UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

CS411 - Indexing

illinois.edu

Announcements

- MP2 due Friday
- Track 1 initial demos Thursday, Friday
  - After this, work on your project
- HW3 assigned the break

Announcements

- Friday's lecture will be short
  - I'll cover "advanced" indexing topics
    - Multidimensional indexing
    - Large string indexing
    - Web search indexing

Review

- Why index a relation?
- What is a *search key*?
- What is a *clustered index*?
- What is a *sparse index*?

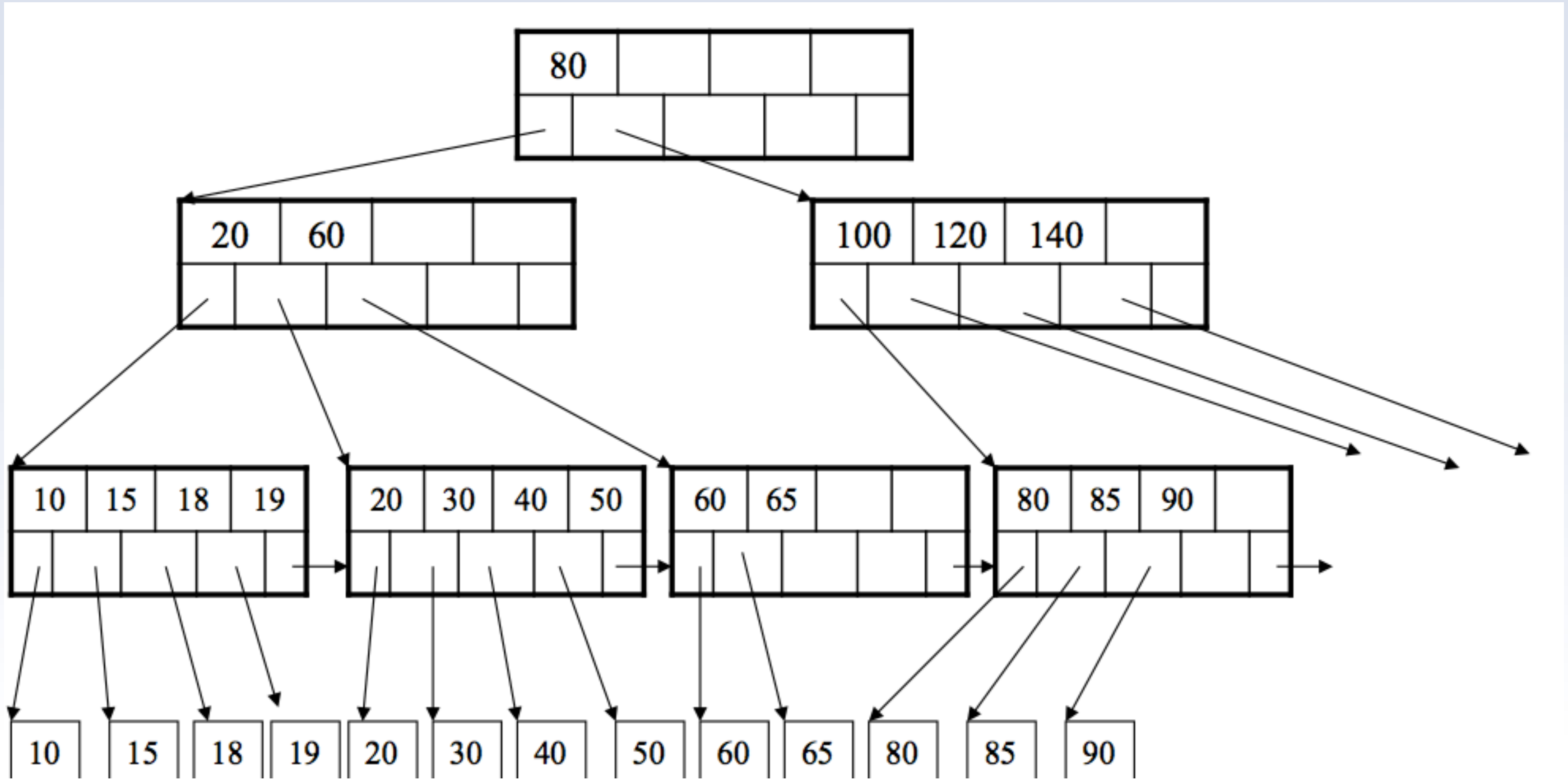
Review

- What is a *secondary index*?
- What is a *sequential file*?
- What is a *multilevel index*?
- What is *bucket intersection*?

Review

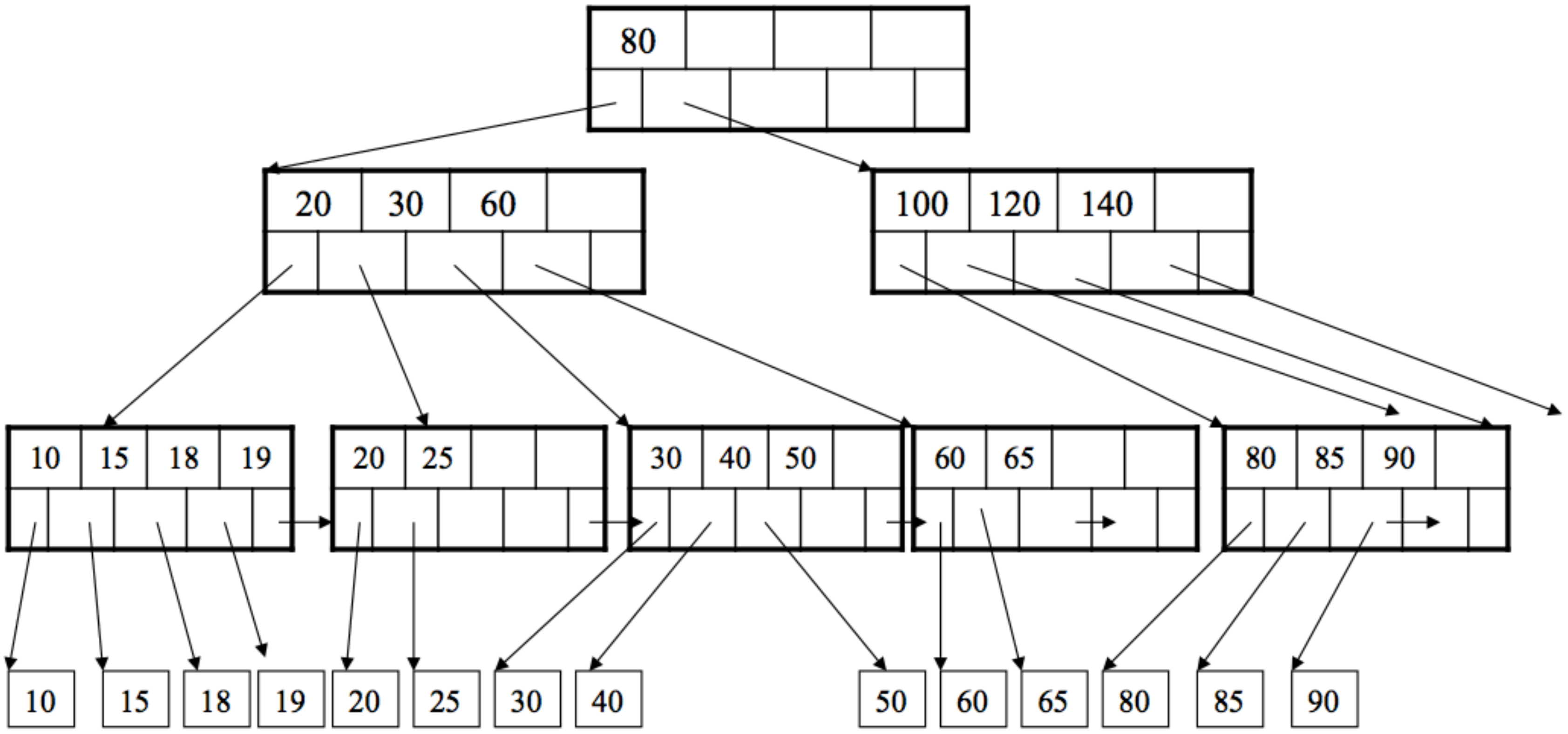
- What are the advantages of a B-tree index?
- What does the parameter *n* of a B-tree index represent?
- Why do the leaf nodes of a B-tree have "next" pointers?

Review: Insert 25



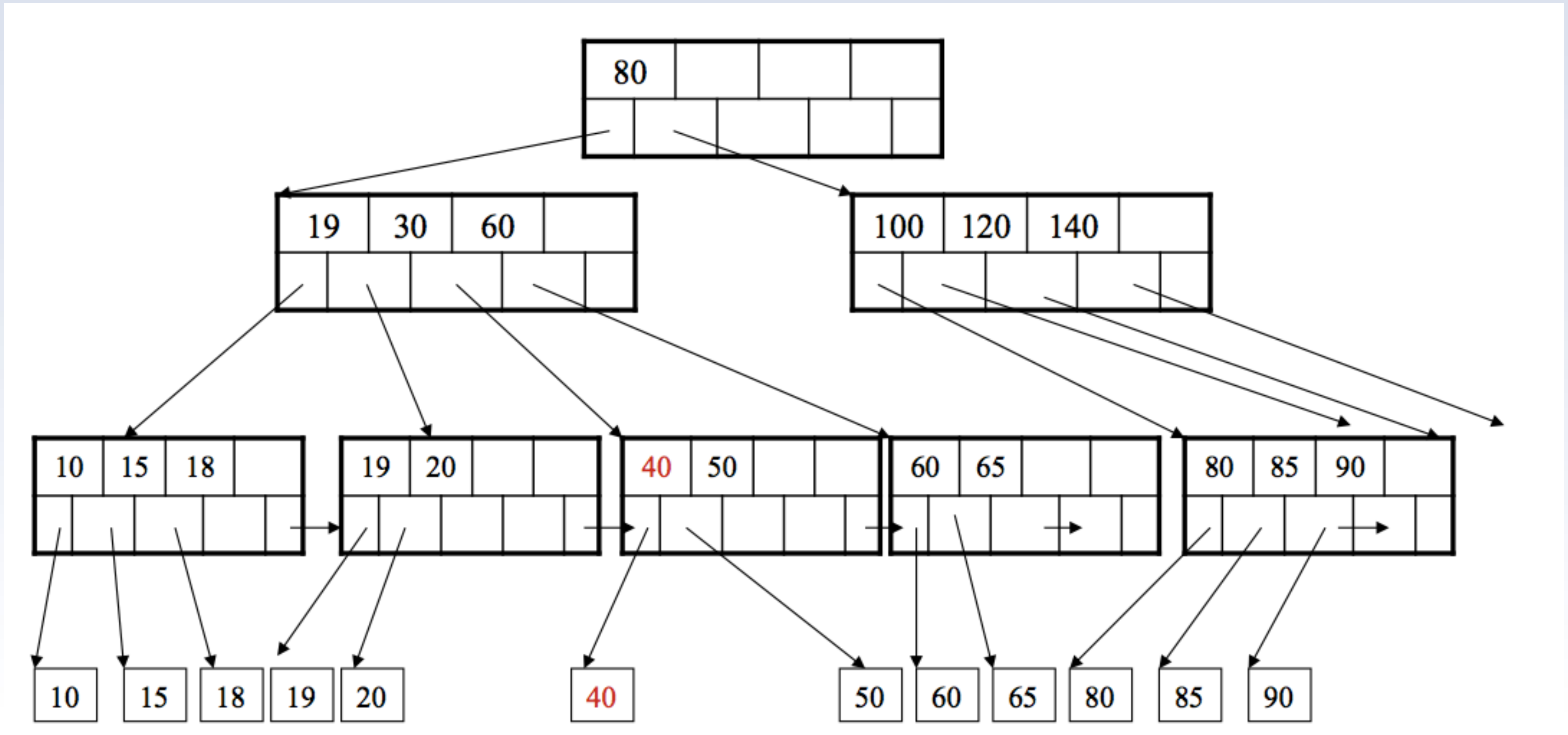
illinois.edu

Solution

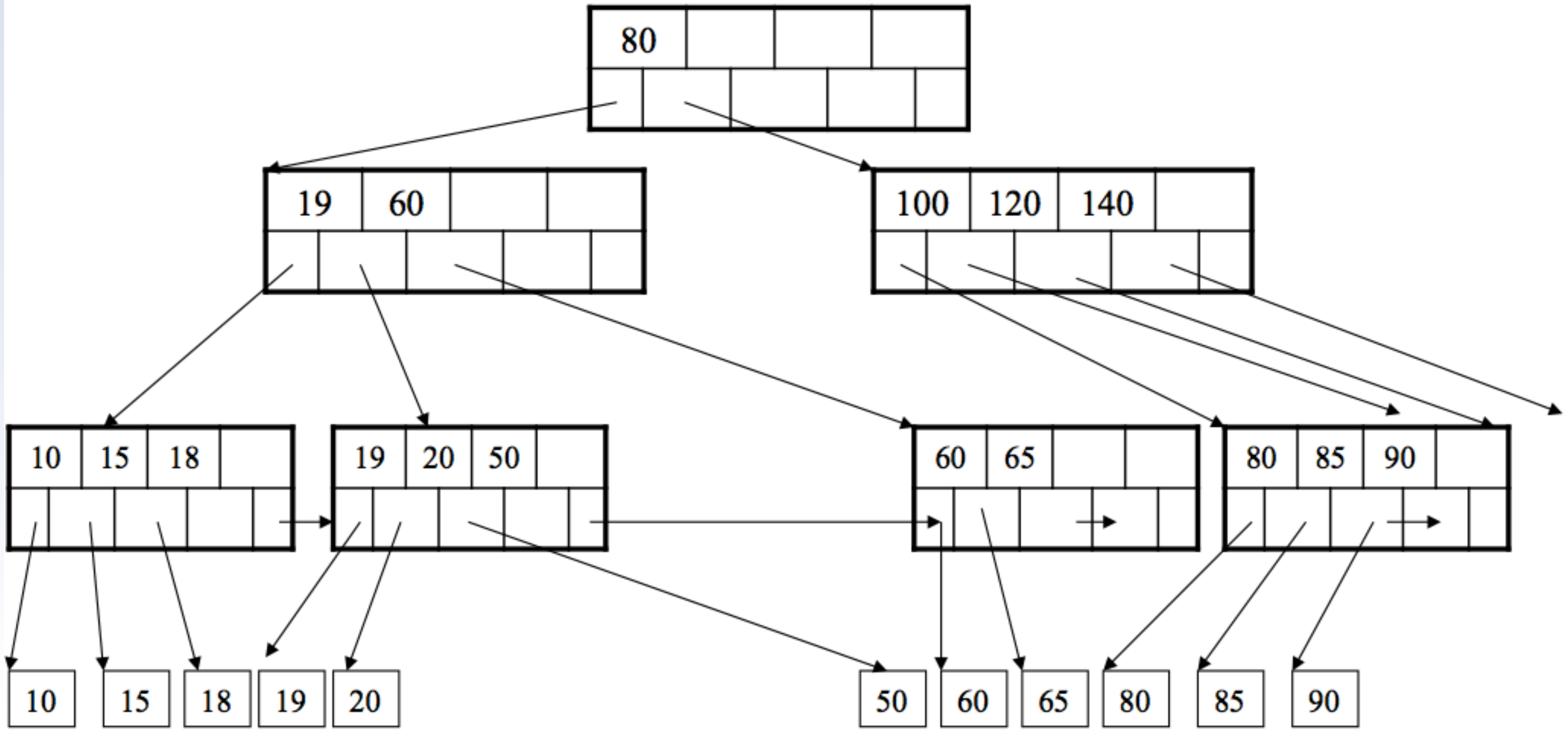


illinois.edu

Review: Delete 40



Solution



illinois.edu

Indexing

- Sparse indexes
- Dense indexes
- B-trees
- Hash tables

Hash Tables

- Basics:
  - n *buckets* used to store keys
  - *hash function* maps keys to buckets

Hash functions

- Maps the set of search keys to the set of buckets
  - $f: K \to \{0,1,\dots,n-1\}$
  - K is the set of all keys
  - n is the number of buckets

Example

- Given 5 buckets, let f(k)=k%5
  - modulo (remainder after dividing by 5)
- Insert 100, 12, 59, 3, 33

Bucket	Keys
0	
1	
2	
3	
4	

Example

- Given 5 buckets, let f(k)=k%5
  - modulo (remainder after dividing by 5)
- Insert 100, 12, 59, 3, 33

100%5=0

Bucket	Keys
0	100
1	
2	
3	
4	

illinois.edu

Example

- Given 5 buckets, let f(k)=k%5
  - modulo (remainder after dividing by 5)
- Insert 100, 12, 59, 3, 33

12%5=2

Bucket	Keys
0	100
1	
2	12
3	
4	

Example

- Given 5 buckets, let f(k)=k%5
  - modulo (remainder after dividing by 5)
- Insert 100, 12, 59, 3, 33

59%5=4

Bucket	Keys
0	100
1	
2	12
3	
4	59

illinois.edu

Example

- Given 5 buckets, let f(k)=k%5
  - modulo (remainder after dividing by 5)
- Insert 100, 12, 59, 3, 33

3%5=3

Bucket	Keys
0	100
1	
2	12
3	3
4	59

Example

- Given 5 buckets, let f(k)=k%5
  - modulo (remainder after dividing by 5)
- Insert 100, 12, 59, 3, 33

