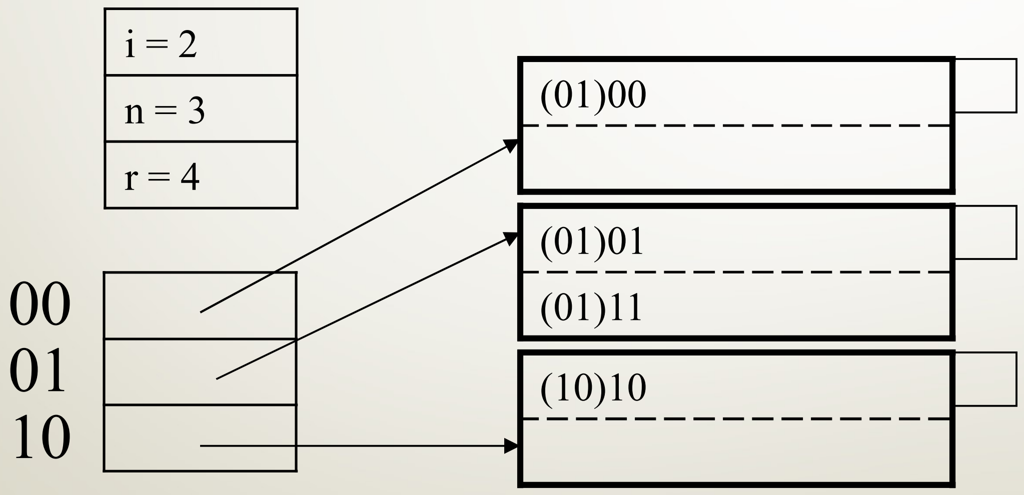
Linear Hash Table Example

- After inserting 0111



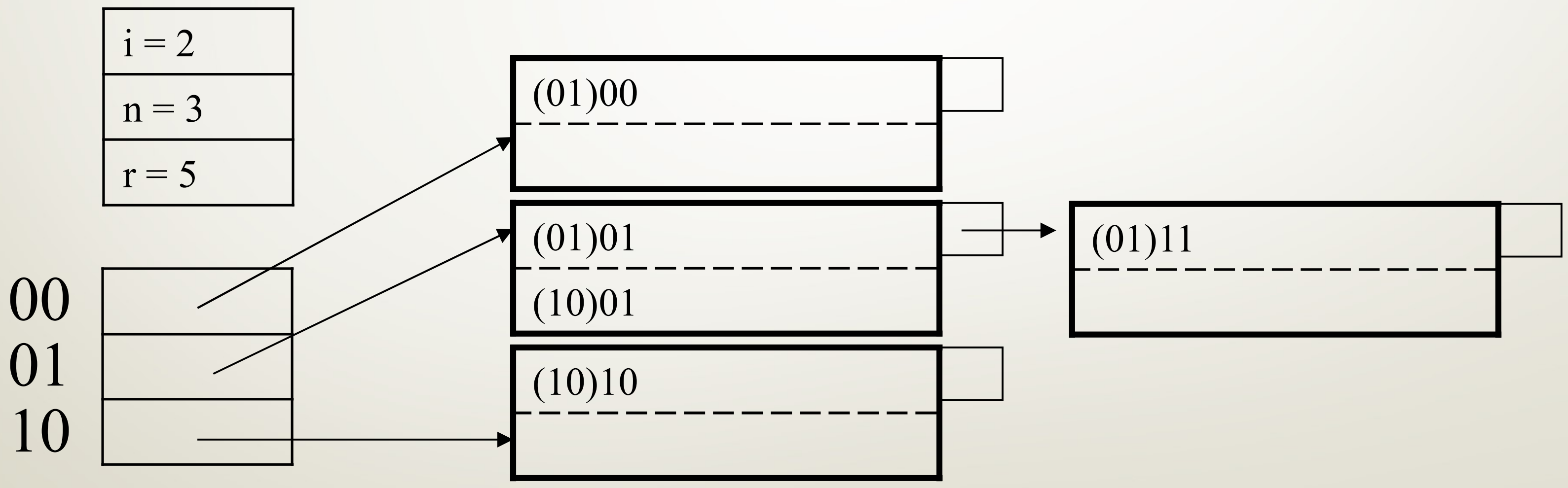
Linear Hash Table Example

- Insert 1001:



