Data Storage and Query Answering

Indexing and Hashing (4)

Hash Tables

Introduction

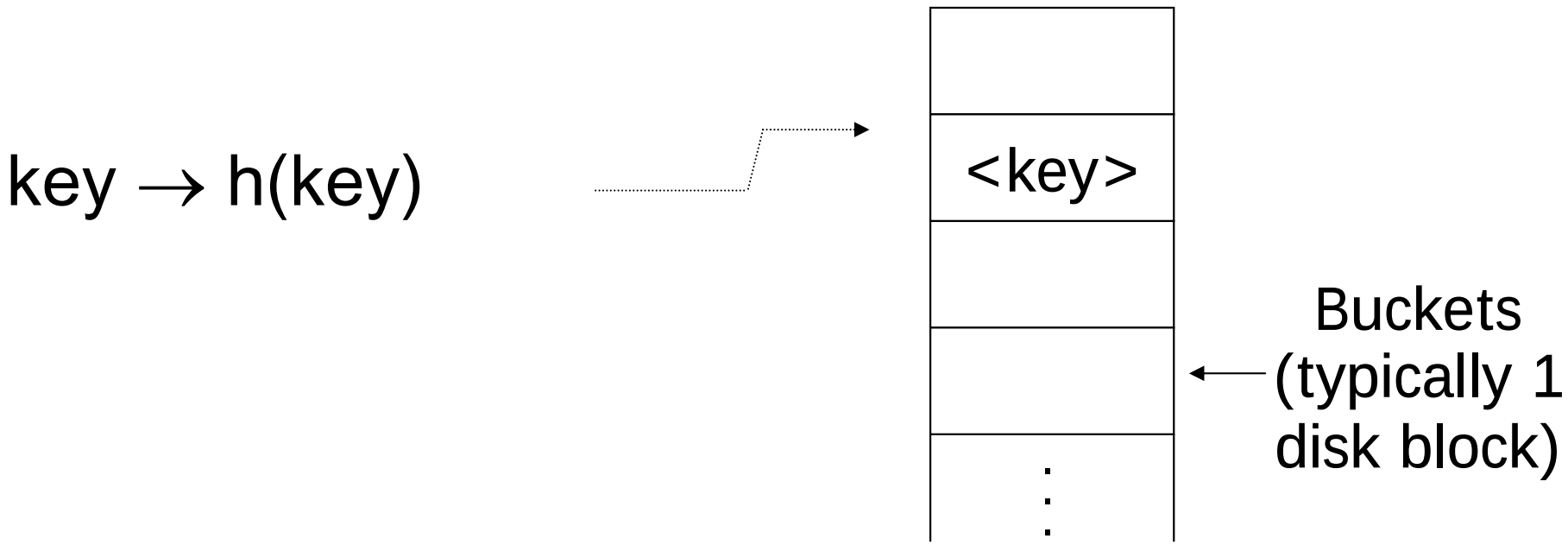
- Tree-based index structures map search key values to record addresses via a tree structure.
- Hash tables perform the same mapping via a hash function, which computes the record address.
- Search key K
- *Hash function* h

$$h(K) \in \{0, \ldots, B-1\}$$

- B: number of buckets.

Hash Tables

Hashing



Hash Tables

Introduction

- Good hash function should have the following property: expected number of keys the same (similar) for all buckets.
- This is difficult to accomplish for search keys that have a highly skewed distribution, e.g. names.
- Common hash function

  K = ‘$x_1\ x_2 \ldots x_n$’ n byte character string

  $$h(K) = (x_1 + \ldots + x_n)\ MOD\ B$$

  →B often chosen as prime number

Hash Tables

- This may not be the best function …
- Read Knuth Vol. 3 if you really need to select a good function.