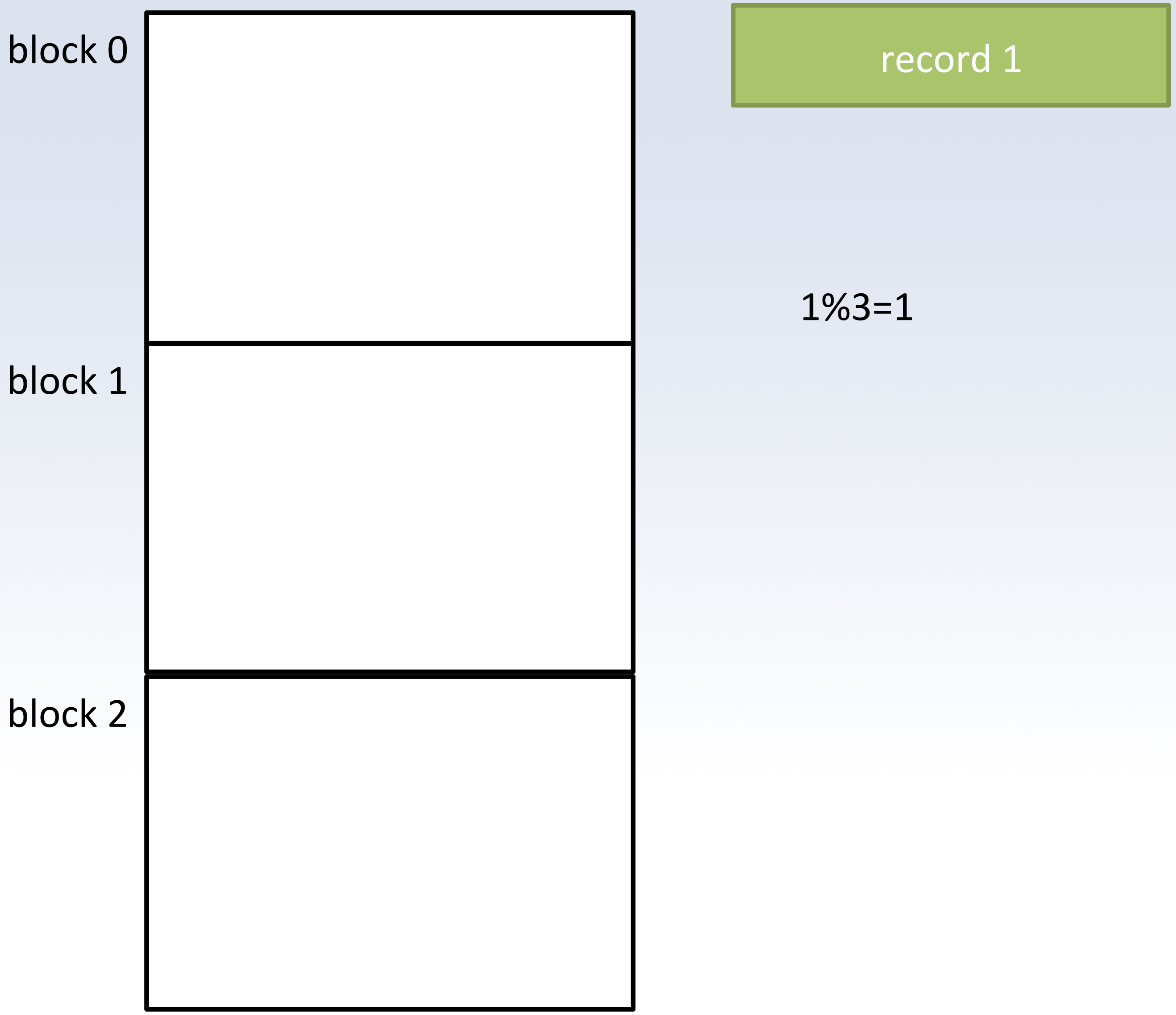
33%5=3

Bucket	Keys
0	100
1	
2	12
3	3, 33
4	59

DBMS Hash Tables

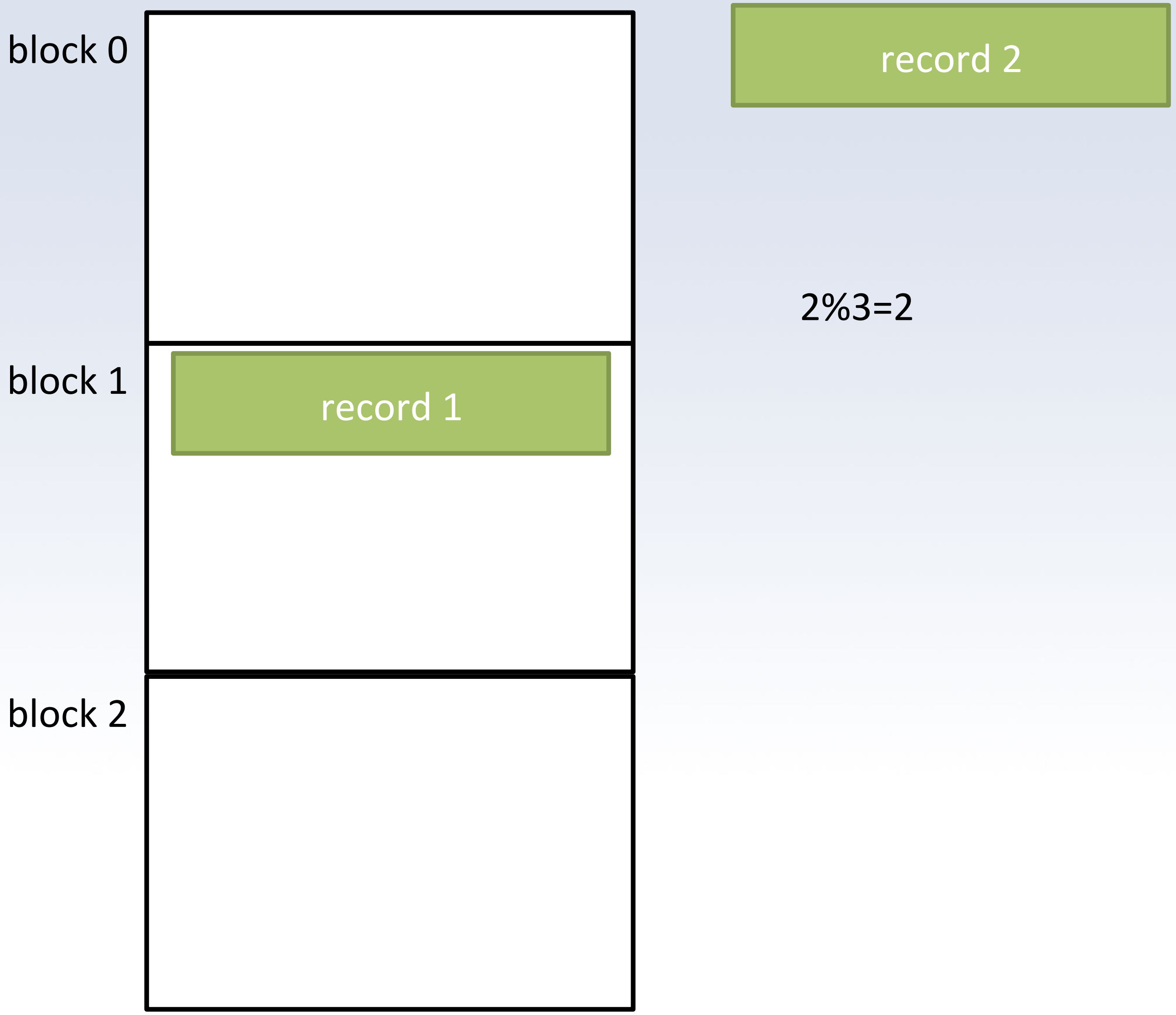
- Buckets consist of blocks
- Blocks contain records
- Overflow blocks added as needed
  – Pointers to overflow blocks go in the header