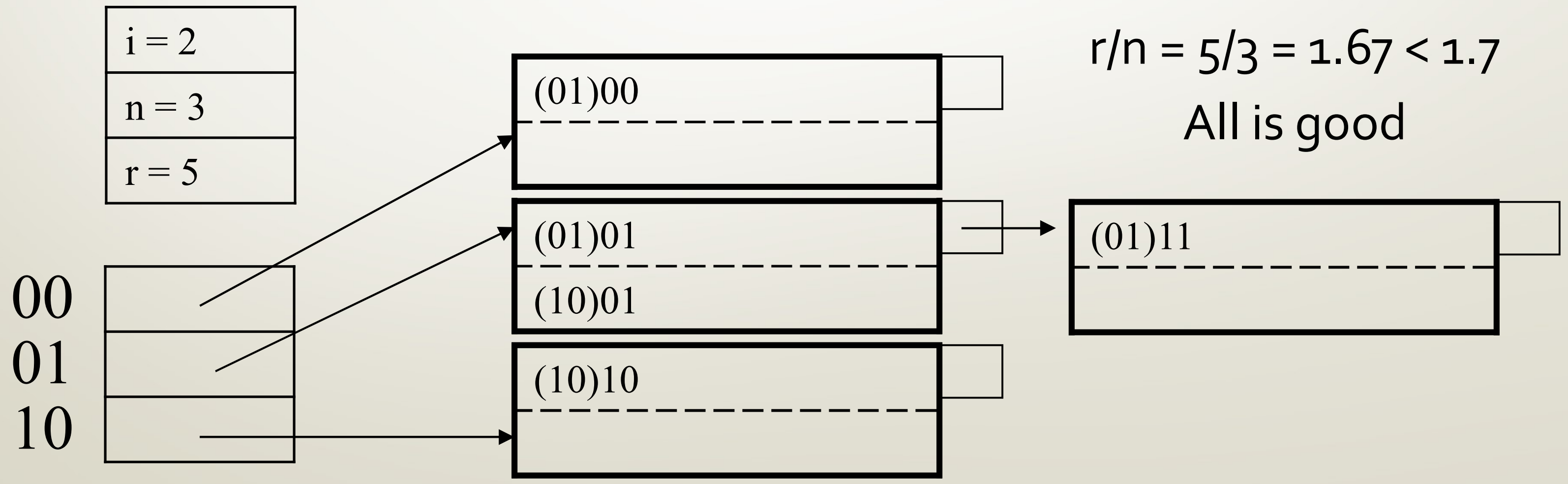
Linear Hash Table Example

- Insert 1001: overflow blocks…



Linear Hash Tables

- Extend n → n+1 when average number of records per bucket exceeds (say) 85% of total number of records per block
  - e.g., r/n <= 0.85 * 2 = 1.7 (for block size = 2)
- Until then, use overflow blocks (cheaper than adding buckets)