Hash Tables

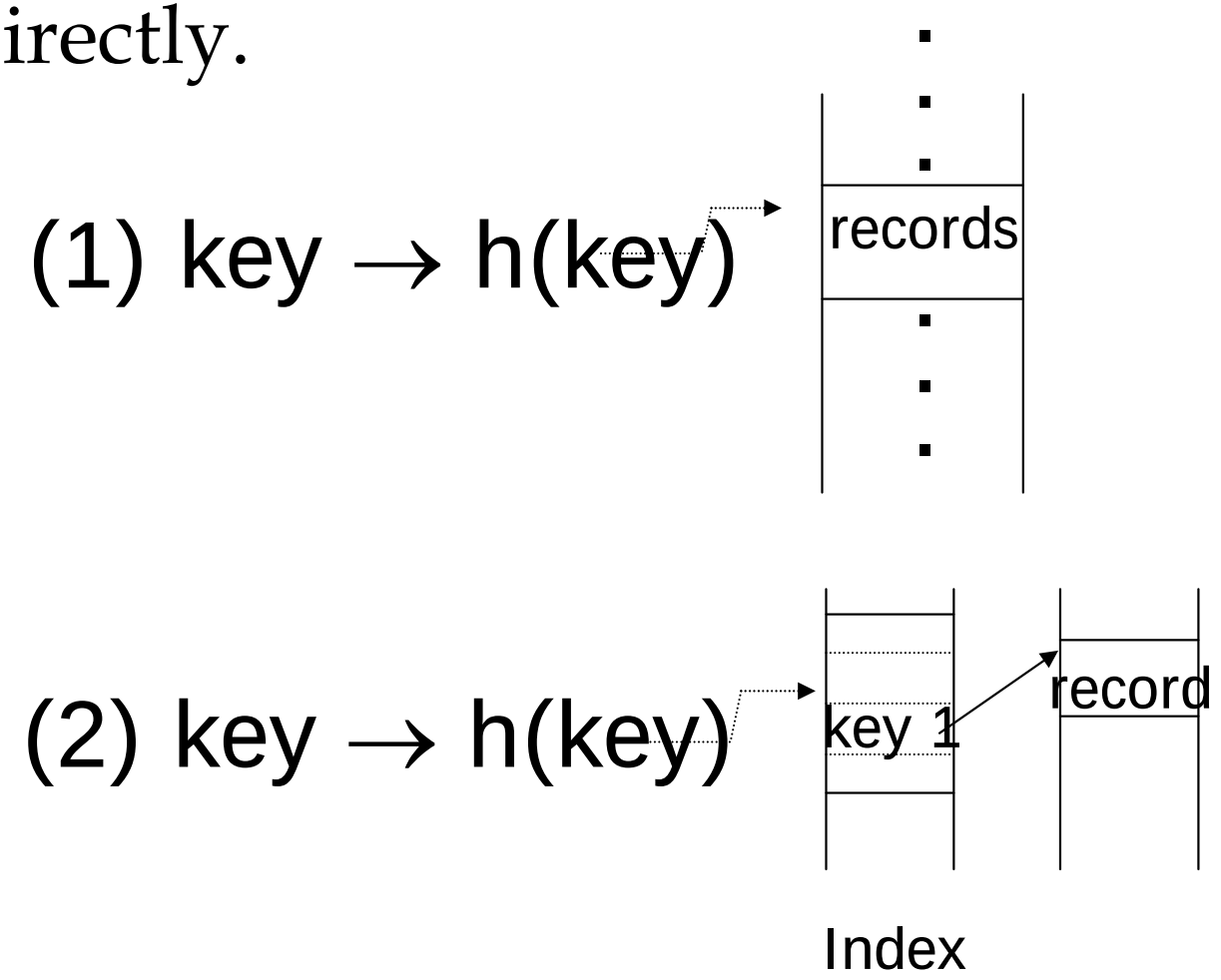
Secondary-Storage Hash Tables

- *Bucket*: collection of blocks.
- Initially, bucket consists of one block.
- Records hashed to b are stored in bucket b.
- If bucket capacity exceeded, link chain of overflow buckets.
- Assume that address of first block of bucket i can be computed given i.
- E.g., main memory array of pointers to blocks.

Hash Tables

Secondary-Storage Hash Tables

- Hash tables can perform their mapping directly or indirectly.



Within a Bucket

- Do we keep keys sorted?
- Yes, if CPU time critical
  & Inserts/Deletes not too frequent

Hash Tables

Insertions

- To insert record with search key K.
- Compute $h(K) = i$.
- Insert record into first block of bucket i that has enough space.
- If none of the current blocks has space, add a new block to the overflow chain, and store new record there.