Example

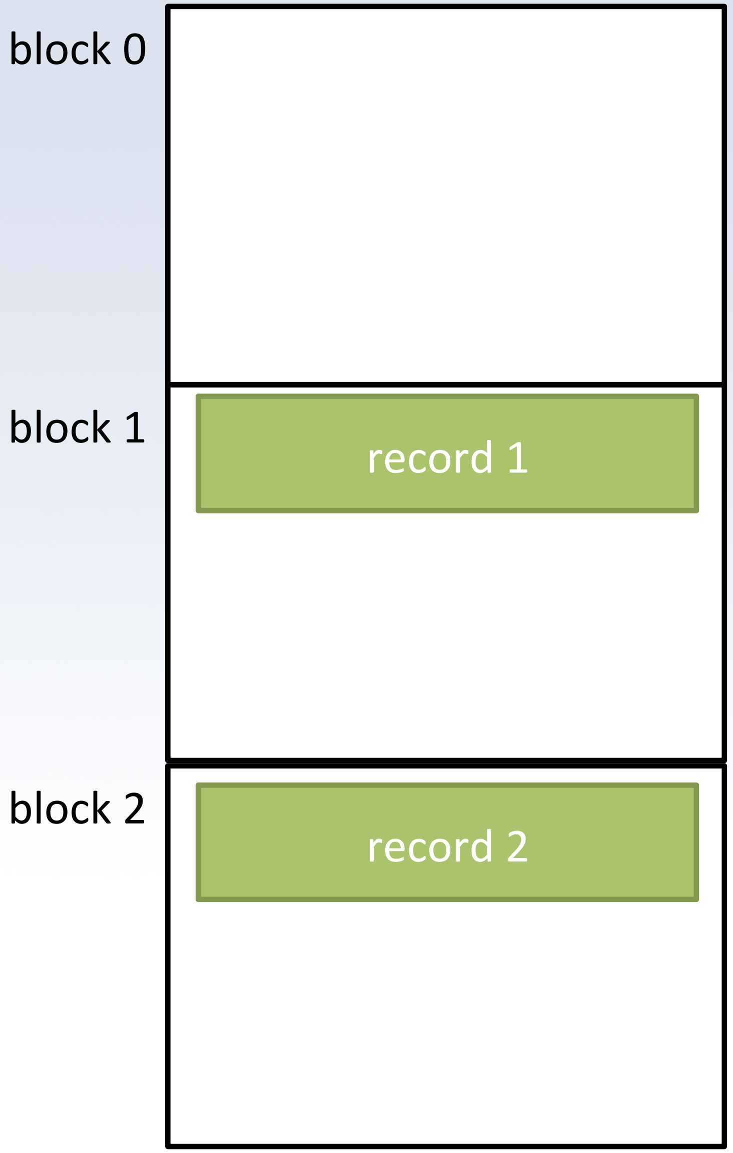


illinois.edu

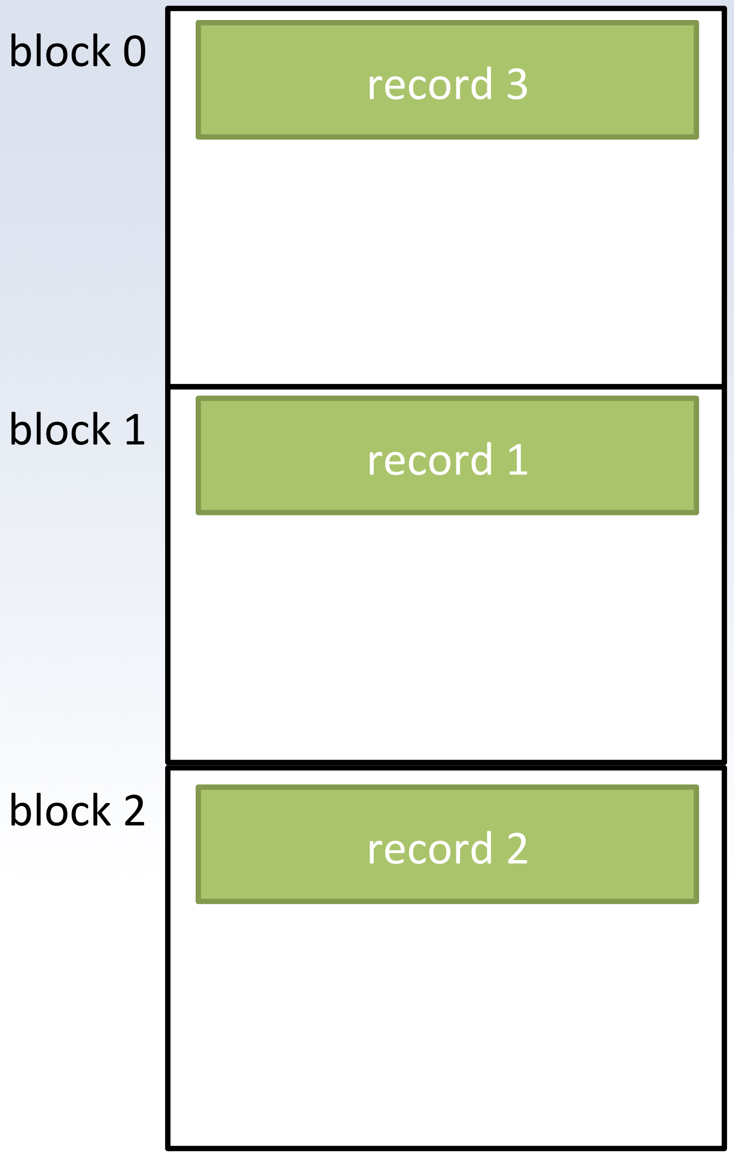
Example



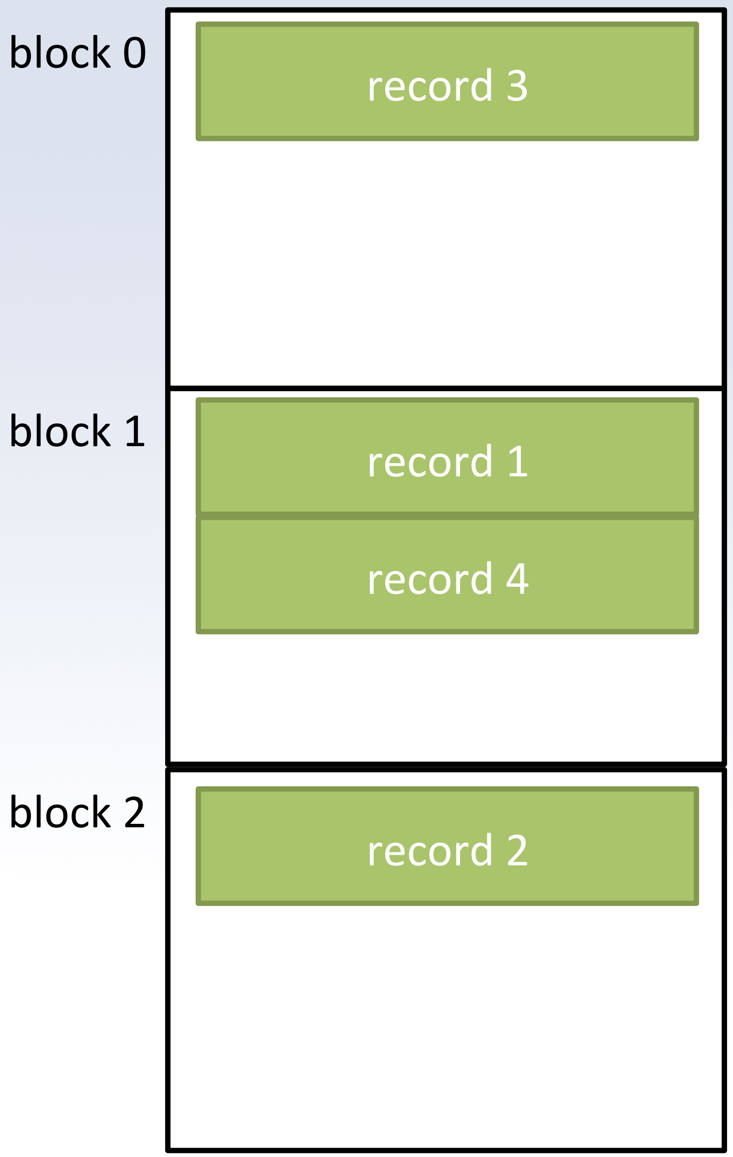
Example



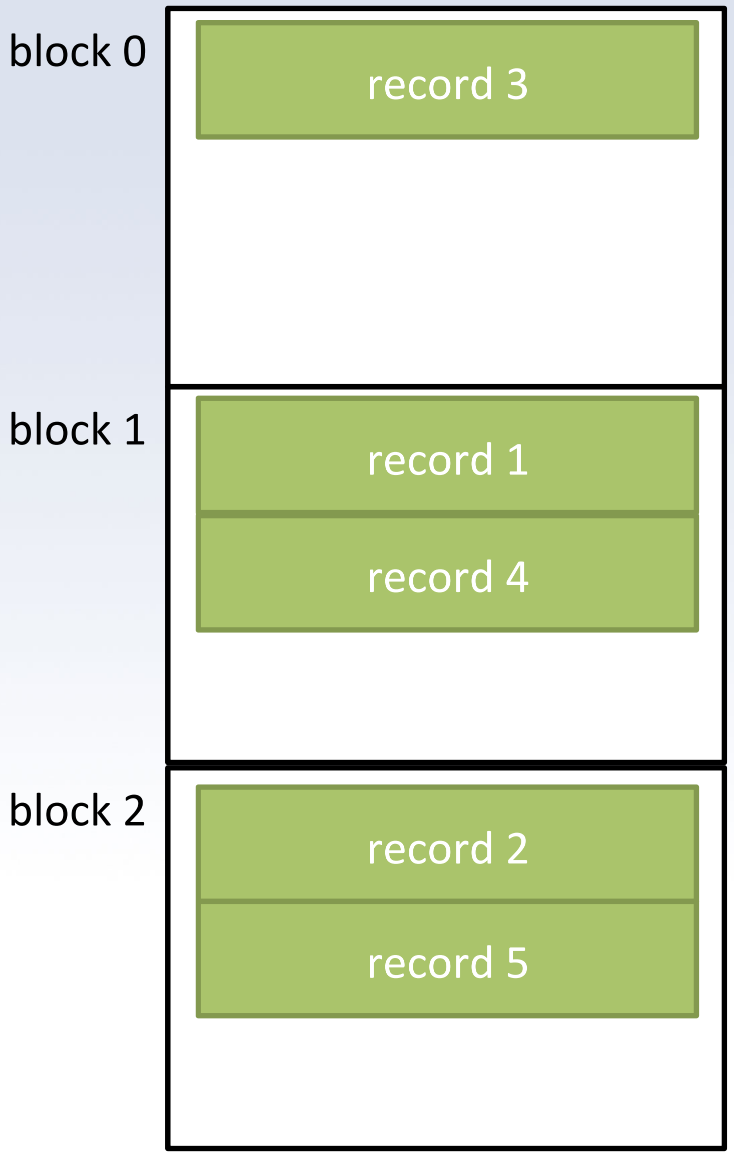
Example



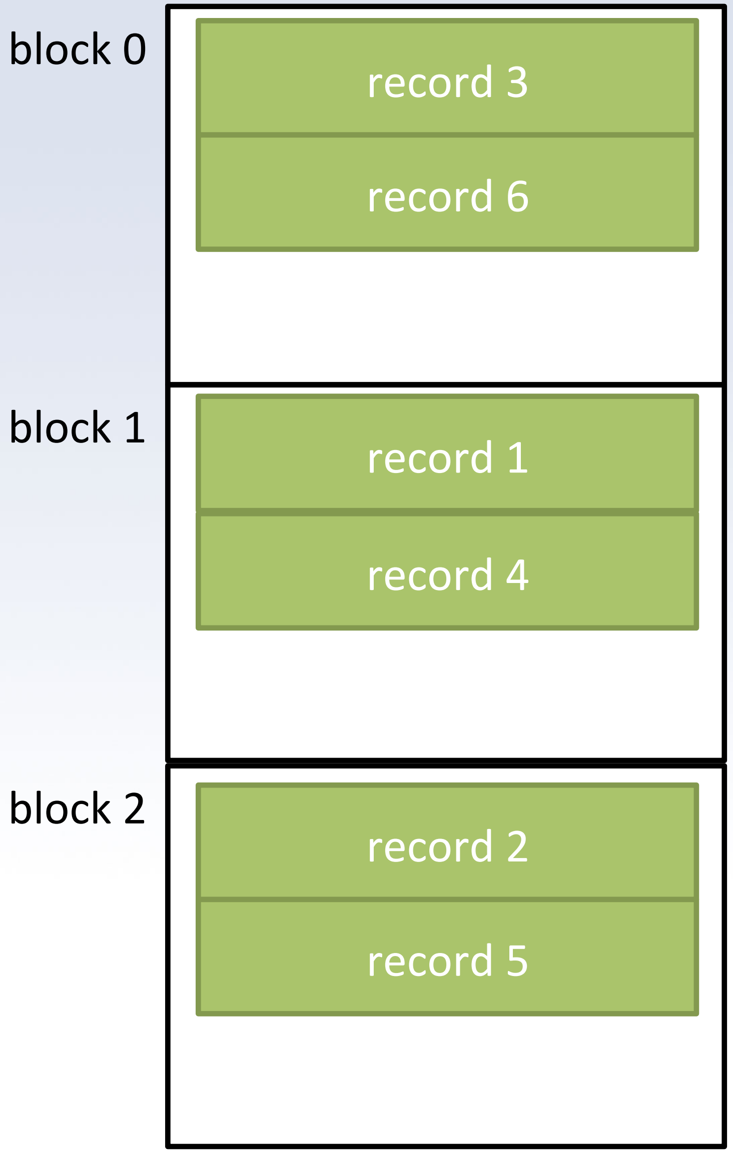
Example



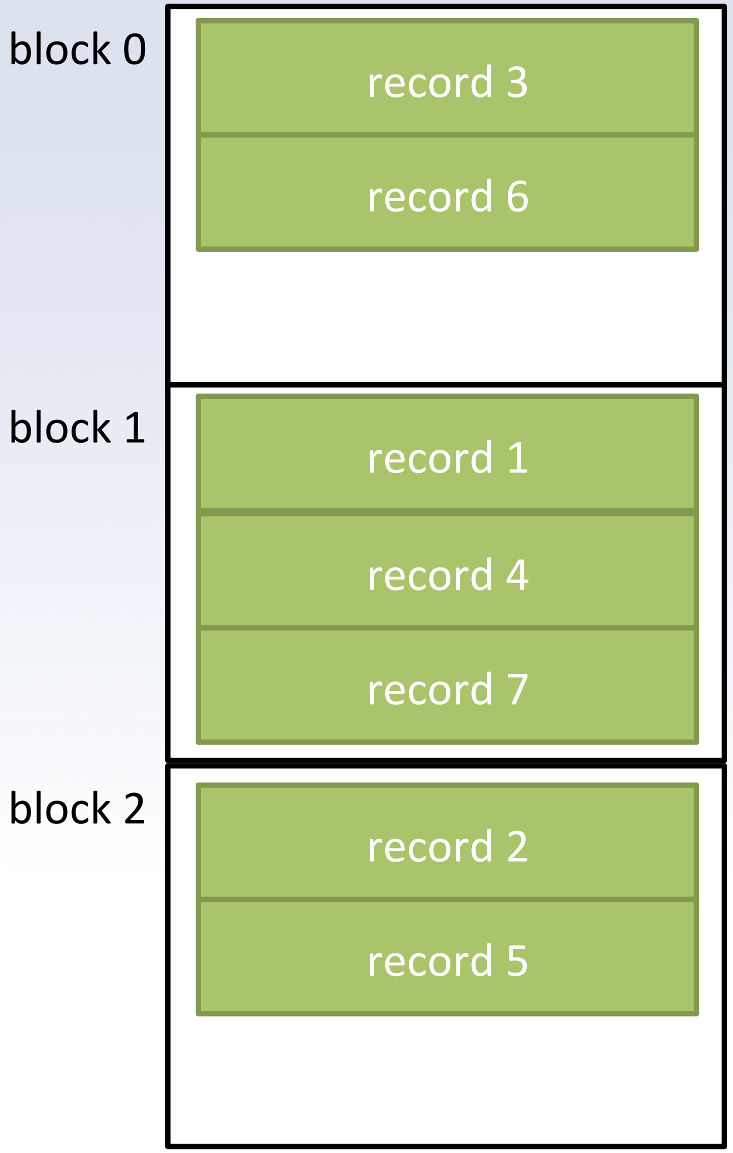
Example


block 0
record 3
block 1
record 1
record 4
block 2
record 2
record 5

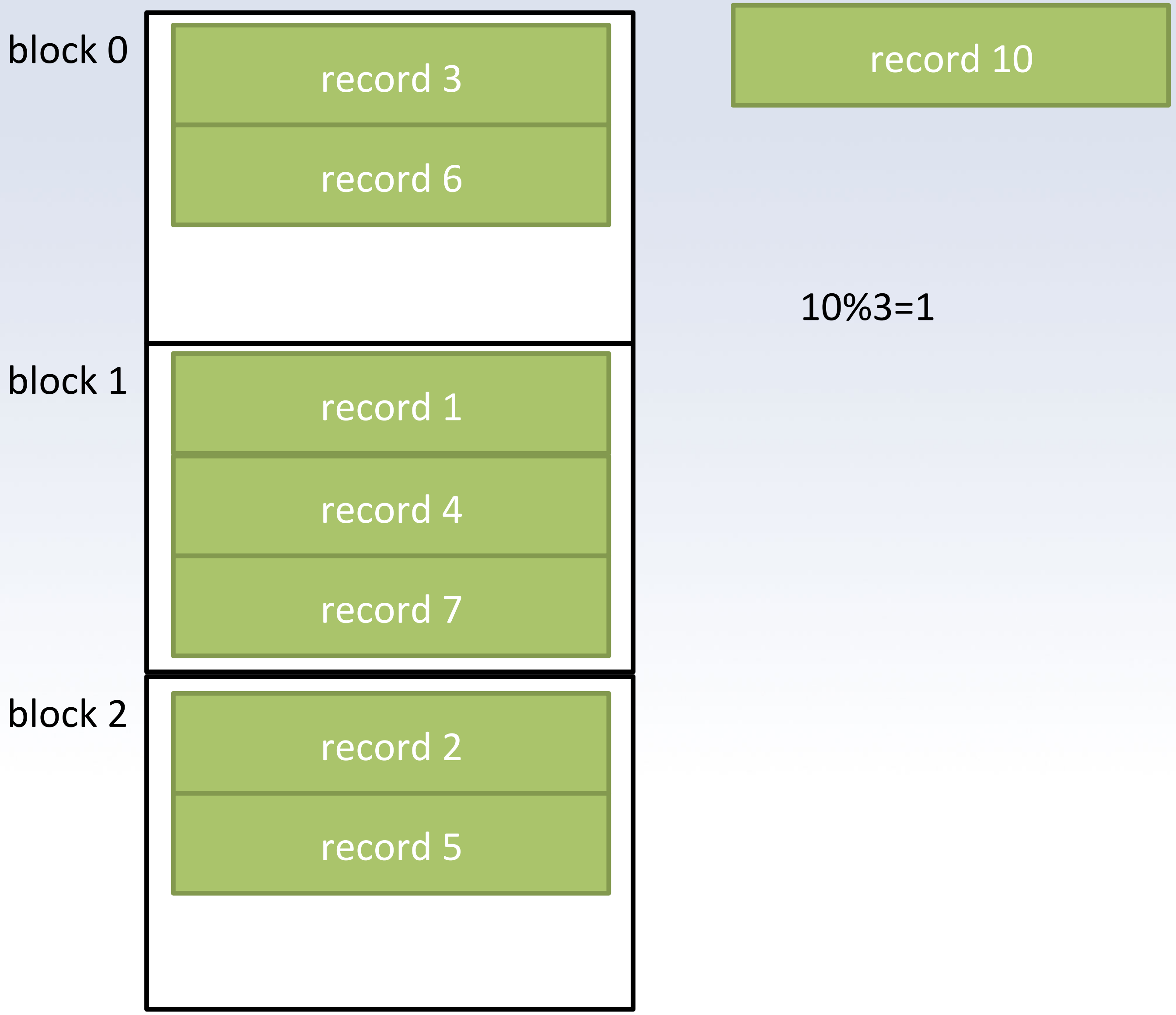
Example


block 0
record 3
record 6
block 1
record 1
record 4
block 2
record 2
record 5

Example

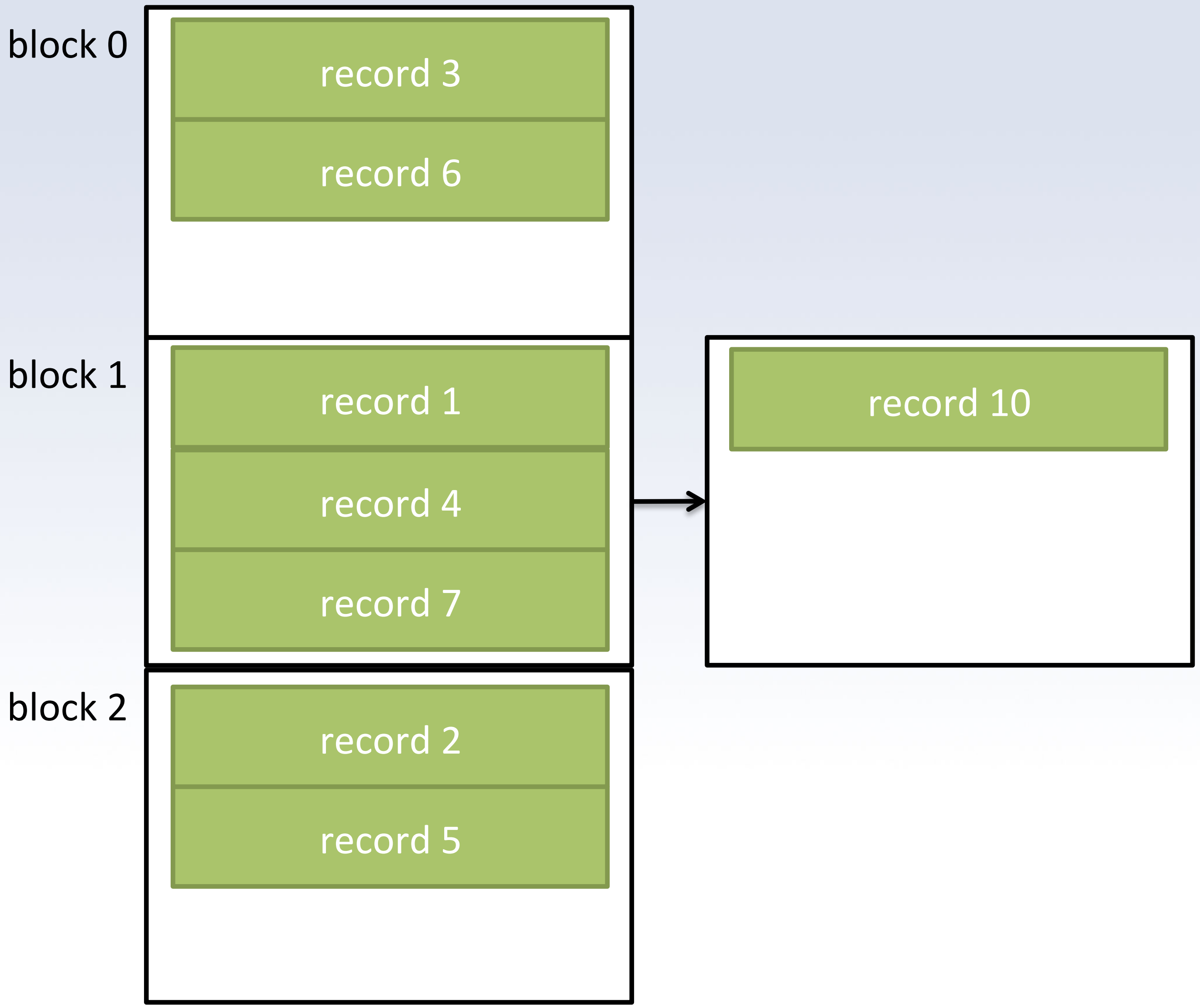

block 0
record 3
record 6
block 1
record 1
record 4
record 7
block 2
record 2
record 5

Example



illinois.edu

Example


block 0
record 3
record 6
block 1
record 1
record 4
record 7
record 10
block 2
record 2
record 5

Hash Table Indexes

- Efficiency depends on blocks per bucket
  - One disk I/O for lookup if one block/bucket
  - Many disk I/Os if lots of overflow buckets
- Can we improve?

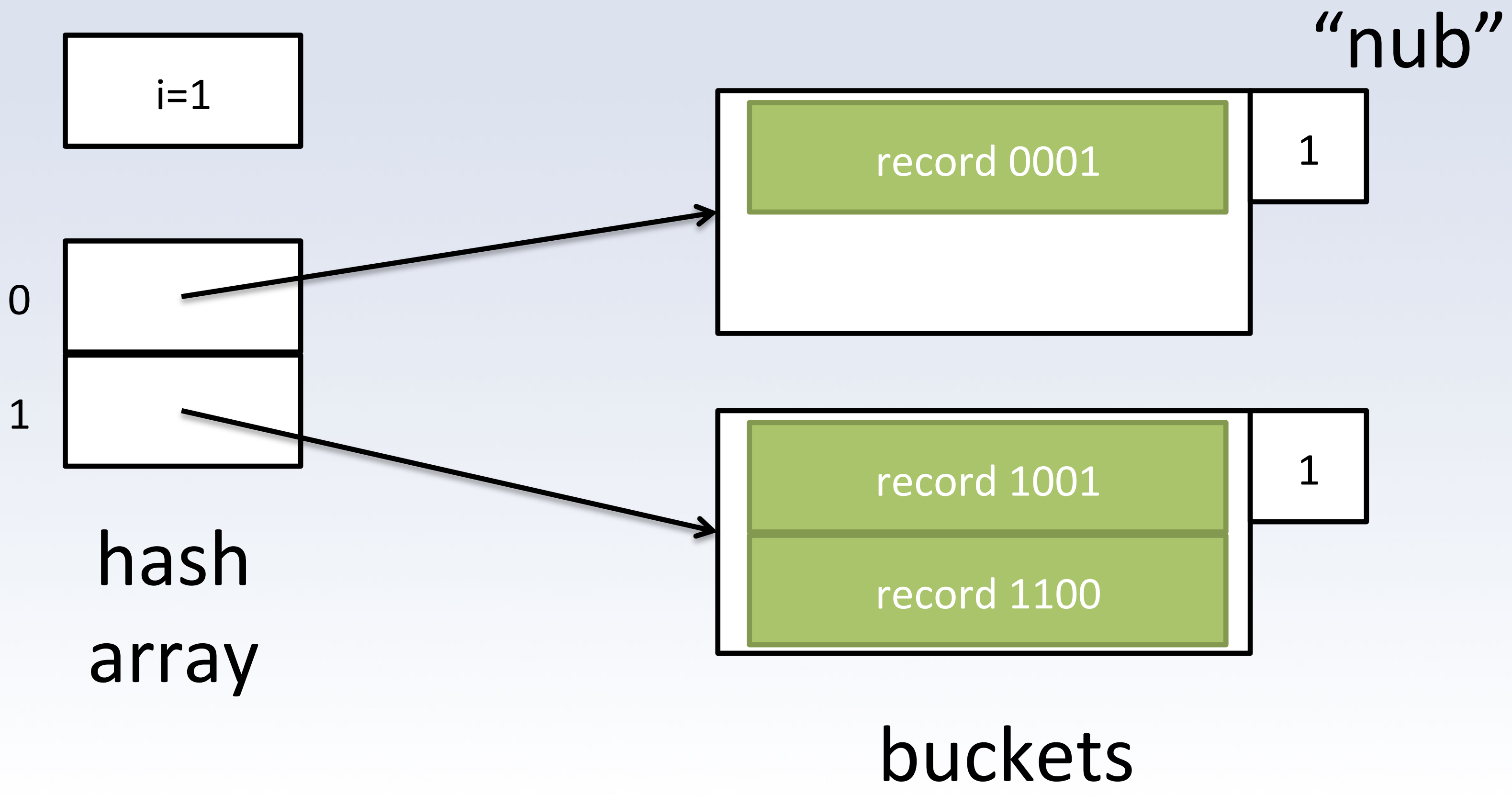
Dynamic Hash Tables

- Allow number of buckets to vary
  - Try to keep number of blocks per bucket low
- Two methods
  - Extensible hashing
  - Linear hashing

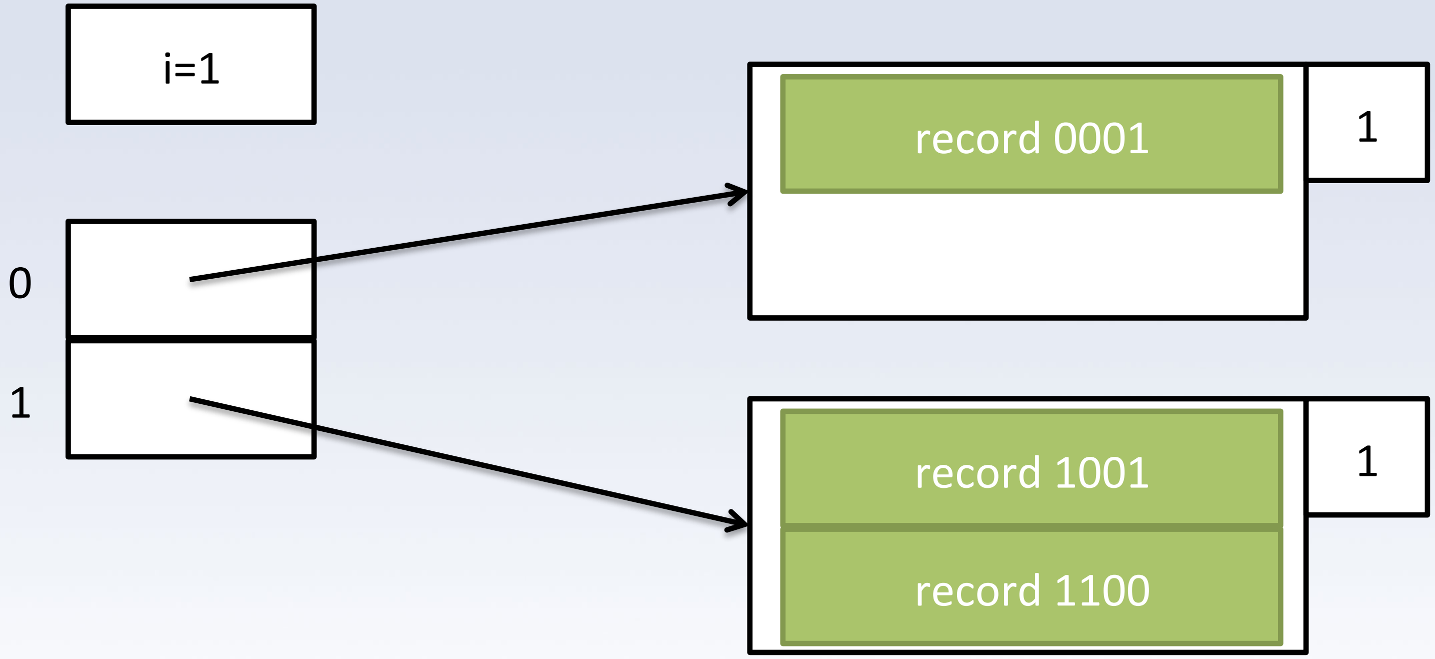
Extensible Hash Tables

- Strategy: double the number of buckets when needed
  - keep counter i
  - hash size = 2^i
  - keep *hash array* = pointers to buckets
  - hash array pointers can point to same bucket