r/n = 5/3 = 1.67 < 1.7

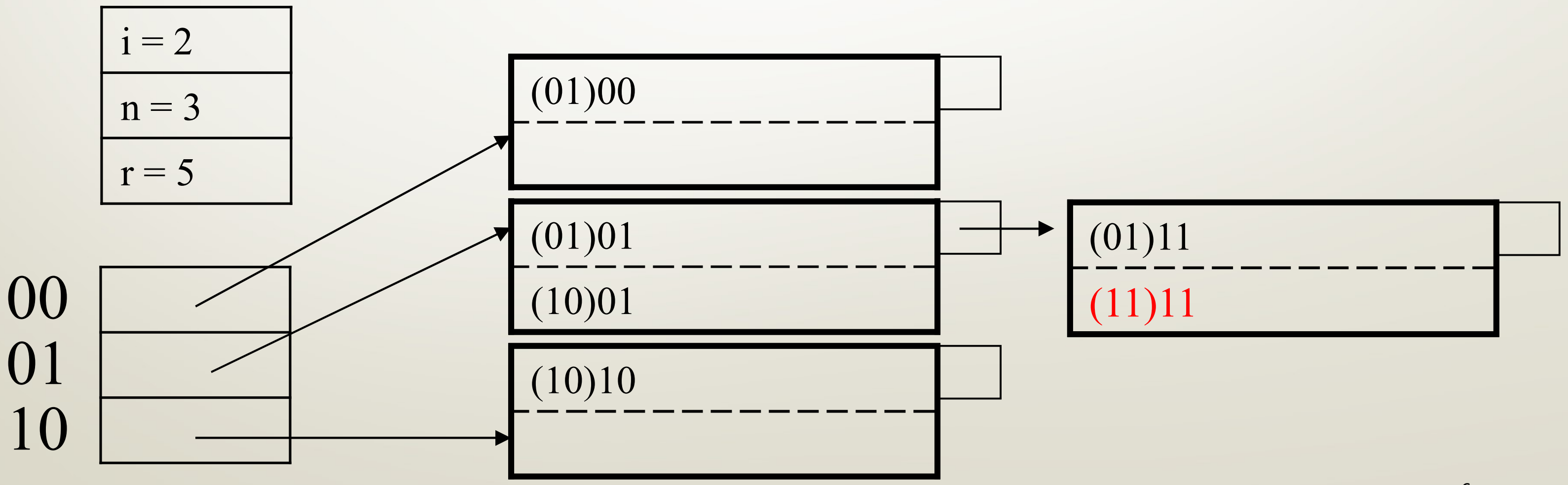
All is good

Linear Hash Tables

- Try to insert 1111

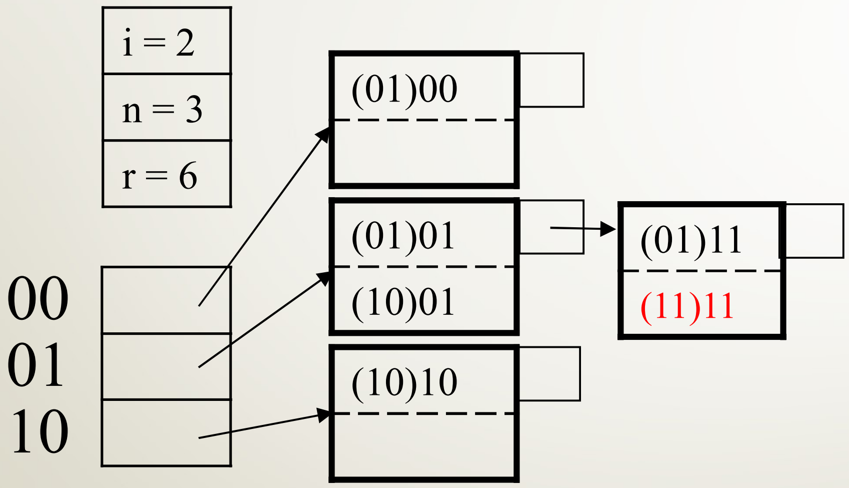
r/n = 6/3 = 2 > 1.7

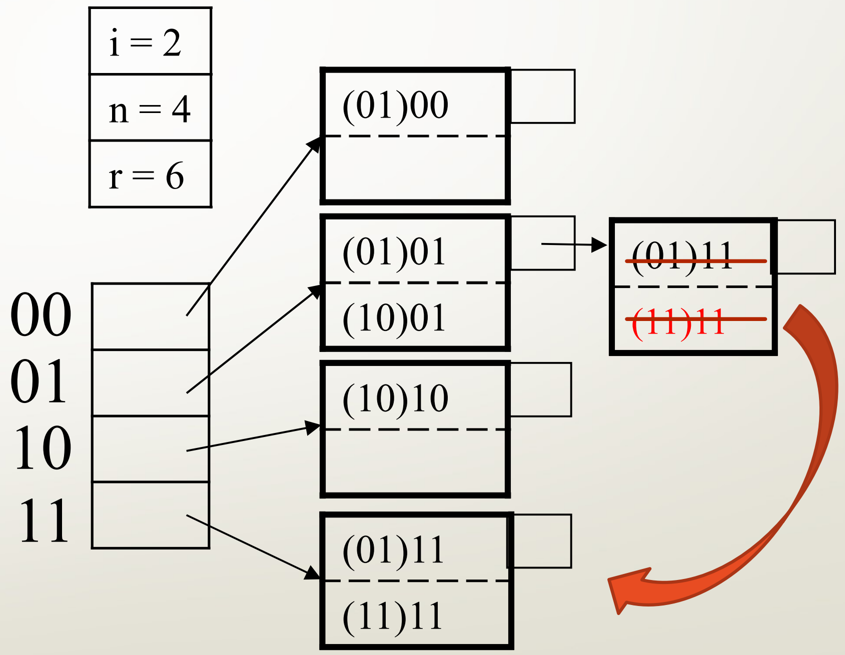
→ Time to add a bucket



Linear Hash Table Extension

- From n=3 to n=4



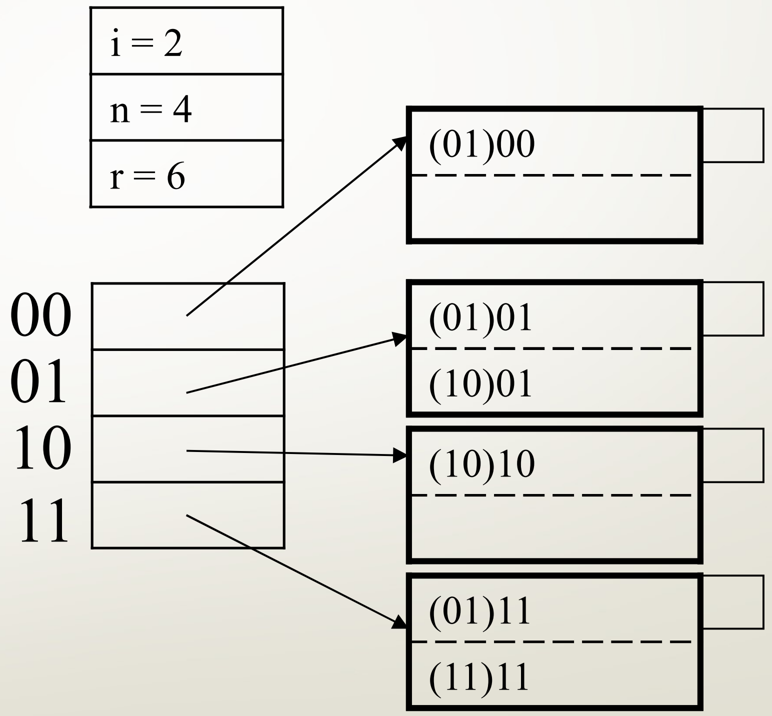


- Only need to touch one block (which one ?)

Linear Hash Table Extension

- From n=3 to n=4 finished

r/n = 6/4 = 1.5 < 1.7 ✓



Indexing Summary

- B+ Trees (search, insertion, deletion)
  - Good for point and range queries
  - Log time lookup, insertion and deletion because of balanced tree
- Hash Tables (search, insertion)
  - Static hash tables: one I/O lookup, unless long chain of overflow
  - Extensible hash tables: one I/O lookup, extension can take long
  - Linear hash tables: ~ one I/O lookup, cheaper extension
- No panacea; dependent on data and use case