Hash Tables

Insertions

INSERT:

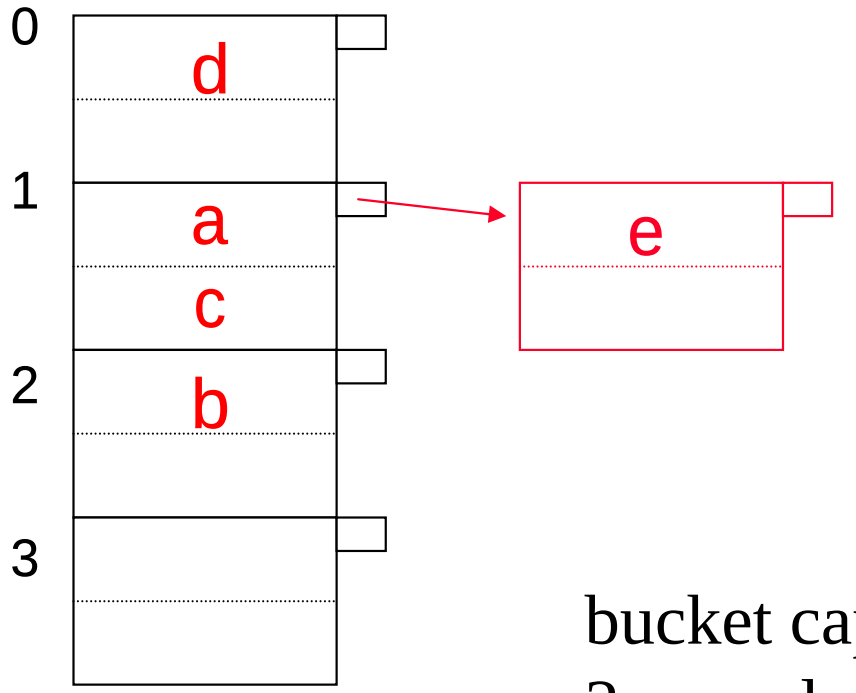
h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0

h(e) = 1



bucket capacity:
2 records

Hash Tables

Deletions

- To delete record with search key K.
- Compute $h(K) = i$.
- Locate record(s) with search key K in bucket i.
- If possible, move up remaining records within block.
- If possible, move remaining records from overflow chain to the previous block and de-allocate block.

Hash Tables

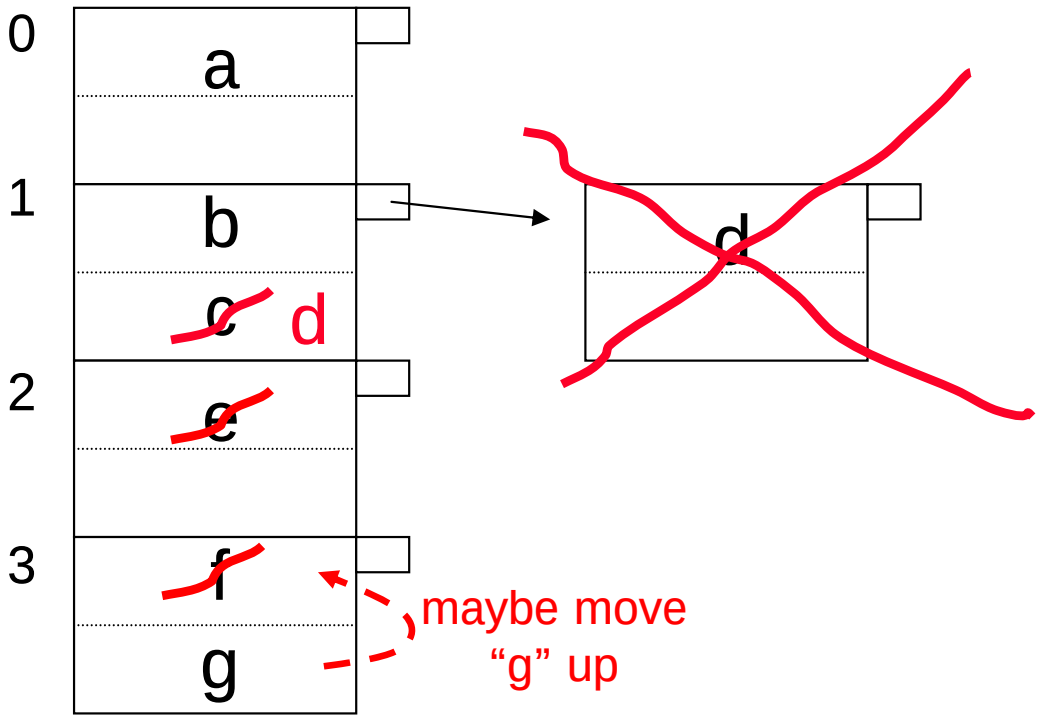
Deletions

Delete:

e

f

c



Hash Tables

Queries

- To find record(s) with search key K.
- Compute $h(K) = i$.
- Locate record(s) with search key K in bucket i, following the overflow chain.
- In the absence of overflow blocks, only one block I/O necessary, i.e. O(1) runtime.
- This is (much) better than B-trees.
- But hash tables do not support range queries!