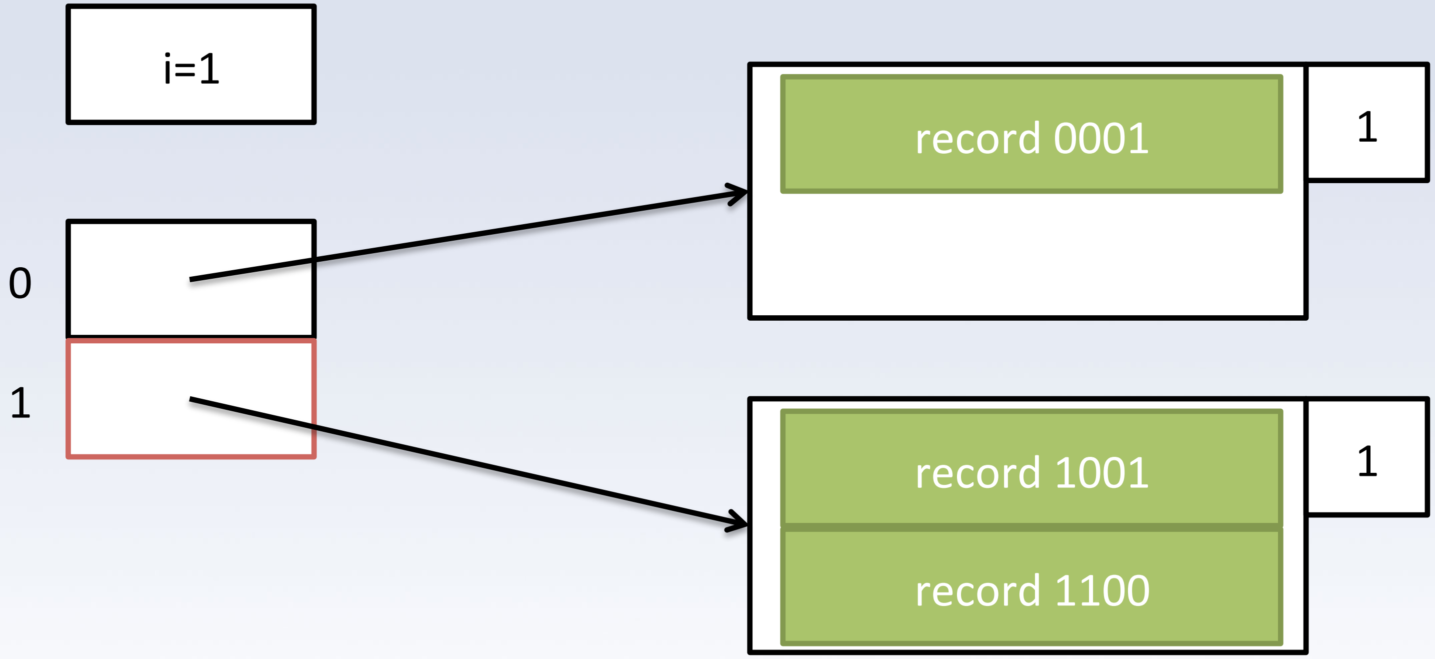
Example


i=1
0
1
hash array
"nub"
record 0001
1
record 1001
record 1100
1
buckets

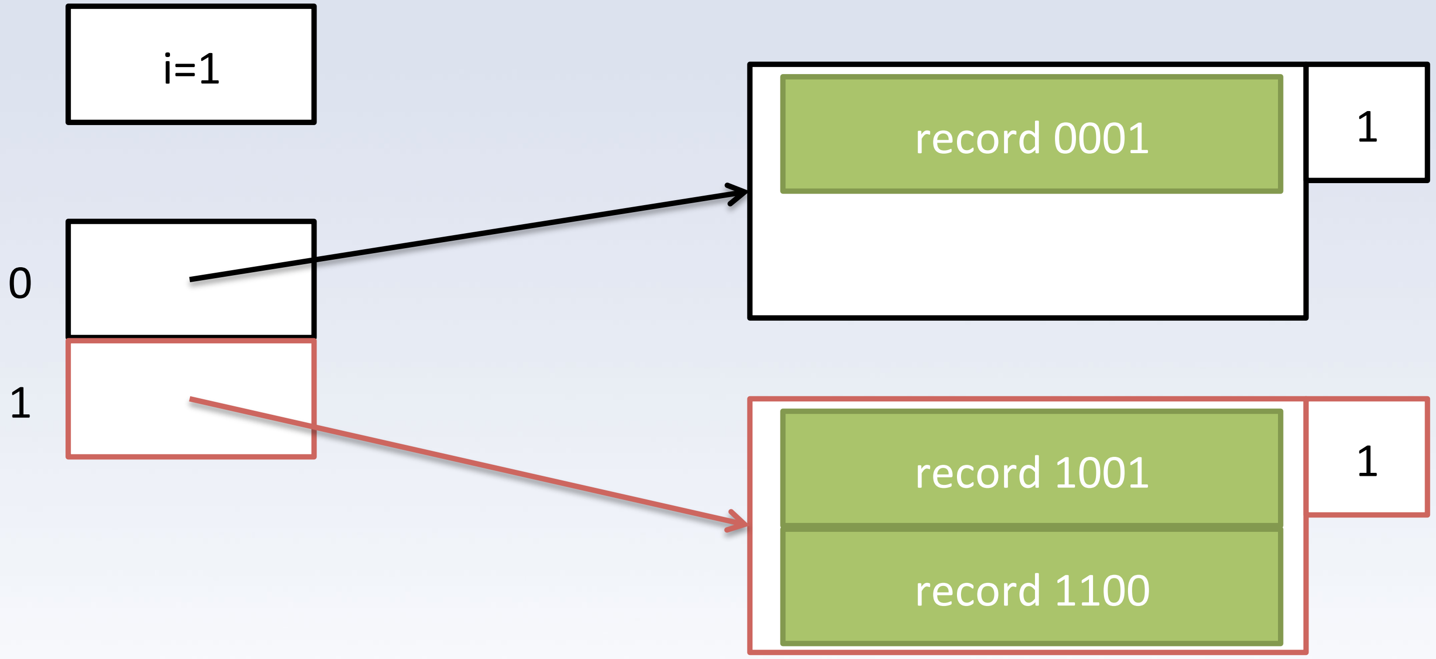
Example: Look up 1001



Example: Look up 1001



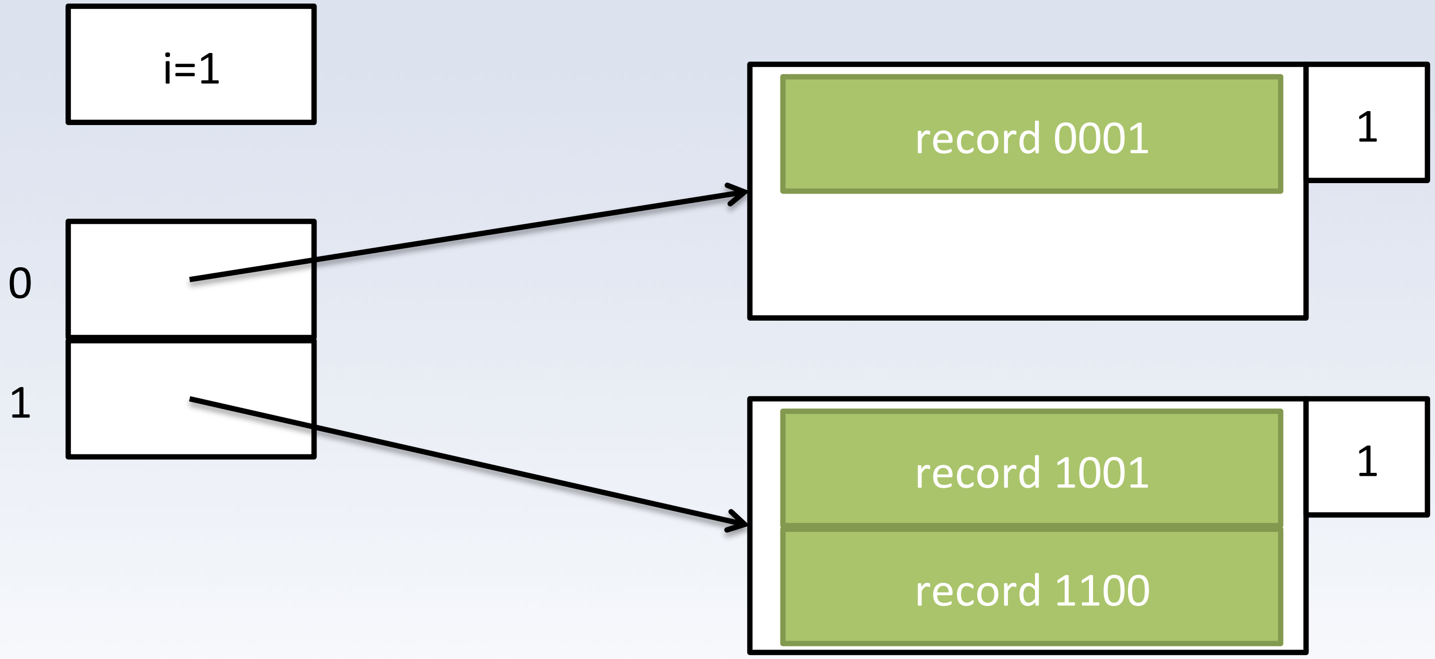
Example: Look up 1001



Extensible Hash Tables

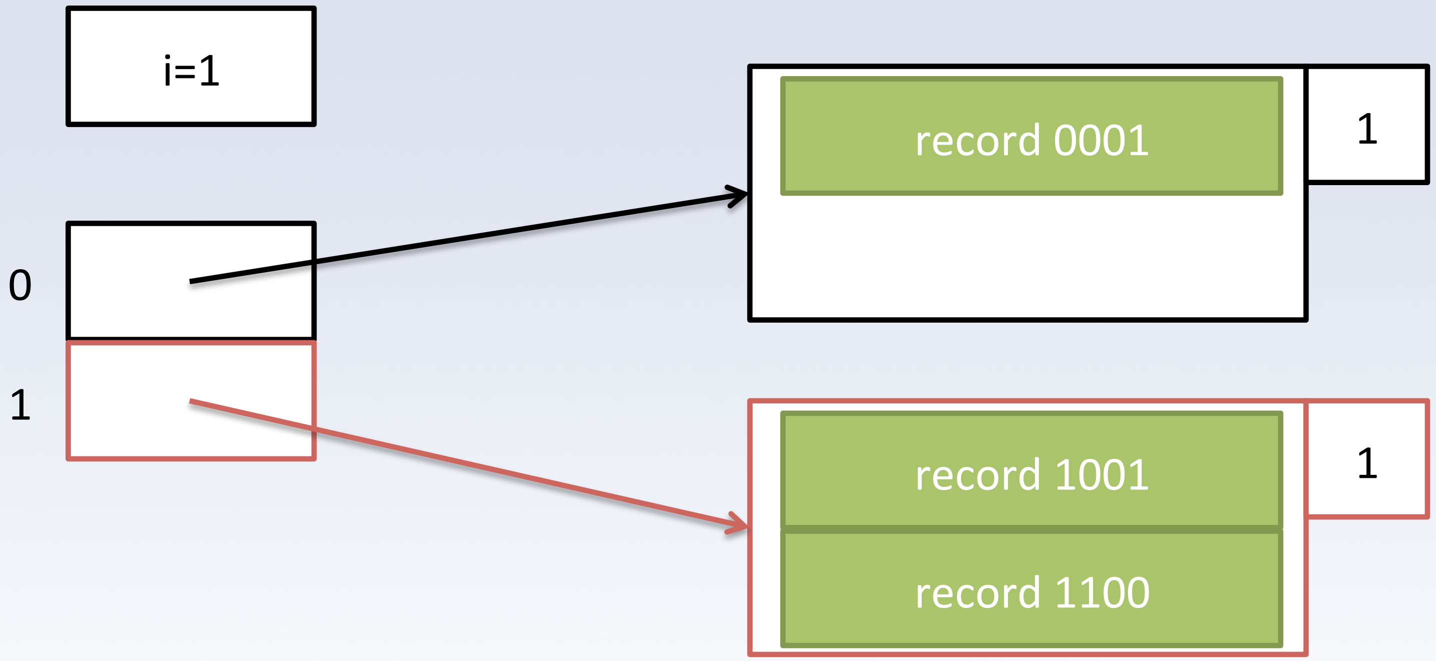
- Insertion
  1. Lookup new key. If room, insert.
  2. No room:
     a. If "nub" value = i, increment i. Doubles hash array (2^{i+1} entries.)
     b. Split block. New blocks have incremented "nub" values.
     c. Distribute original block into new blocks.
     d. Repeat if necessary (still not enough room.)

Example: Insert 1010

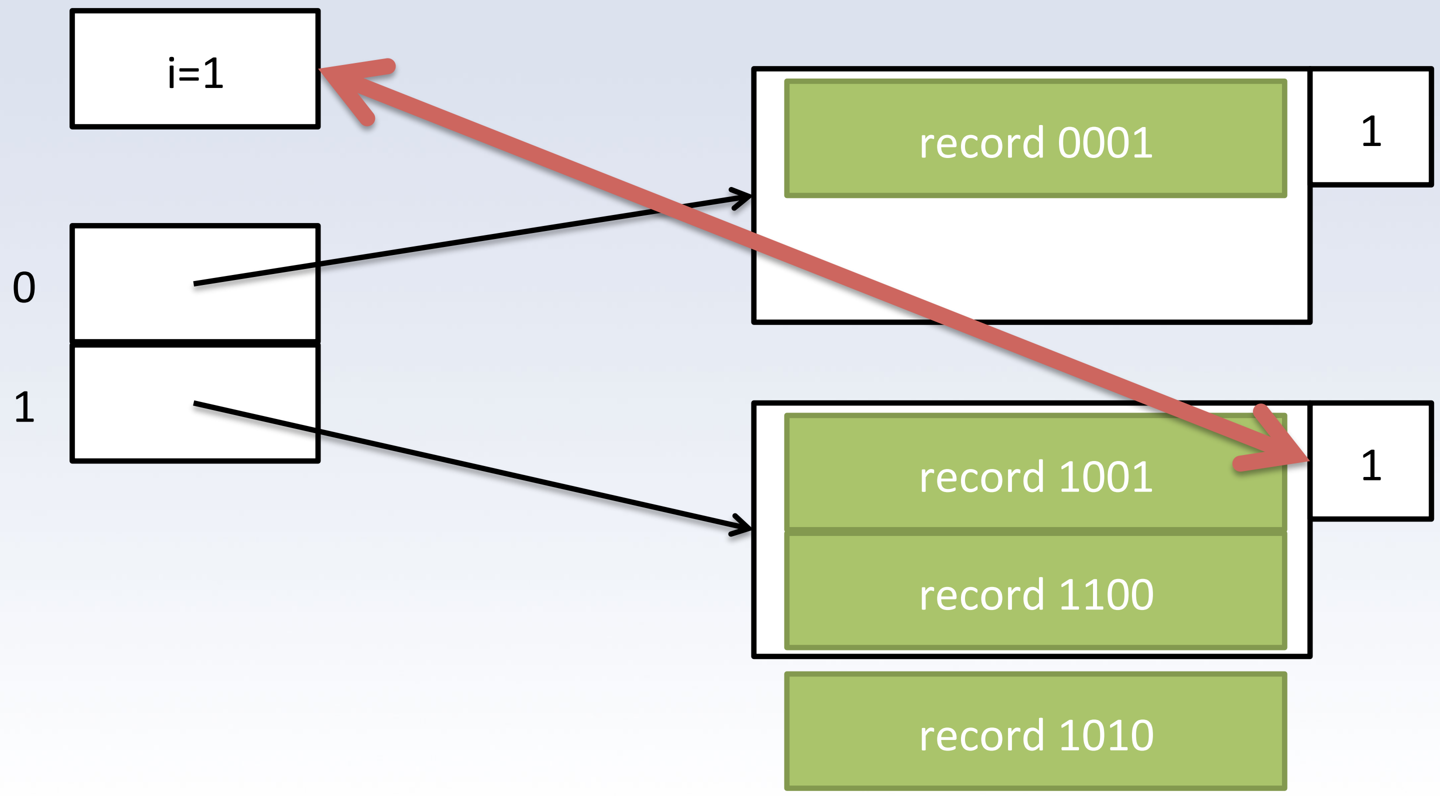


illinois.edu

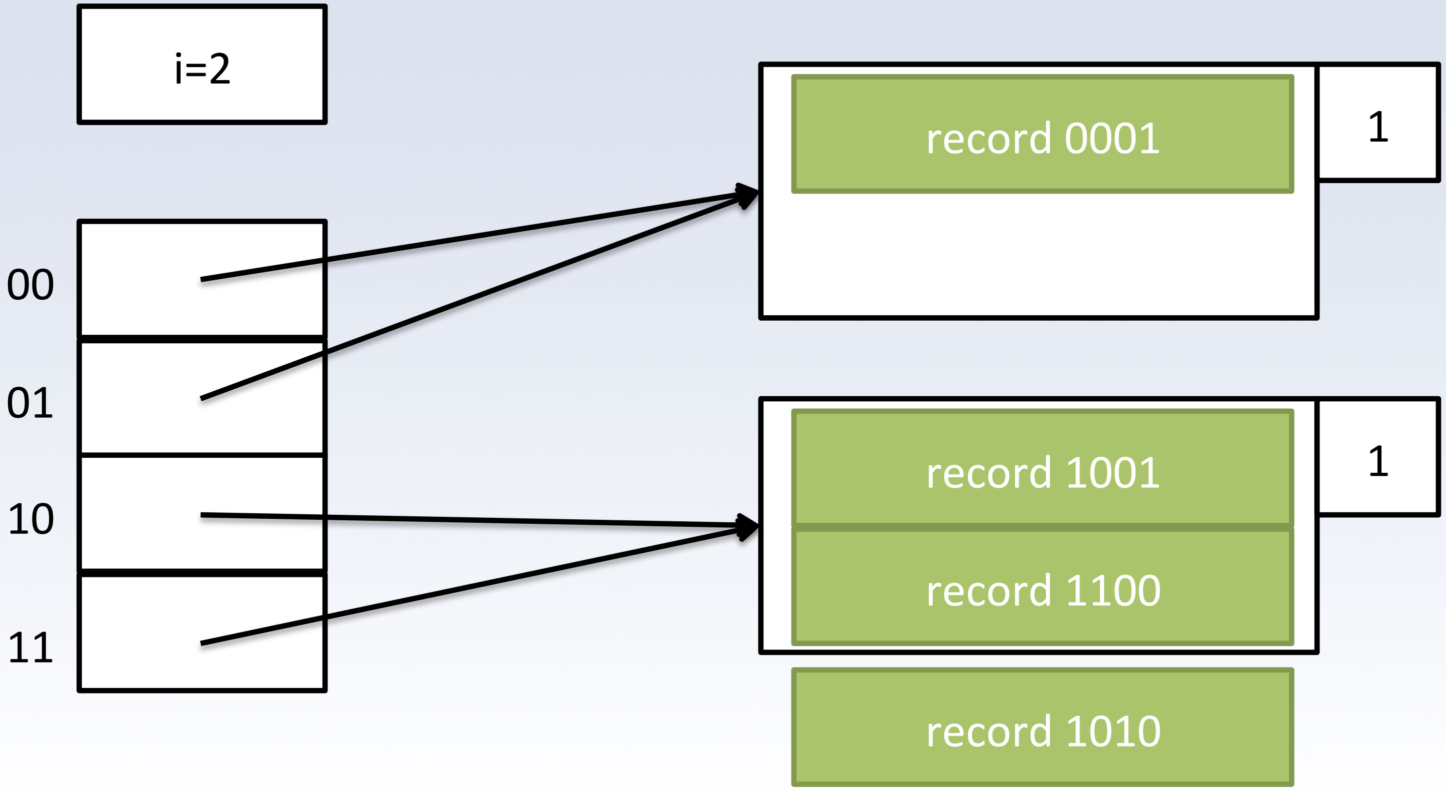
Example: Insert 1010



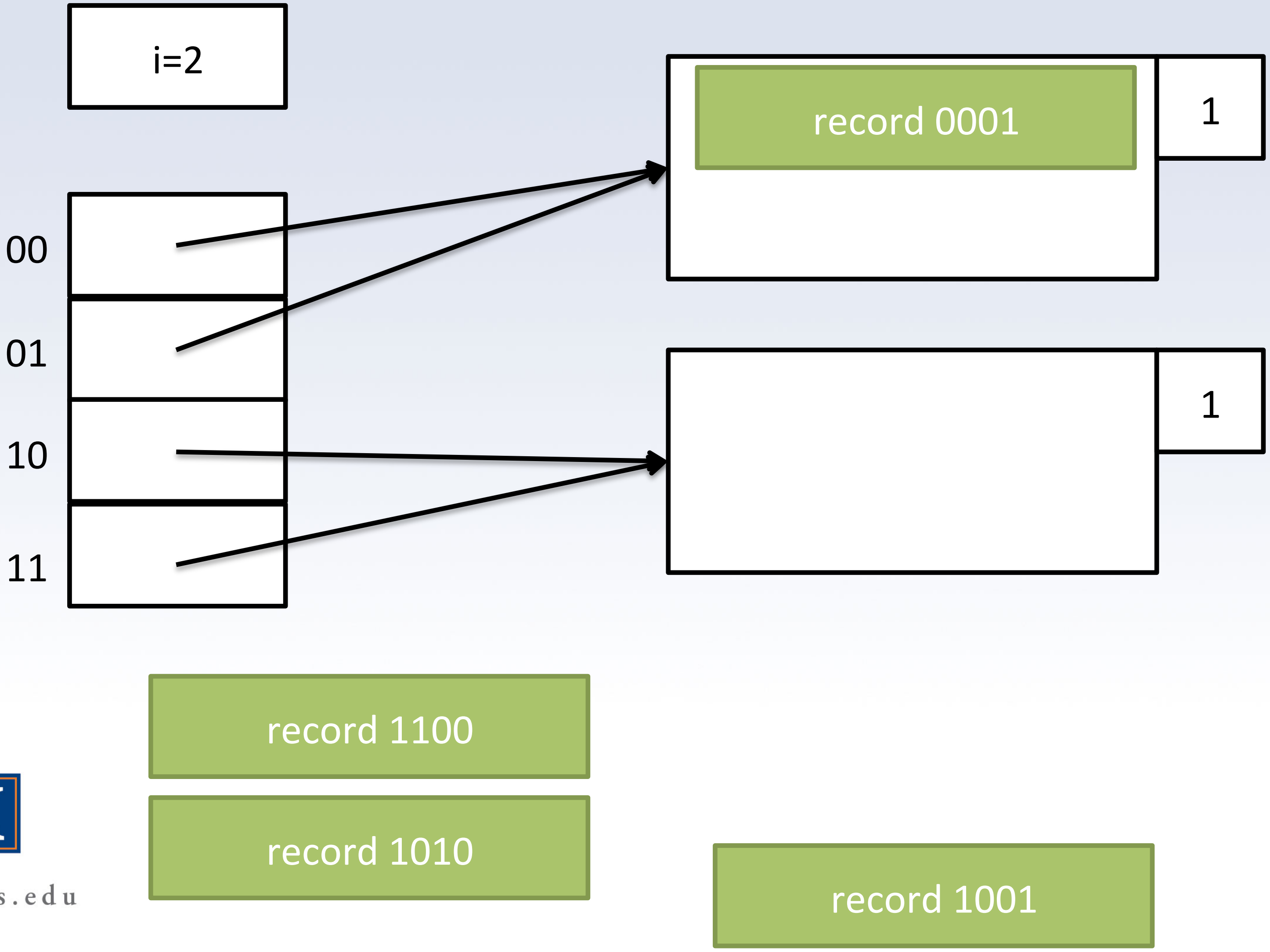
Example: Insert 1010



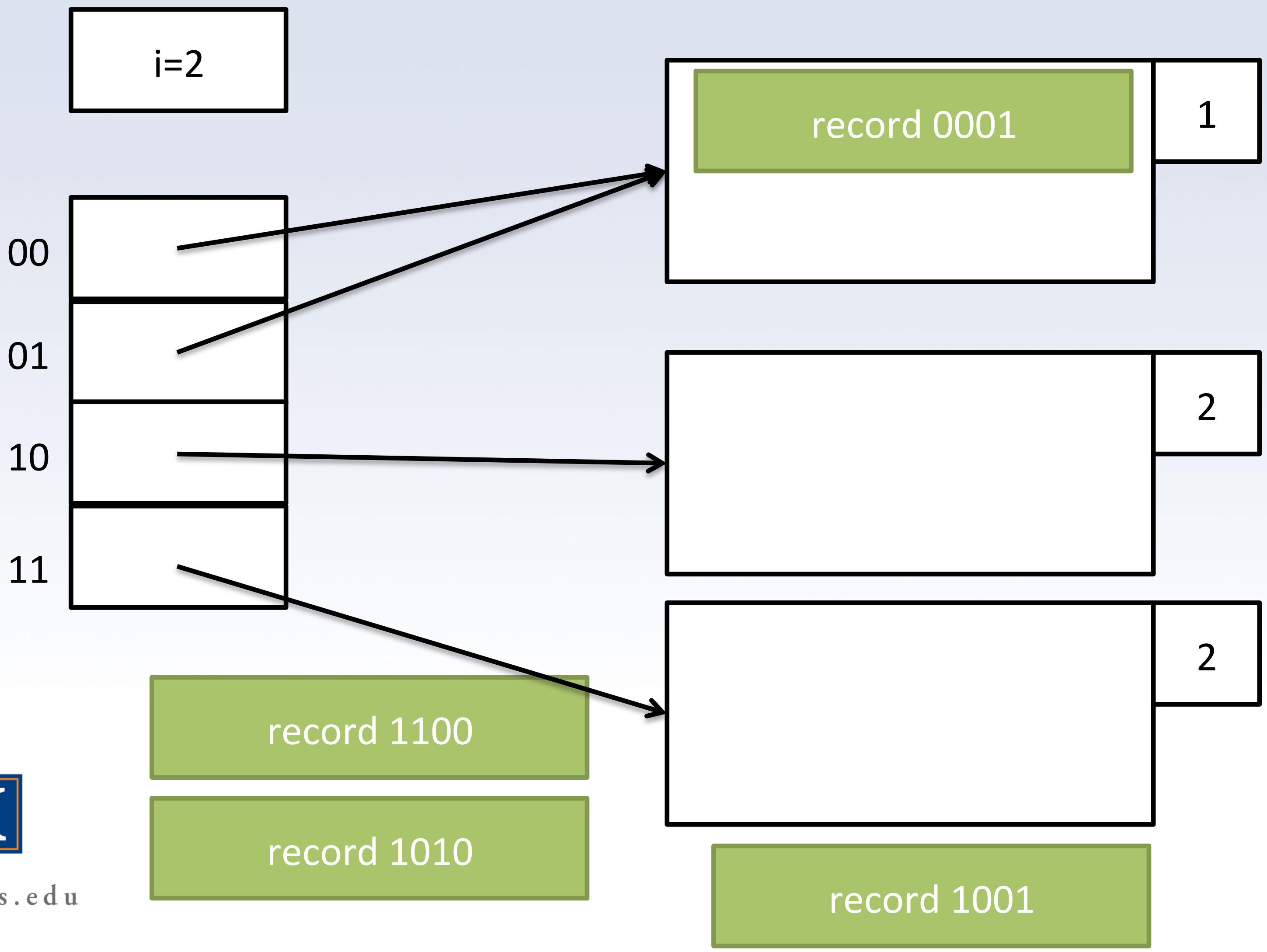
Example: Insert 1010



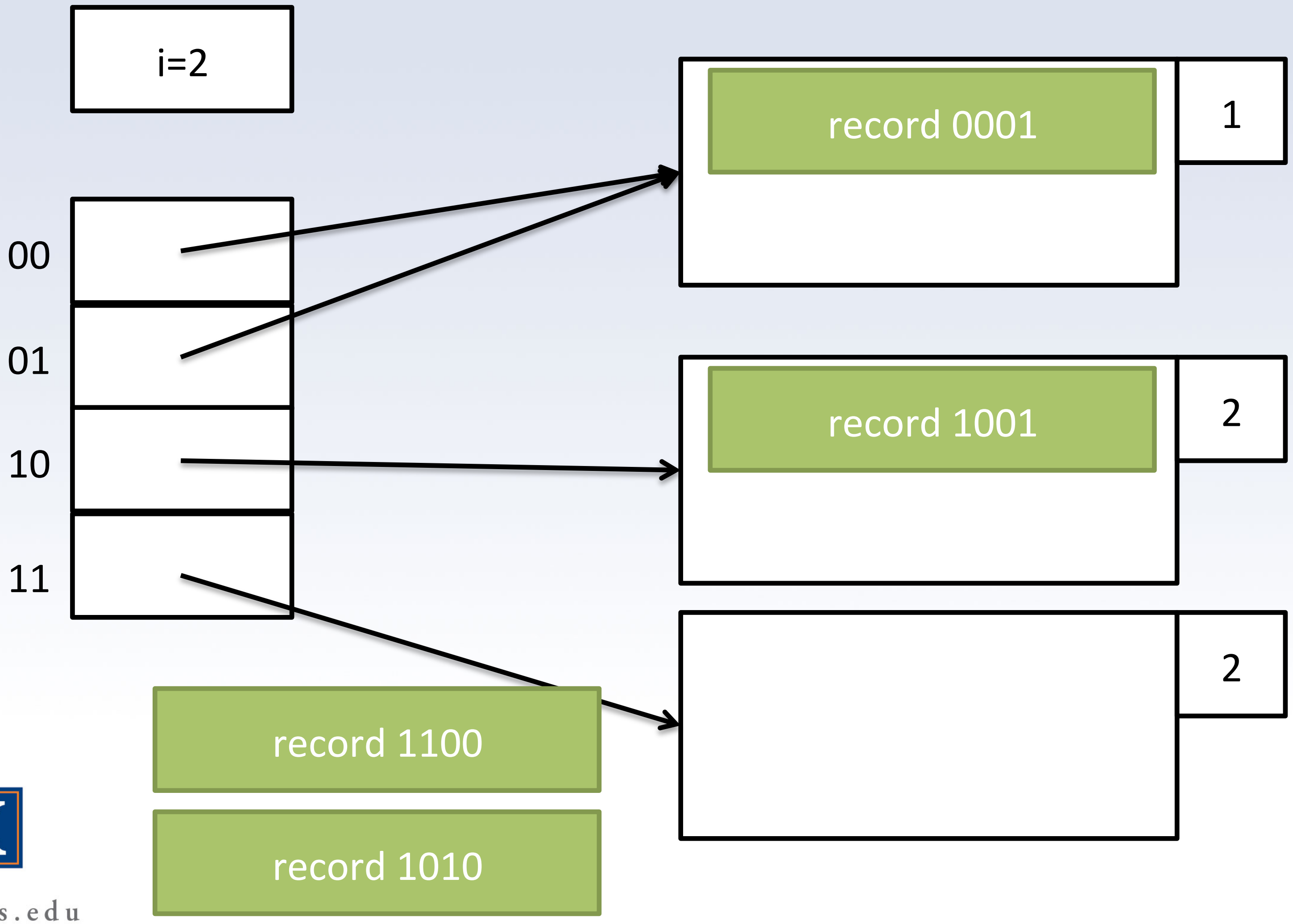
Example: Insert 1010



Example: Insert 1010

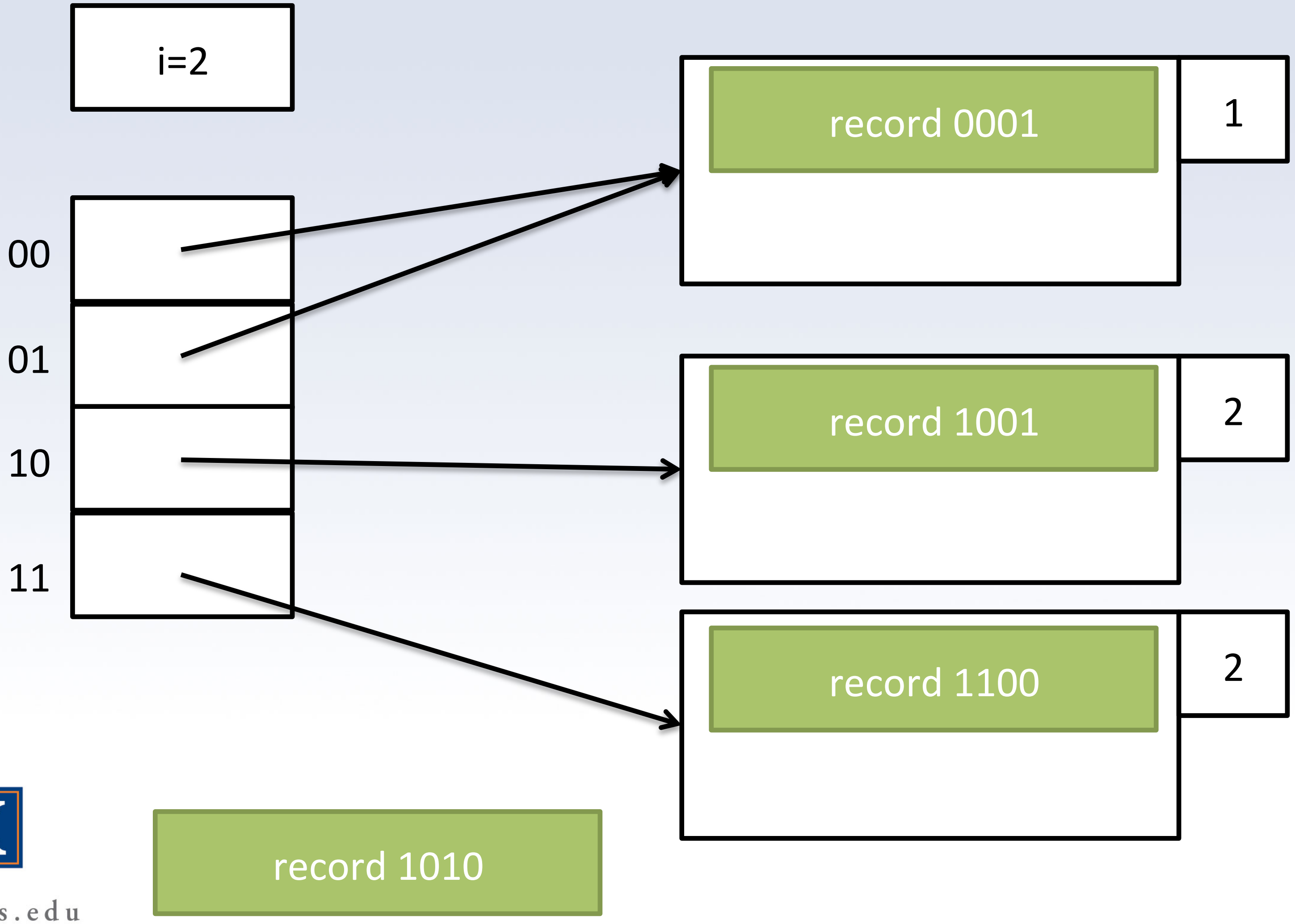


Example: Insert 1010

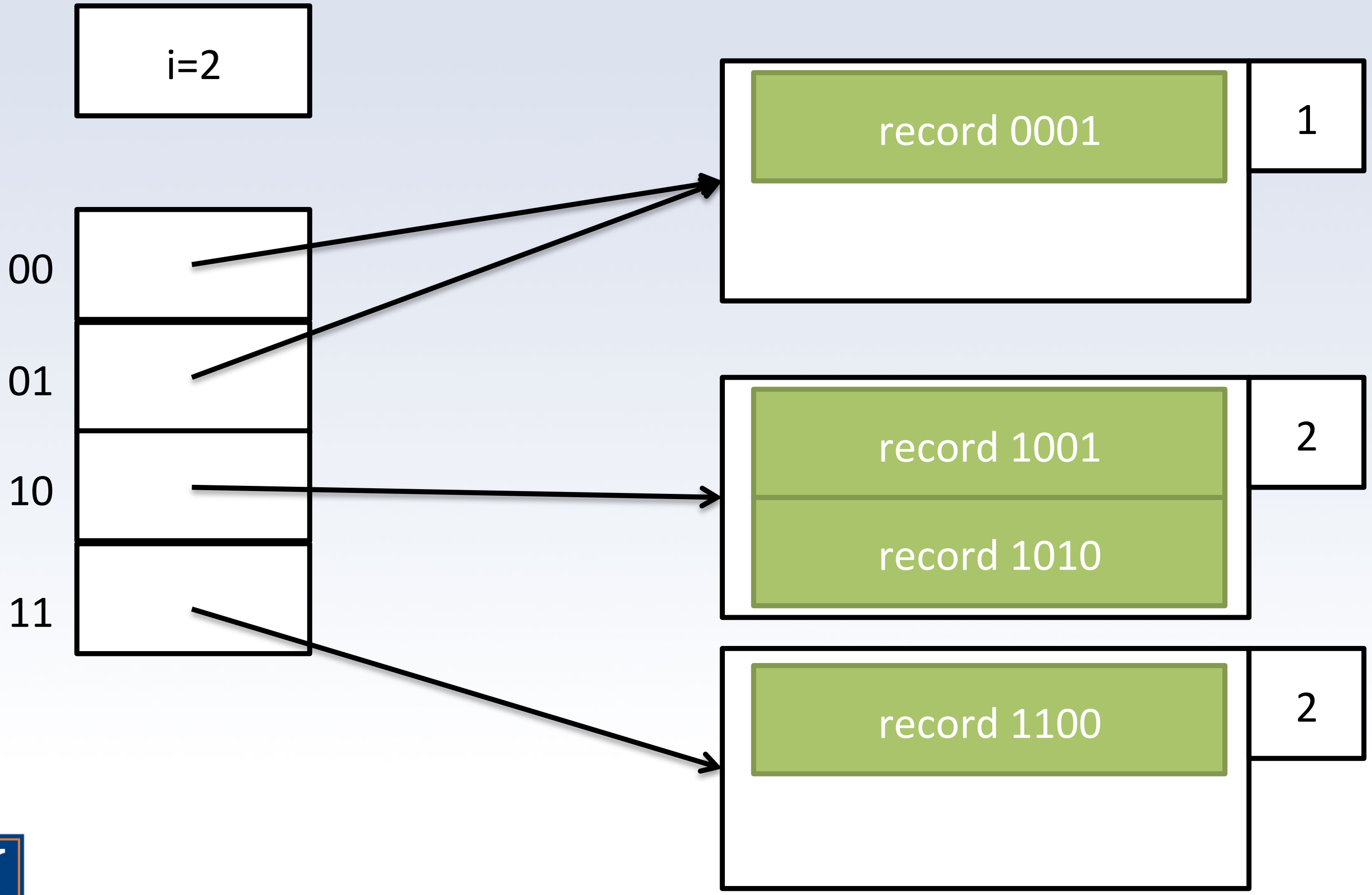


illinois.edu

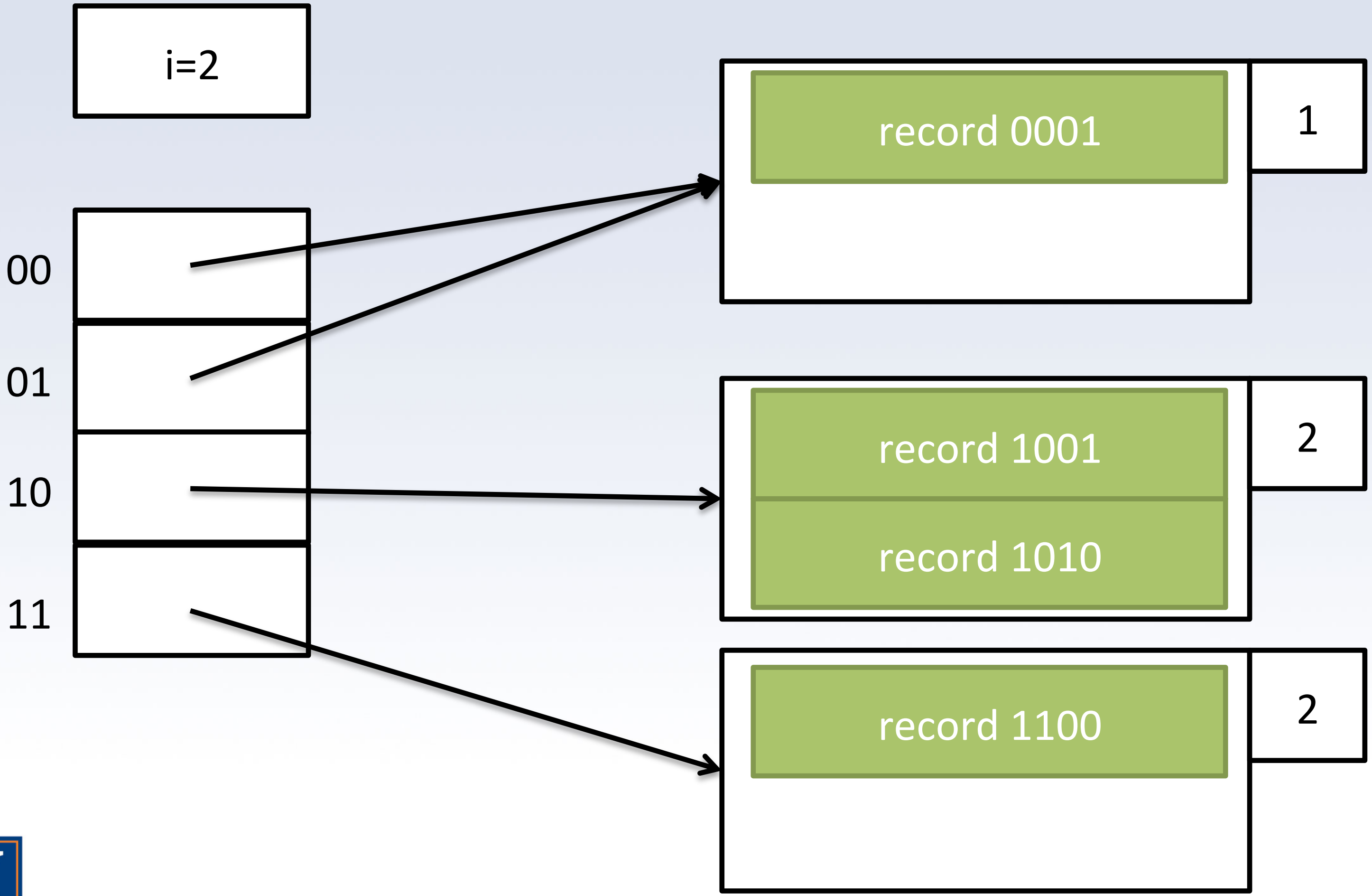