Hash Tables

Queries

- In order to keep overflow chains short, keep space utilization between 50% and 80%.

$$space\ utilization\ (b) = \frac{\#\,\text{keys in bucket } b}{\#\,\text{keys that fit in } b}$$

- If space utilization < 50%: waste space.
- If space utilization > 80%: overflow chains become significant.
- Depends on hash function and on bucket capacity.

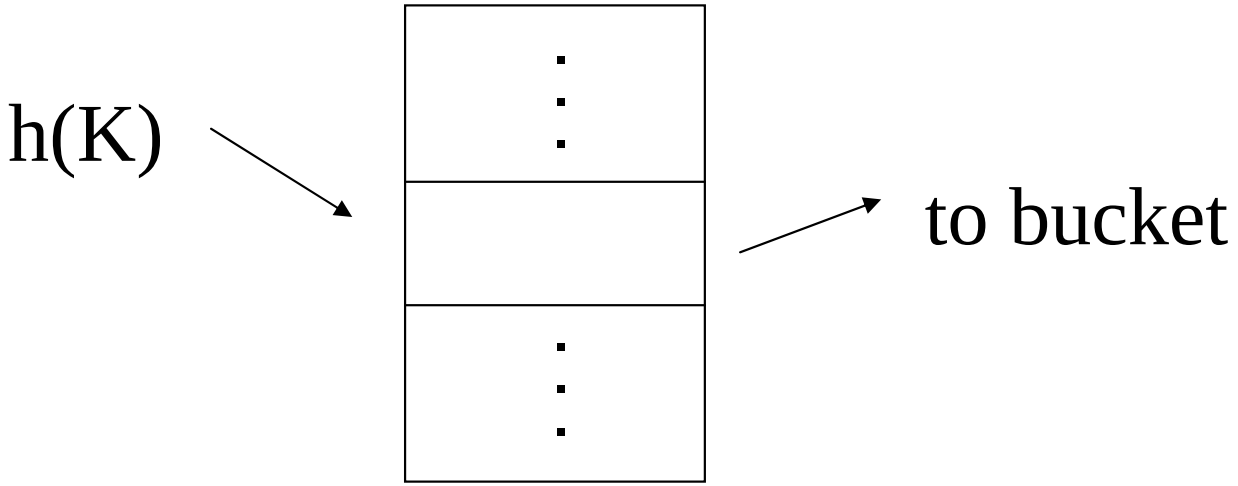
Hash Tables

- So far, only *static hash tables*, i.e. the number B of buckets never changes.
- With growing number of records, space utilization cannot be kept in the desired range.
- *Dynamic hash tables* adapt B to the actual number of records stored.
- Goal: approximately one block per bucket.
- Two dynamic methods:
  - Extensible Hashing, and
  - Linear Hashing.

Extensible Hash Tables

Introduction

- Add a level of indirection for the buckets, a *directory* containing pointers to blocks, one for each value of the hash function.

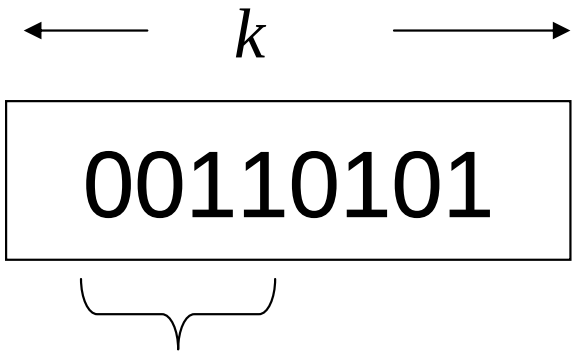


- Size of directory doubles in each growth step.