Example: Insert 1010



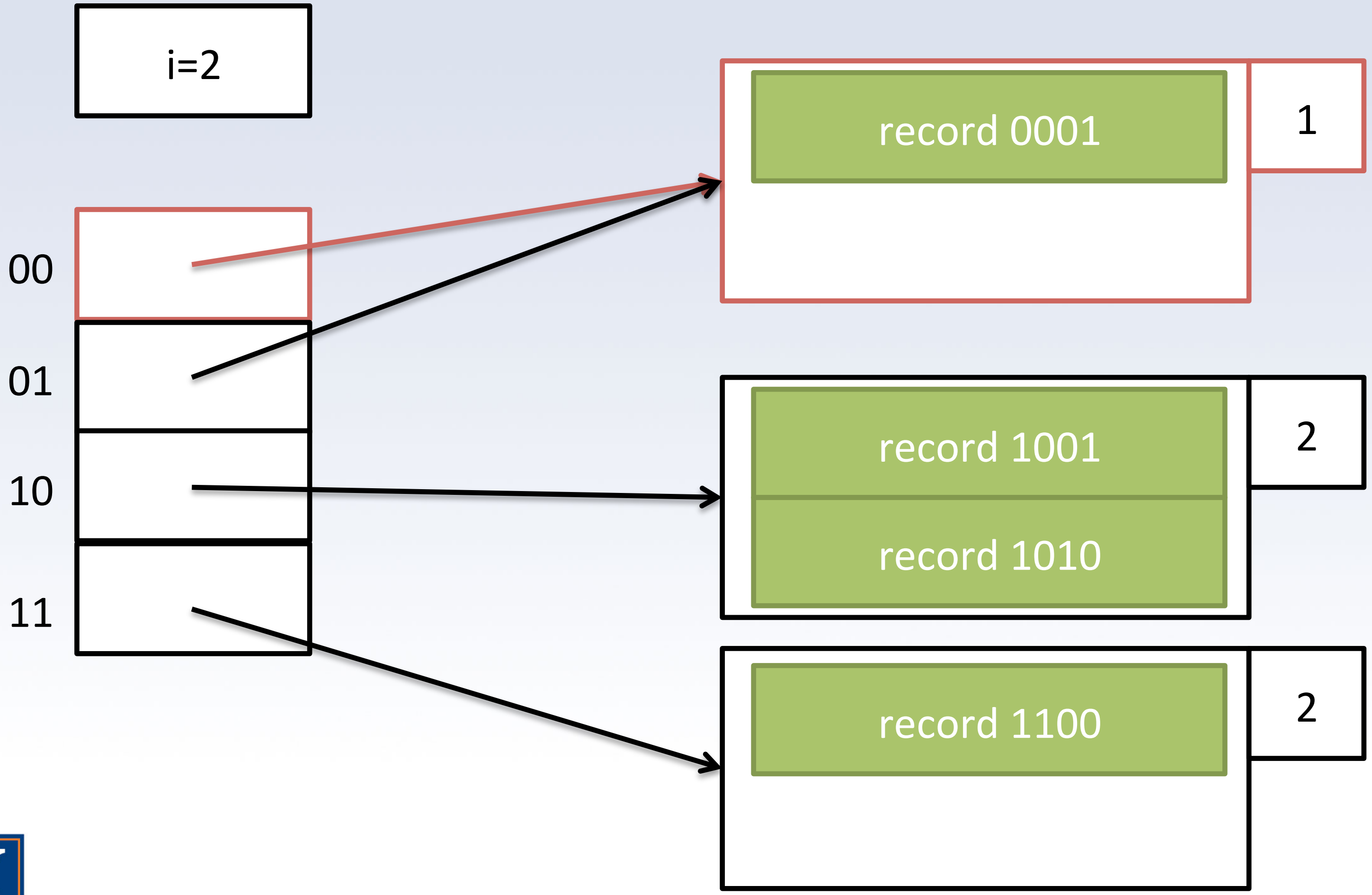
Example: Insert 1010



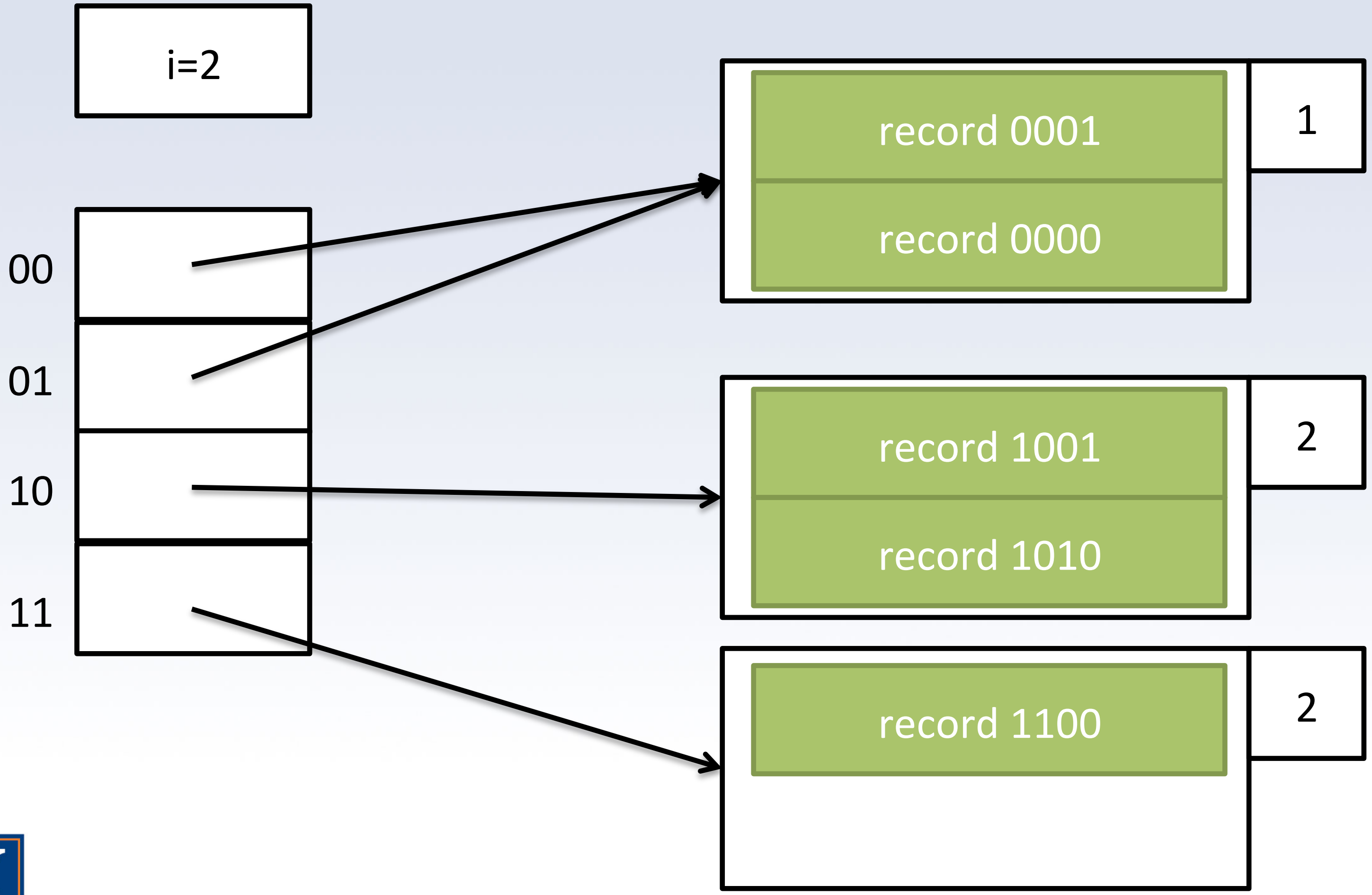
Example: Insert 0000



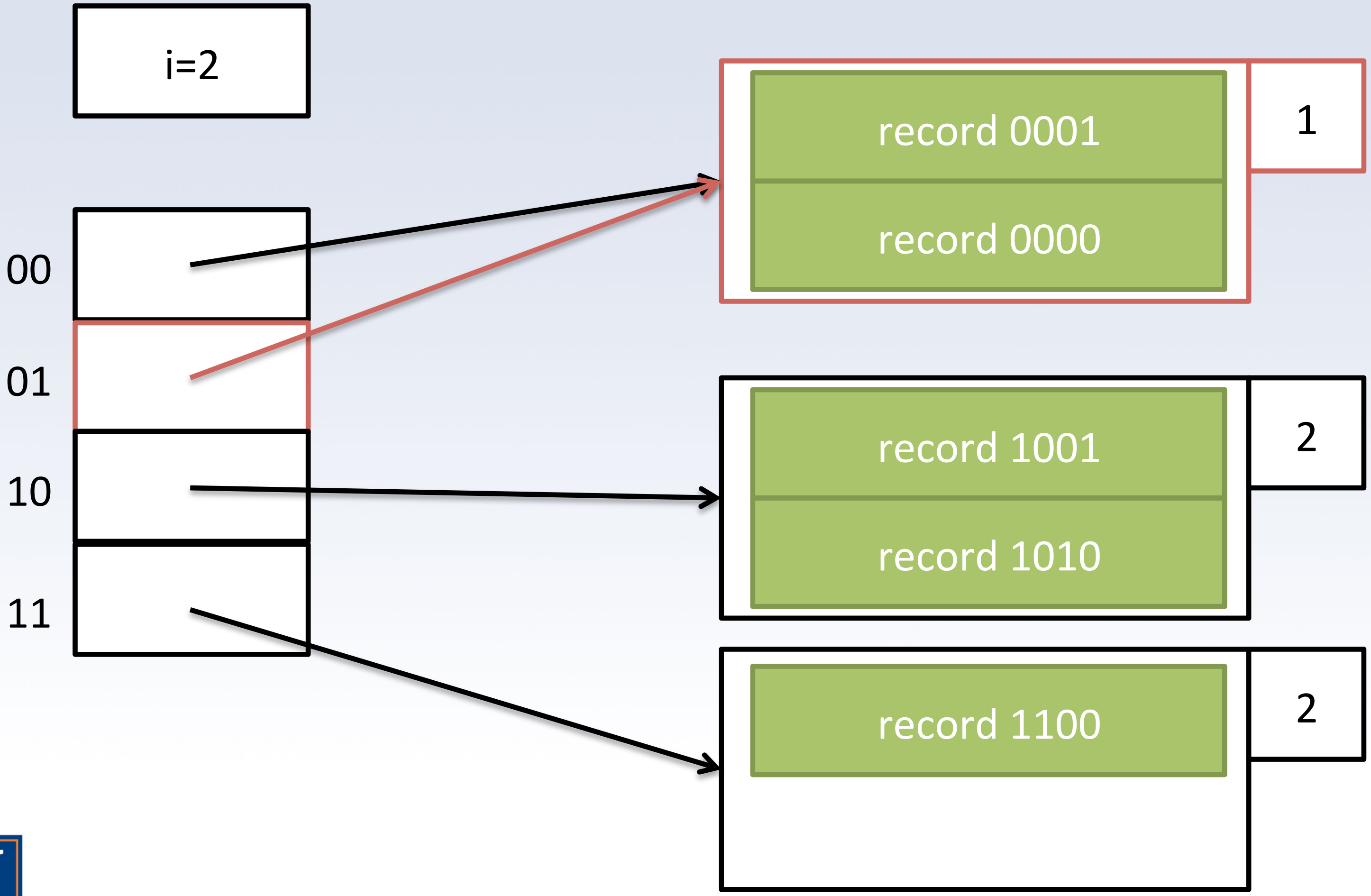
Example: Insert 0000



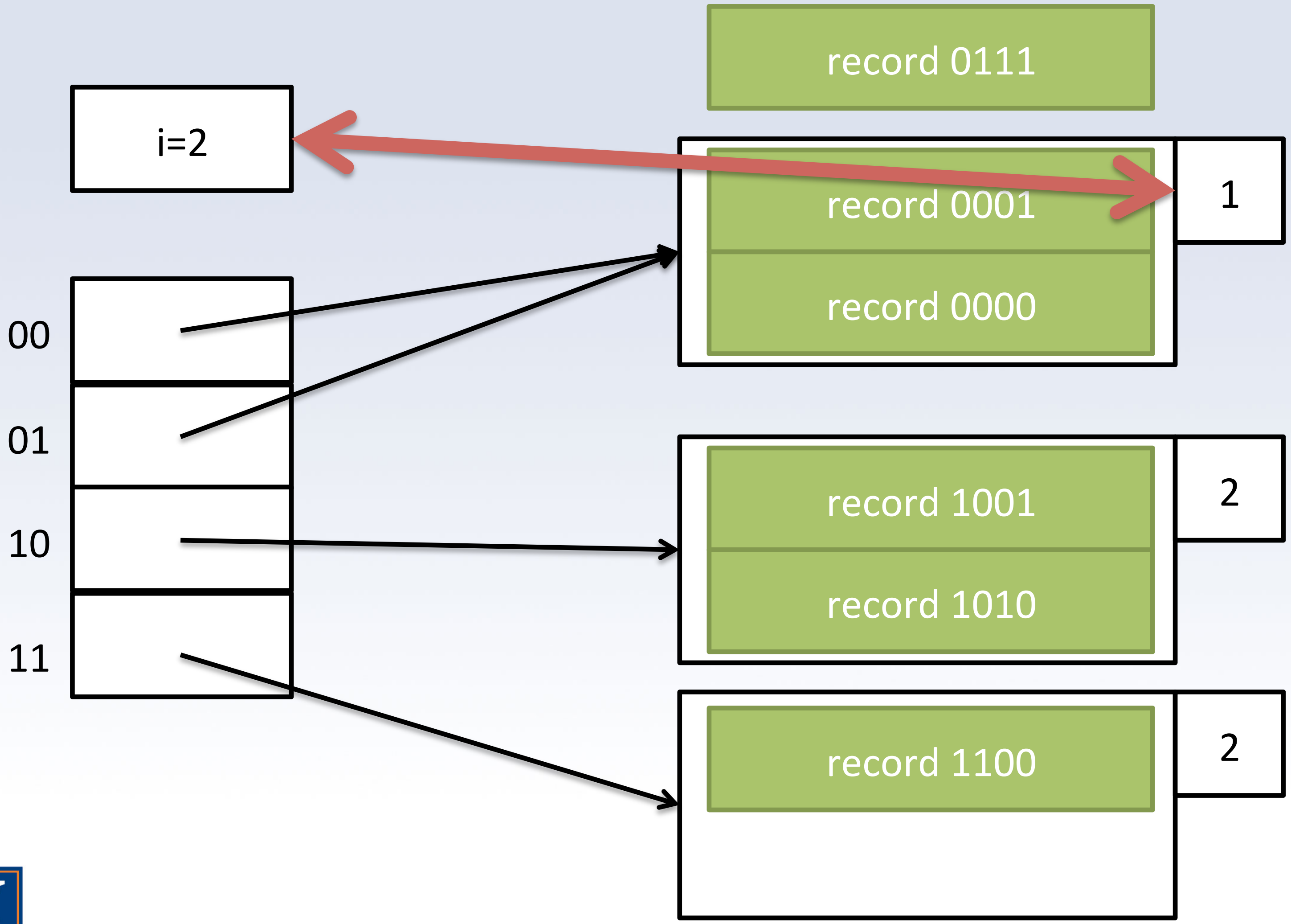
Example: Insert 0000



Example: Insert 0111

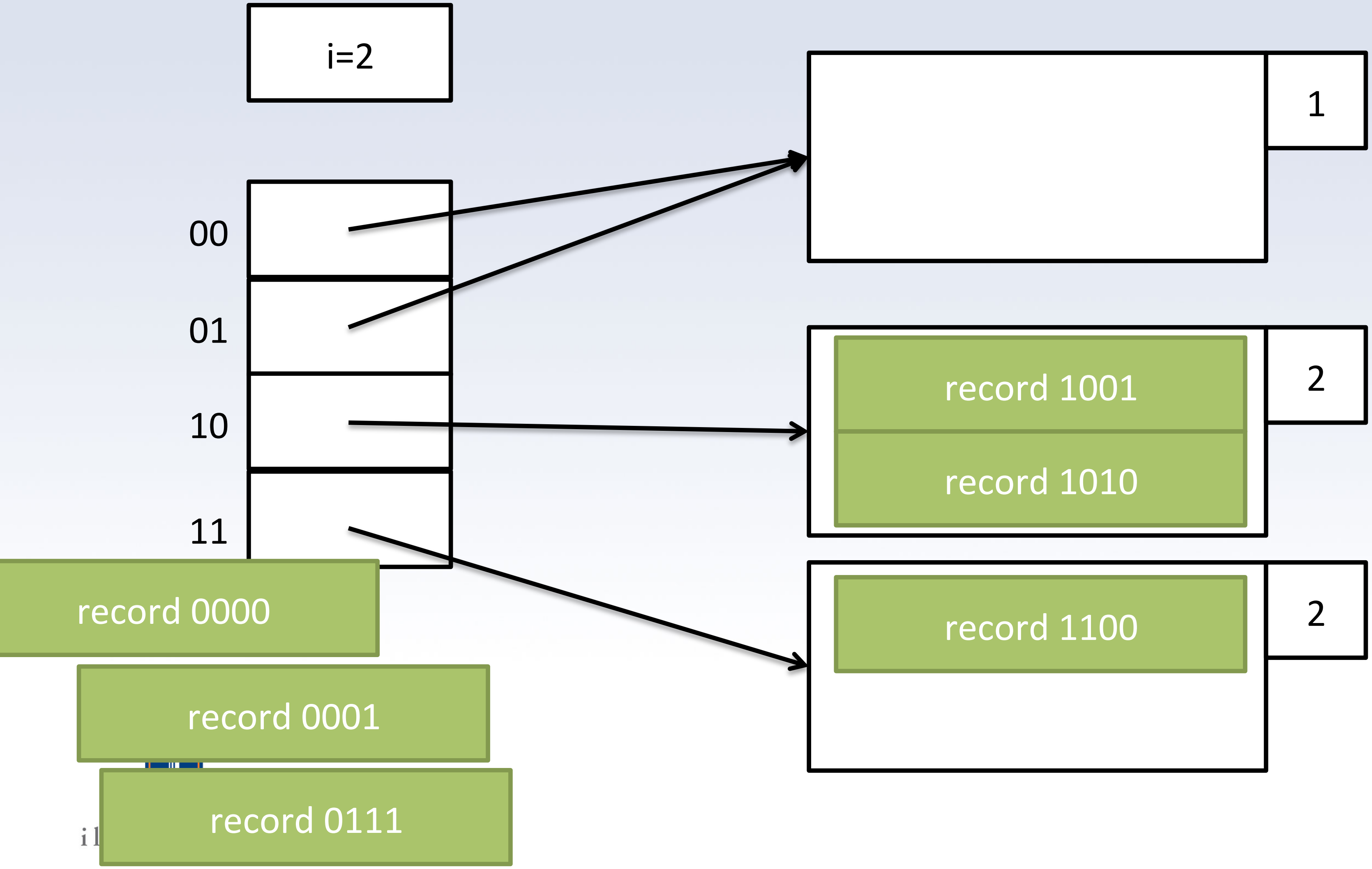


Example: Insert 0111

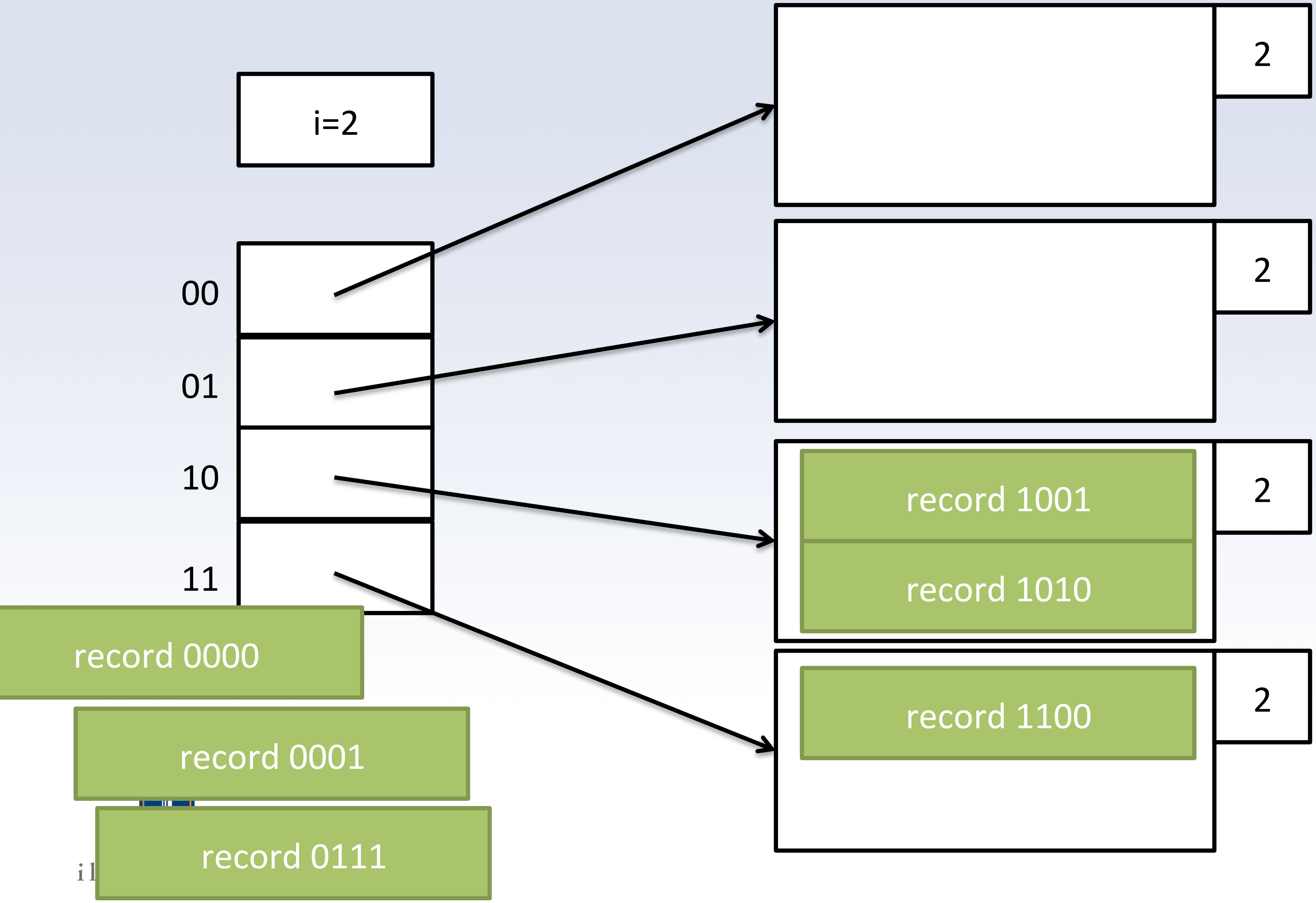


illinois.edu

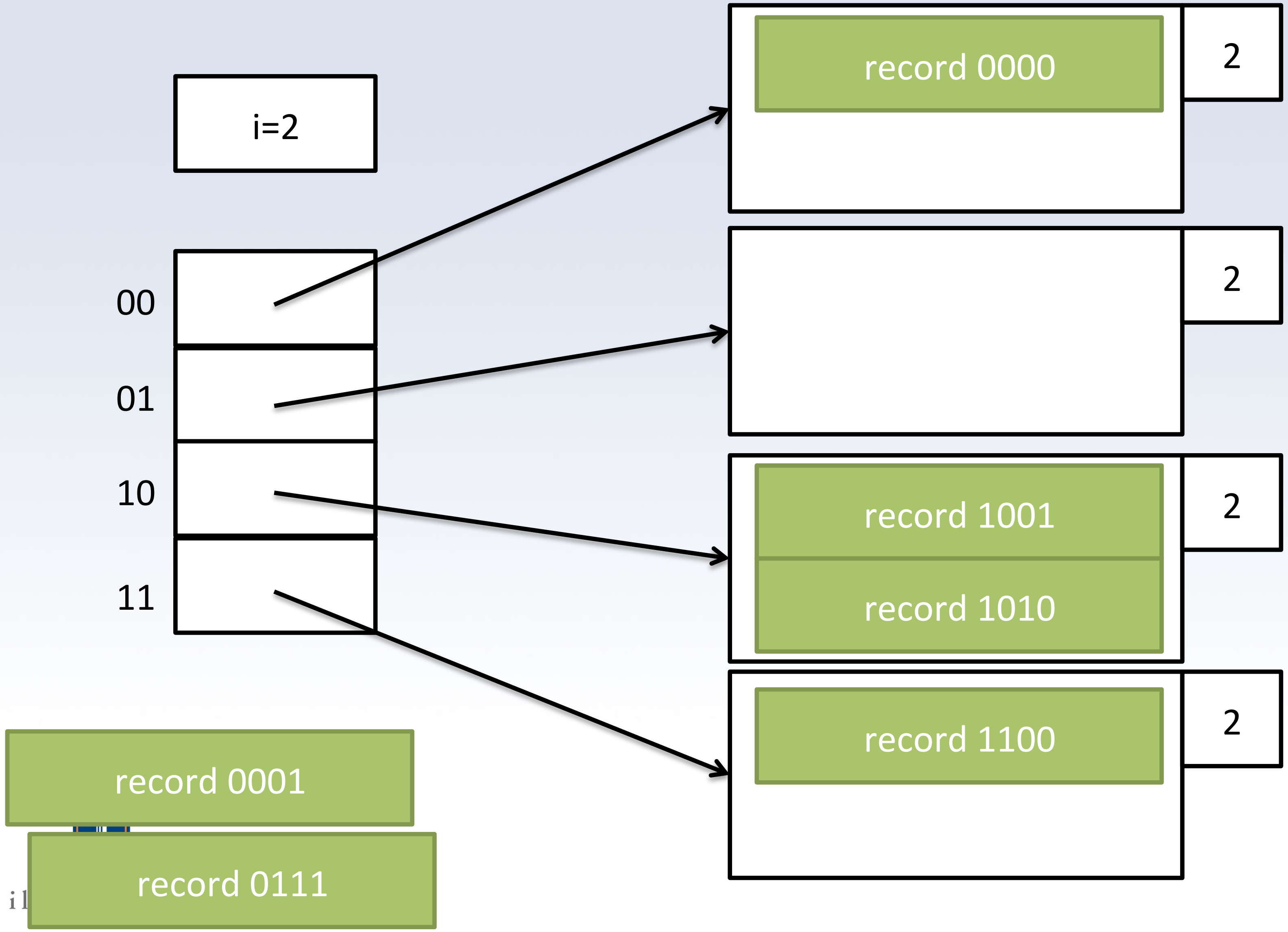
Example: Insert 0111



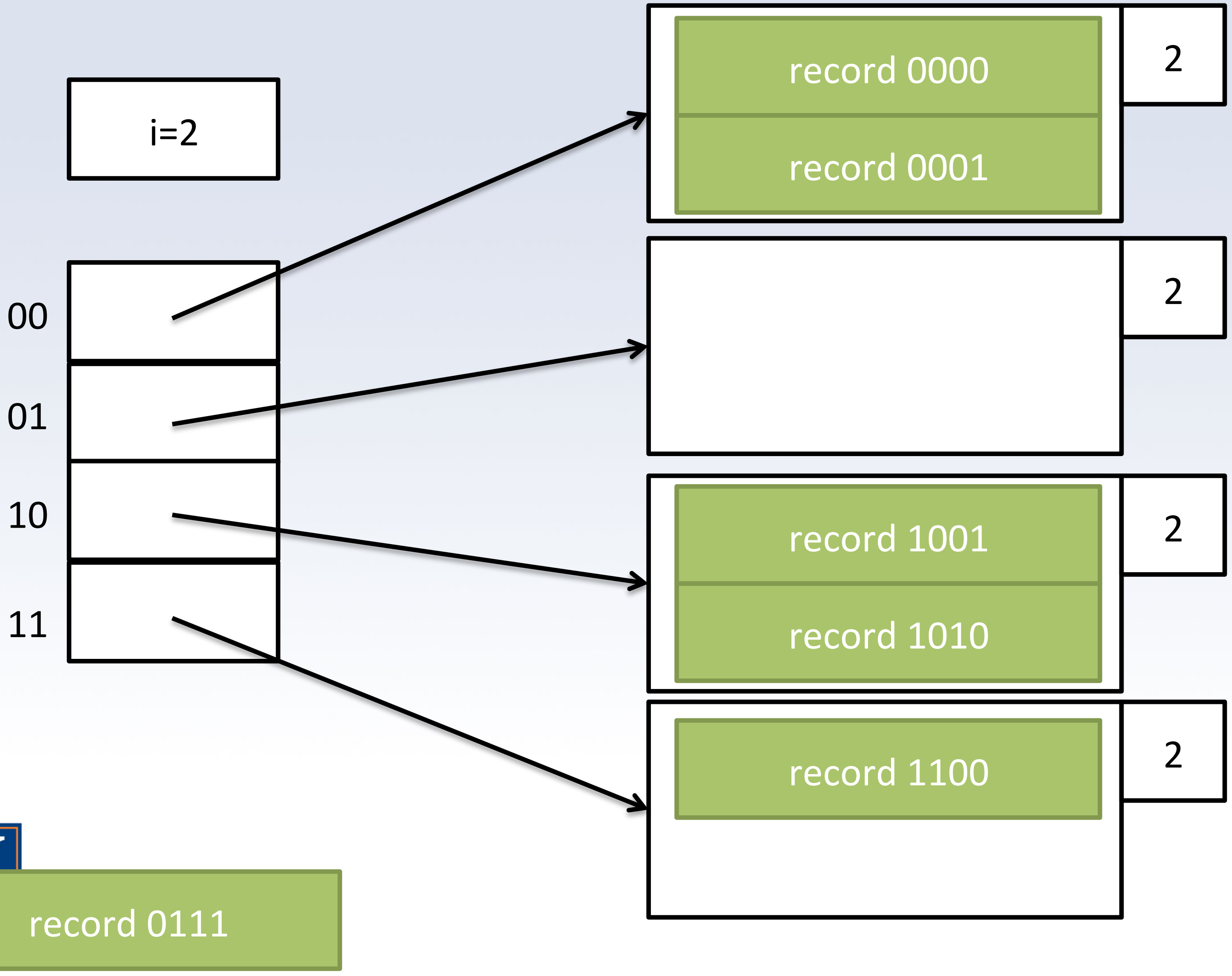
Example: Insert 0111



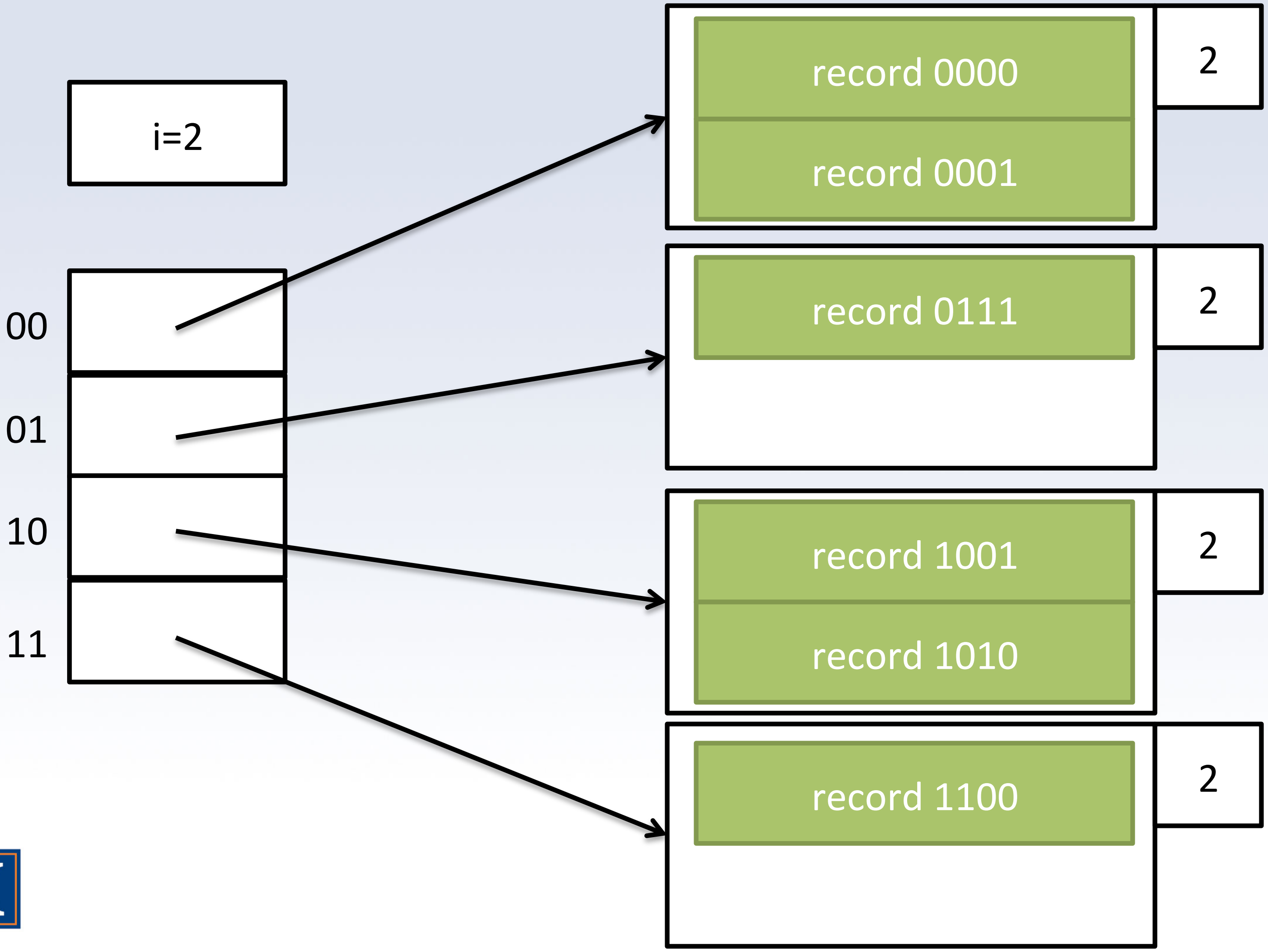
Example: Insert 0111



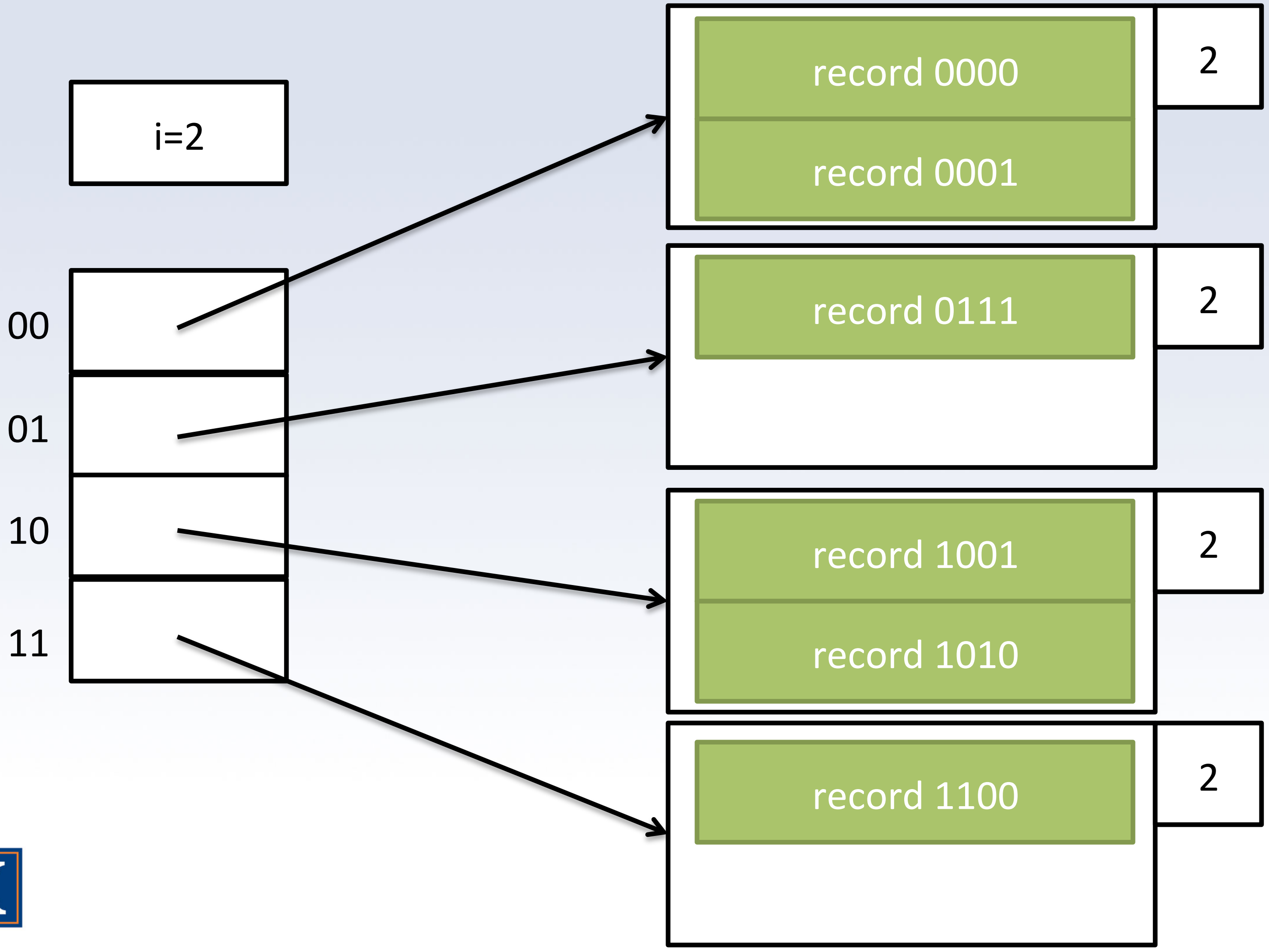
Example: Insert 0111

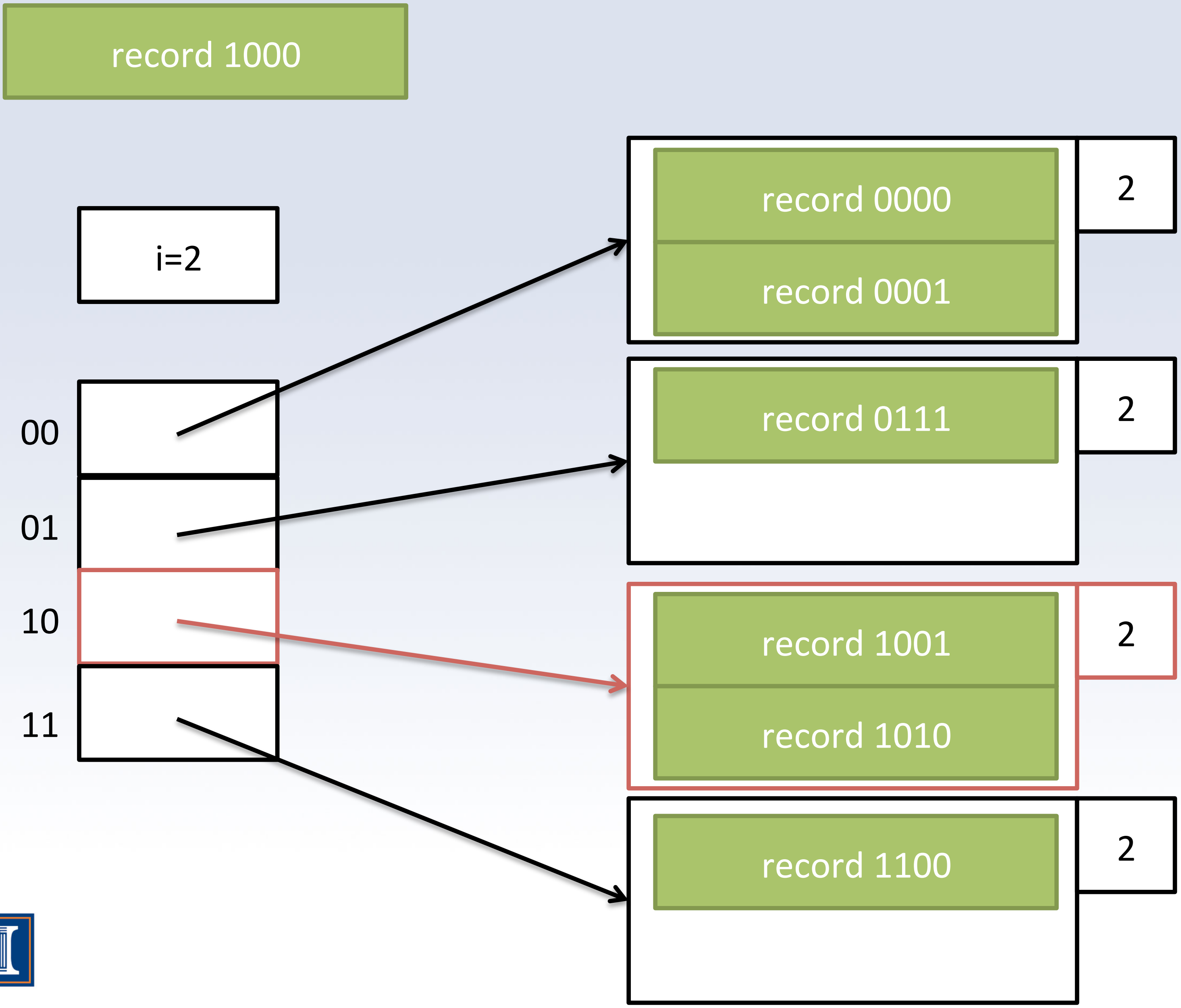


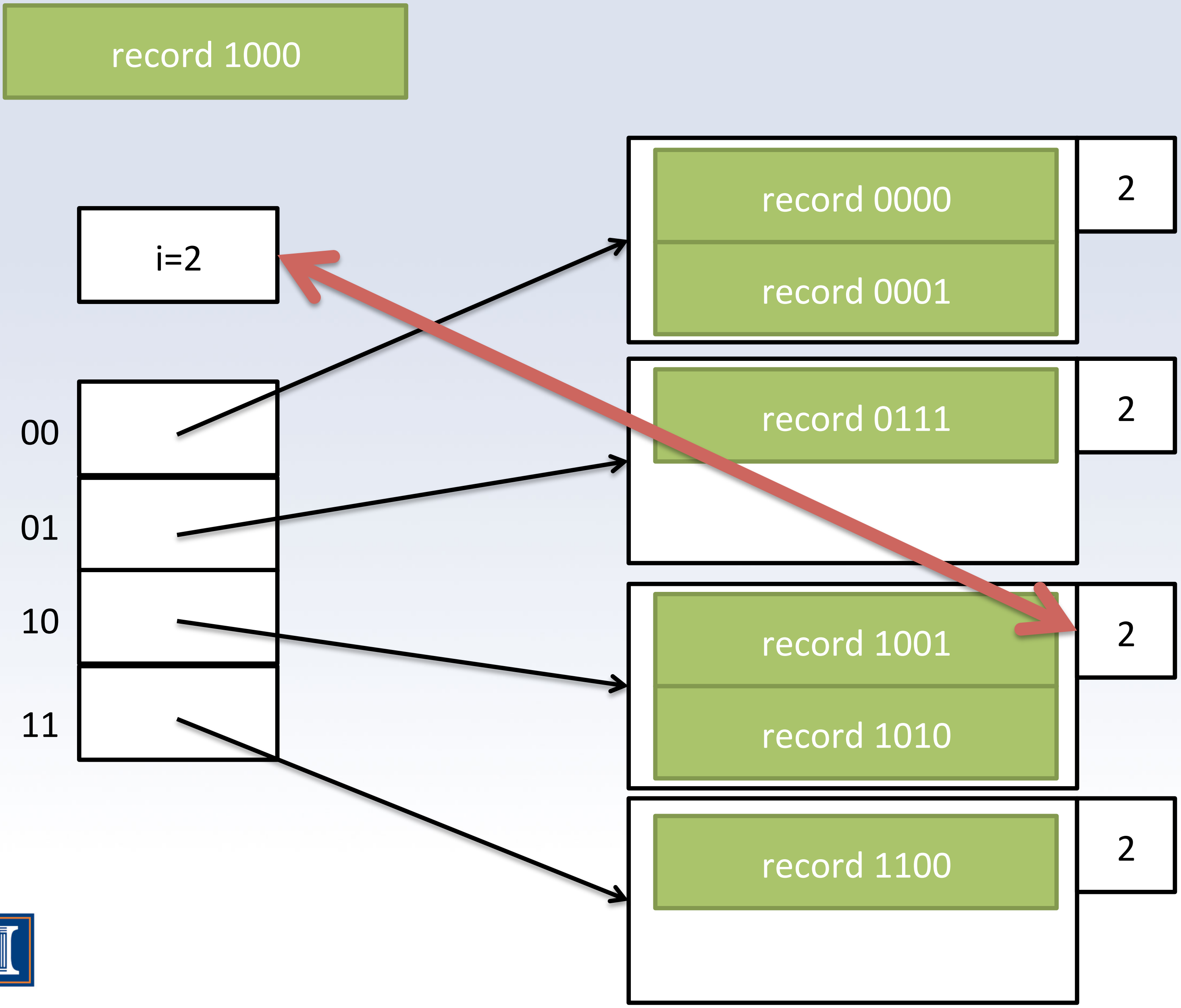
Example: Insert 0111

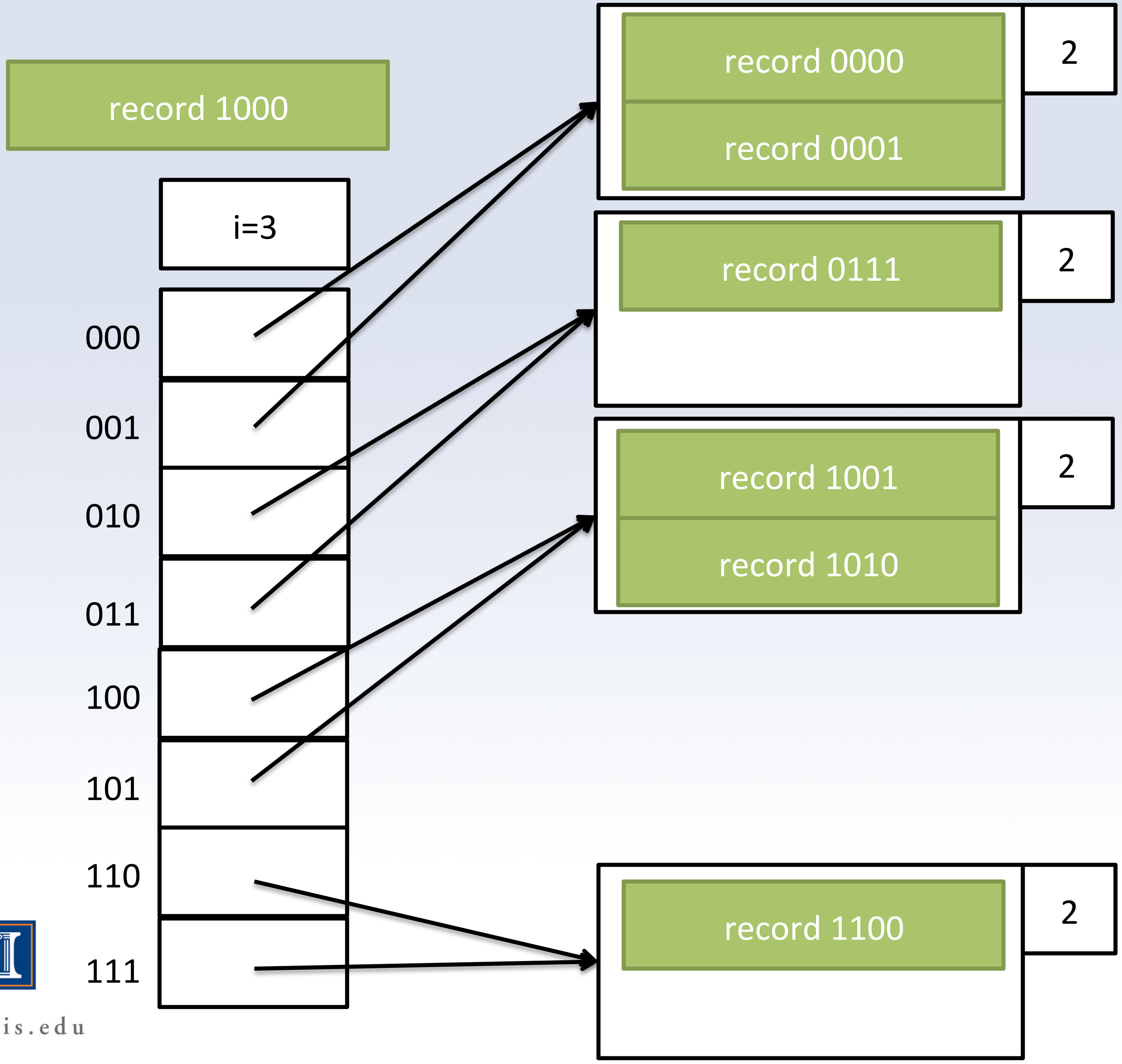


Example: Insert 1000

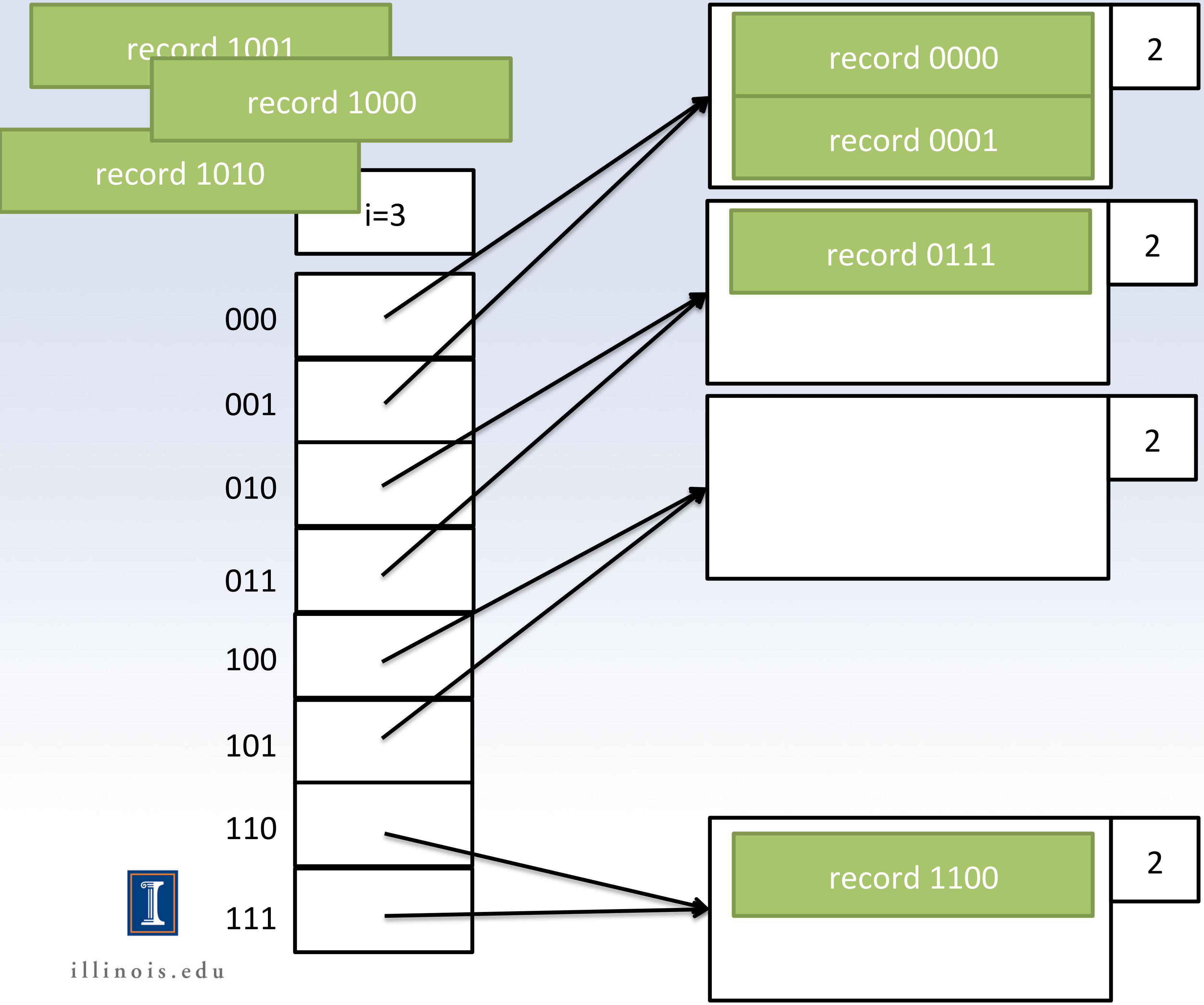



record 1000
i=2
00
01
10
11
record 0000
record 0001
2
record 0111
2
record 1001
record 1010
2
record 1100
2