Extensible Hash Tables

Introduction

- Several buckets can share a data block, if they do not contain too many records.
- Hash function computes sequences of k bits, but bucket numbers use only the i first of these bits. i is the *level* of the hash table.



$$\text{size of directory} = 2^i,$$

$$\text{initially } i = 0$$

Extensible Hash Tables

Insertions

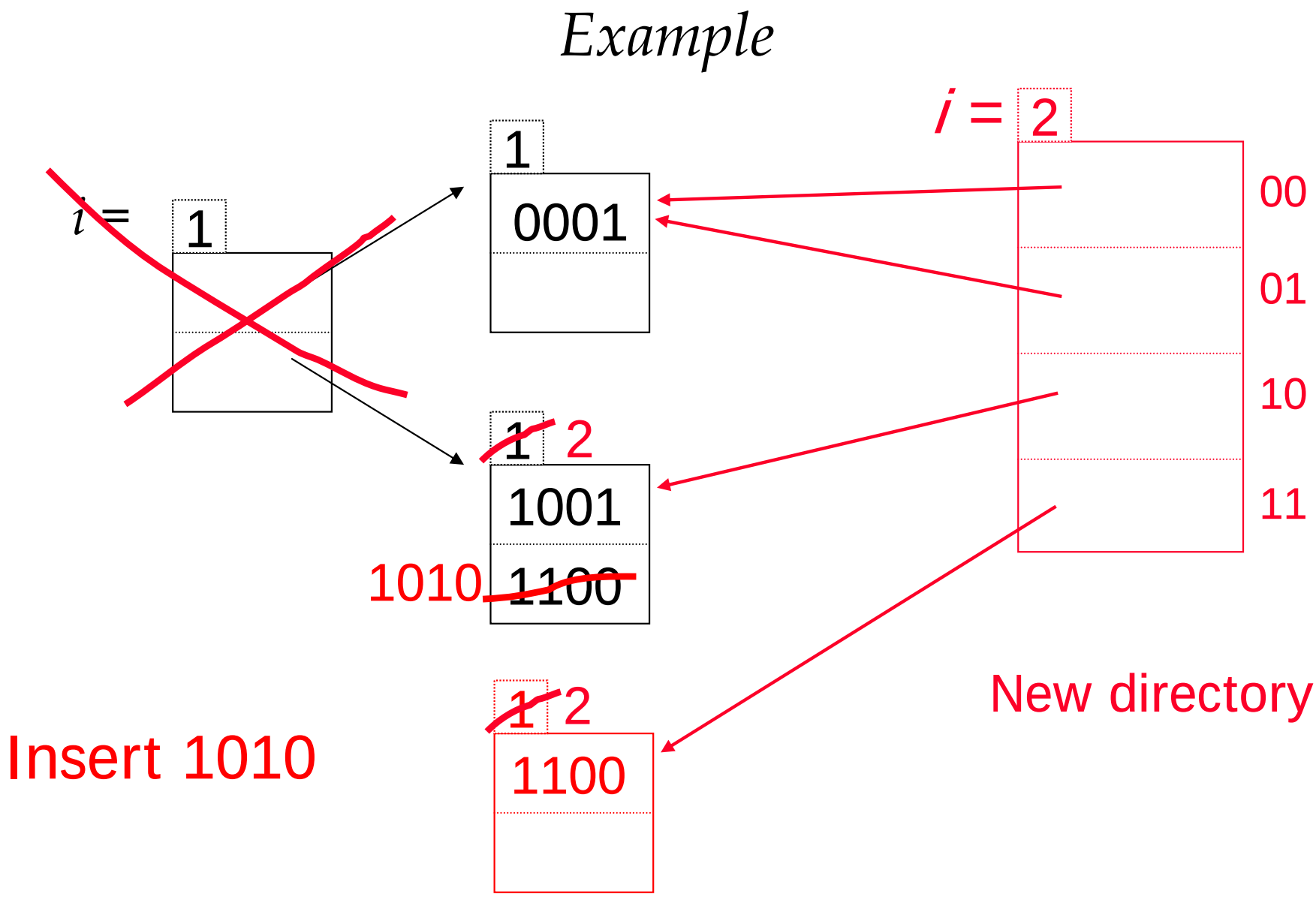
- To insert record with search key K.
- Compute $h(K)$ and take its first i bits. *Global level* i is part of the data structure.
- Retrieve the corresponding directory entry.
- Follow that pointer leading to block b. b has a *local level* $j <= i$.
- If b has enough space, insert record there.
- Otherwise, split b into two blocks.

Extensible Hash Tables

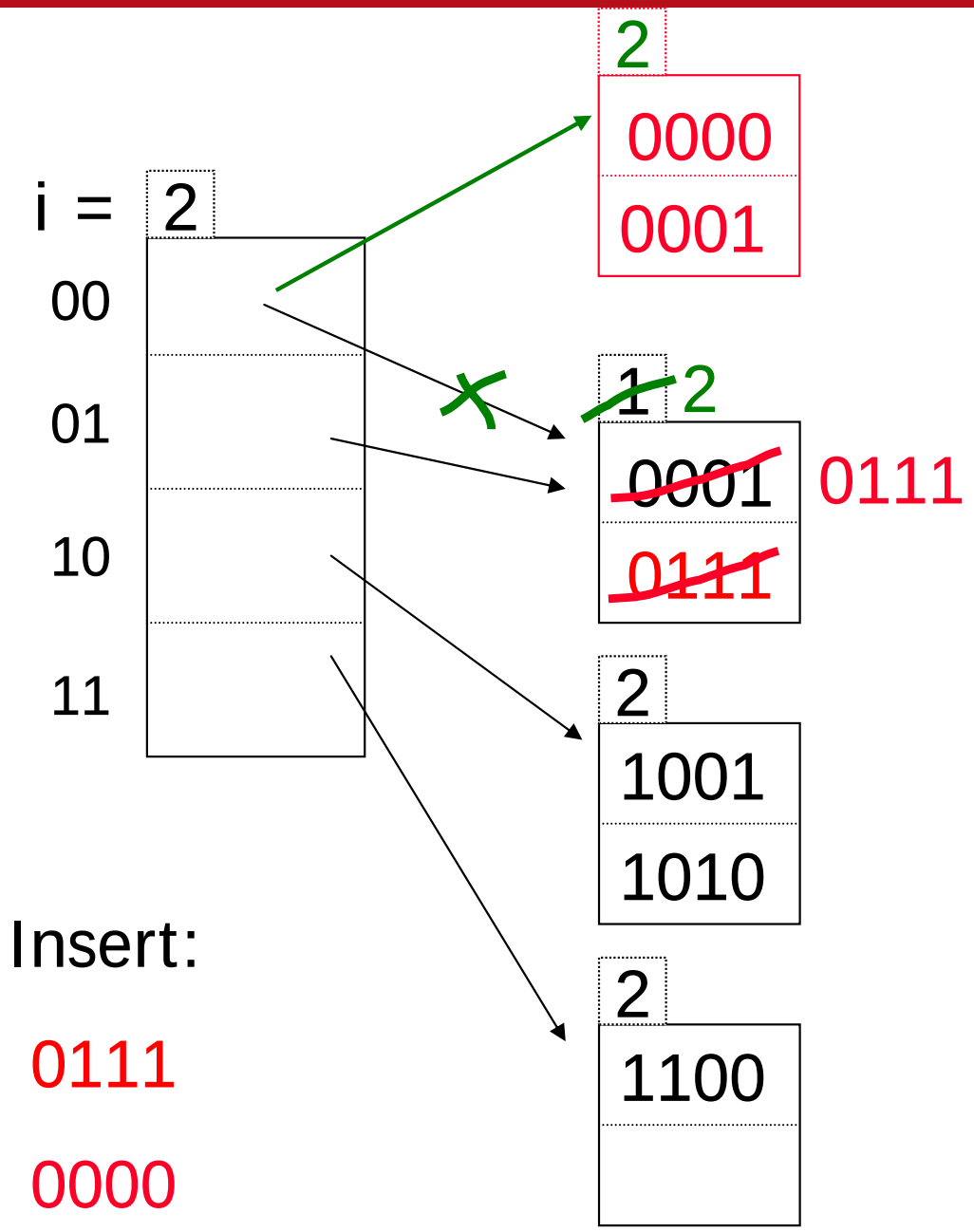
Insertions

- If $j < i$, distribute records in b based on $(j+1)$st bit of $h(K)$: if 0, old block b, if 1 new block b'.
- Increment the local level of b and b' by one.
- Adjust the pointer in the directory that pointed to b but must now point to b'.
- If $j = i$, first increment i by one. Double the directory size and duplicate all entries. Proceed as in case $j < i$.

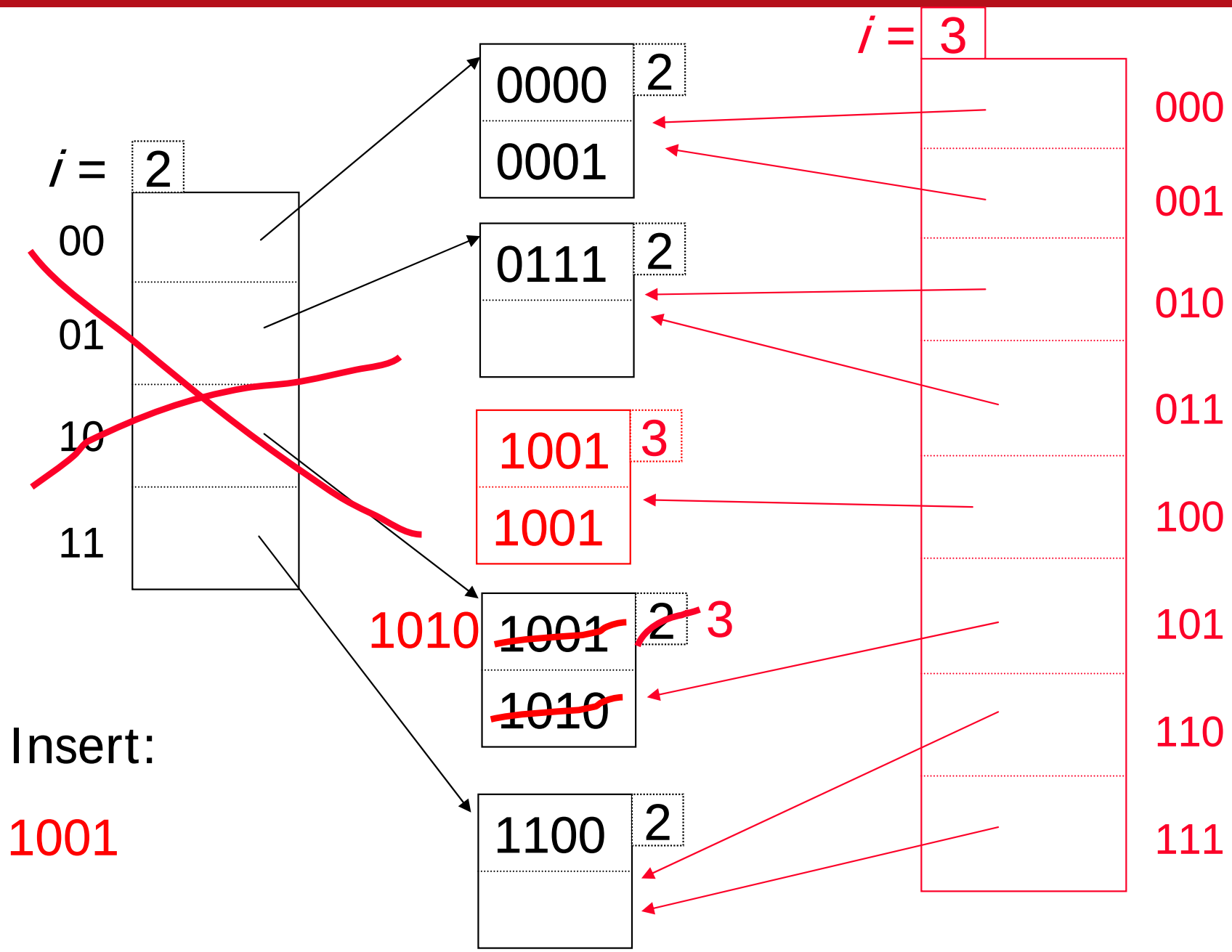
Extensible Hash Tables



Extensible Hash Tables



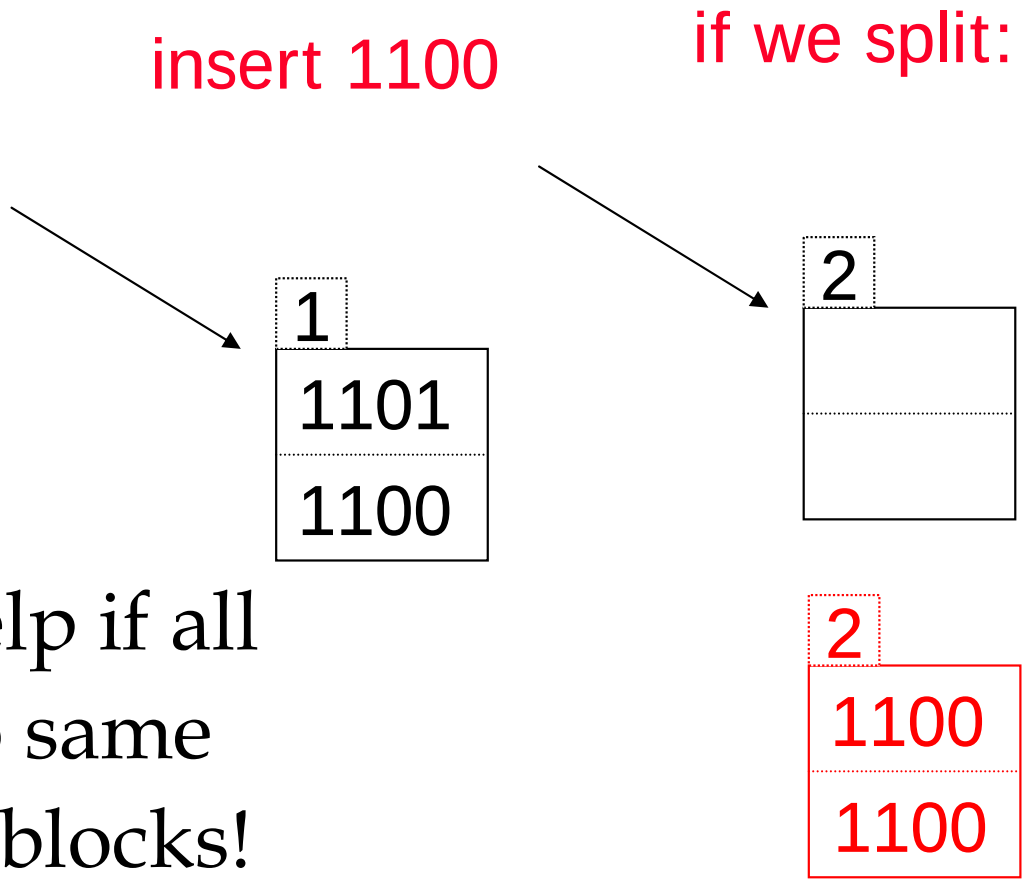
Extensible Hash Tables



Extensible Hash Tables

Overflow Chains

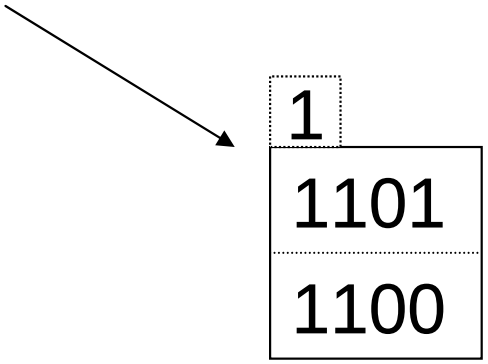
- May still need overflow chains in the presence of too many duplicates of hash values.
- Split does not help if all entries belong to same of two resulting blocks!



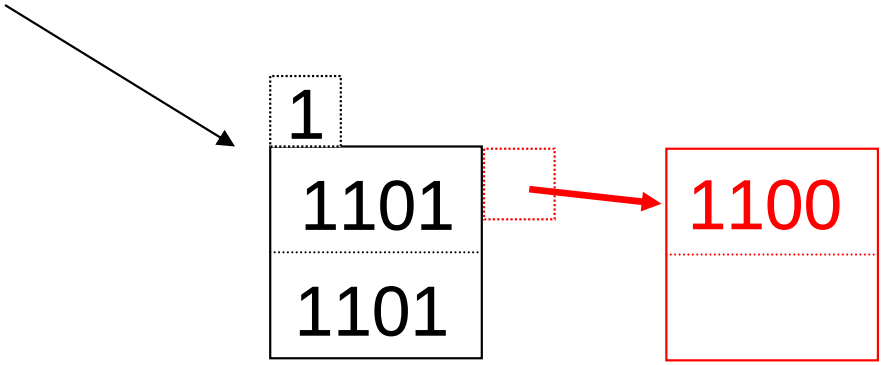
Extensible Hash Tables

Overflow Chains

insert 1100



add overflow block:



Extensible Hash Tables

Deletions

- To delete record with search key K.
- Using the directory, locate corresponding block b and delete record from there.
- If possible, merge block b with "buddy" block b' and adjust the directory pointers to b and b'.
- If possible, halve the directory.
  → reverse insertion procedure

Extensible Hash Tables

Discussion

- Can manage growing number of buckets without wasting too much space.
- Assume that directory fits into main memory.
- Never need to access more than one data block (as long as there are no overflow chains) for a query.
- Doubling the directory is a very expensive operation. Interrupts other operations and may require secondary storage.