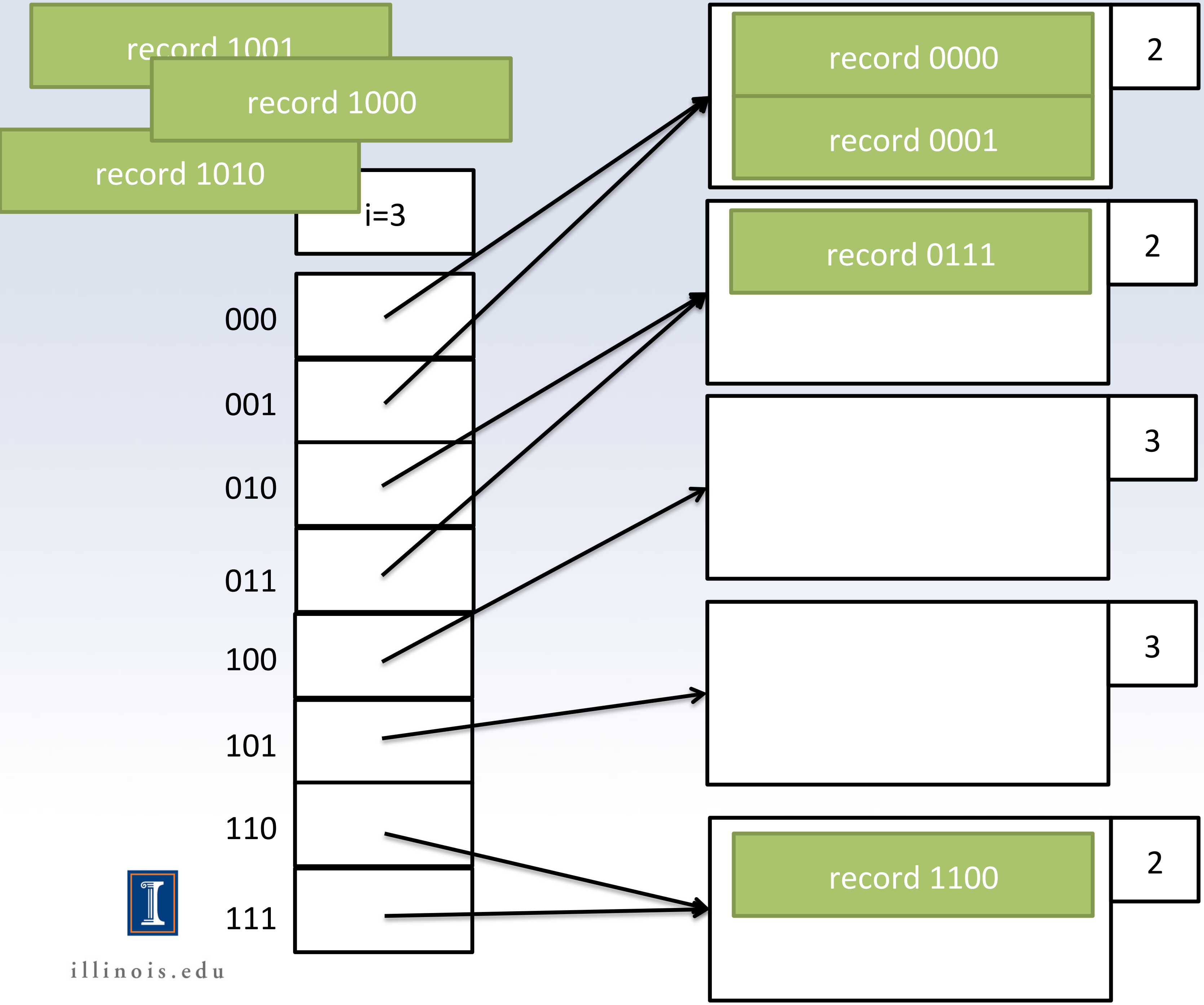

record 1000
i=2
00
01
10
11
record 0000
record 0001
2
record 0111
2
record 1001
record 1010
2
record 1100
2



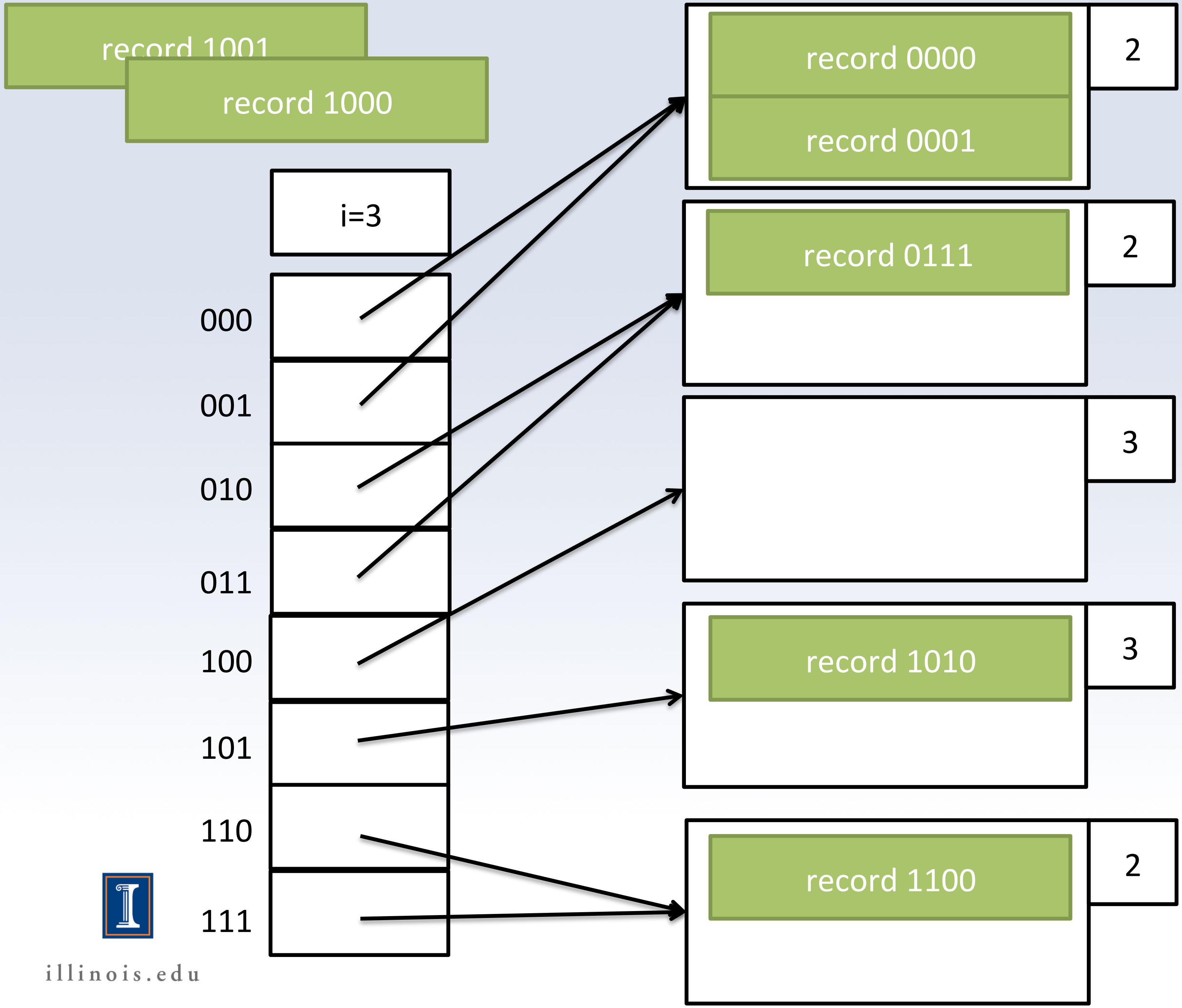
illinois.edu



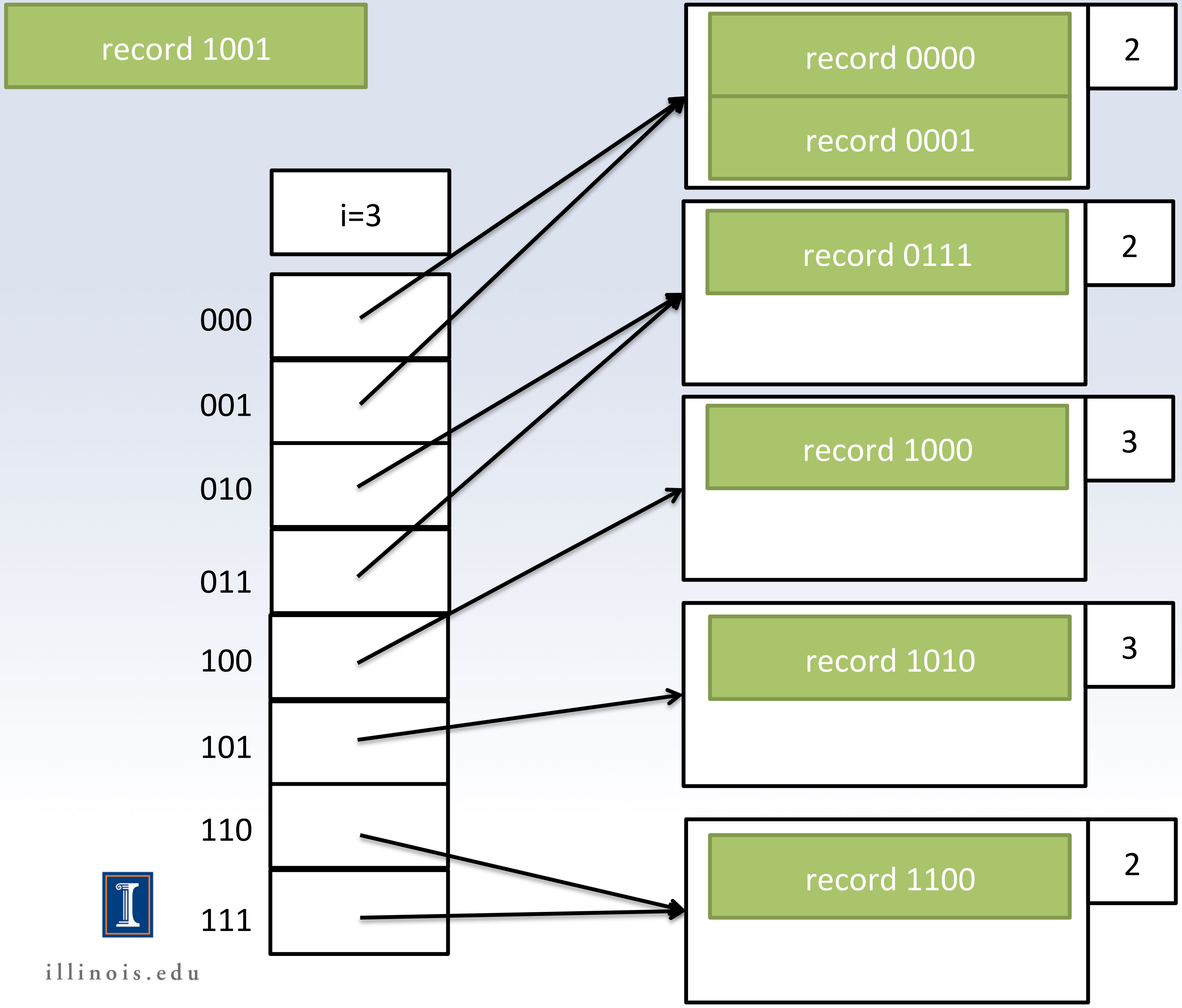
illinois.edu



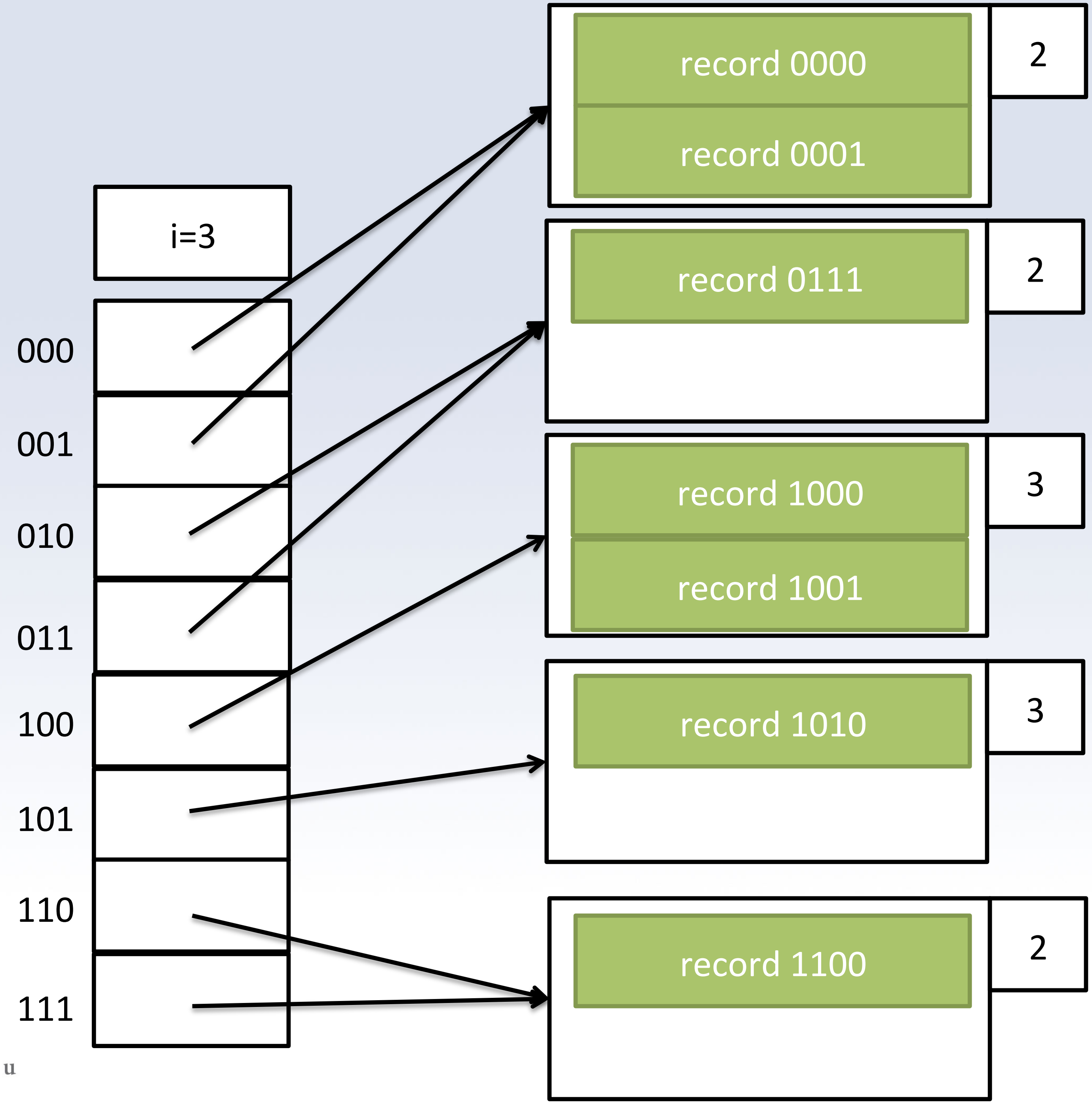
illinois.edu



illinois.edu



illinois.edu


i=3
000
001
010
011
100
101
110
111
record 0000
record 0001
2
record 0111
2
record 1000
record 1001
3
record 1010
3
record 1100
2

Extensible Hash

- Disadvantages:
  - If many keys have the same hash, hash array size explodes
  - Lots of work when hash array doubles
  - Doubling of hash array might make it too big for main memory

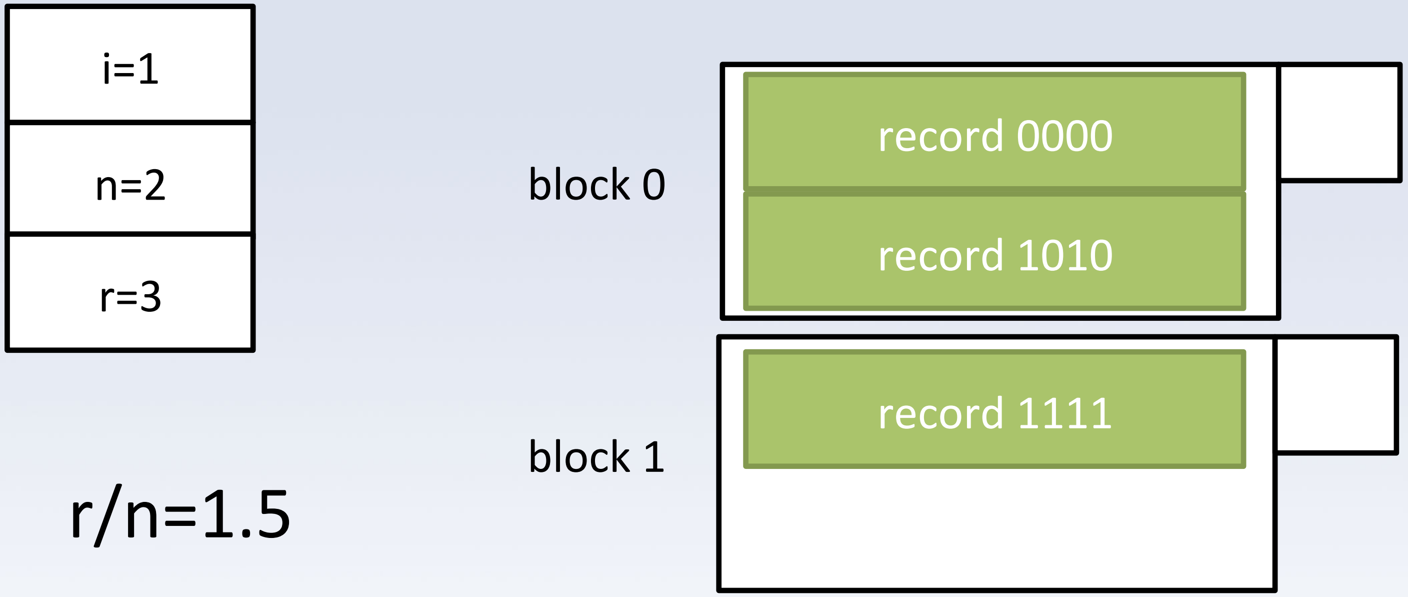
Linear Hash

- Average number of records per bucket fixed (e.g. $r/n \leq 1.7$)
- Number of buckets grows linearly
- Overflow blocks permitted

Linear Hash

- need to track 3 variables
  - i = the number of bits of the hash we use
  - r = the number of records inserted
  - n = the number of buckets in the hash
- $i = \text{ceiling}(\log_2 n)$
- increment n if r/n exceeds threshold

Example

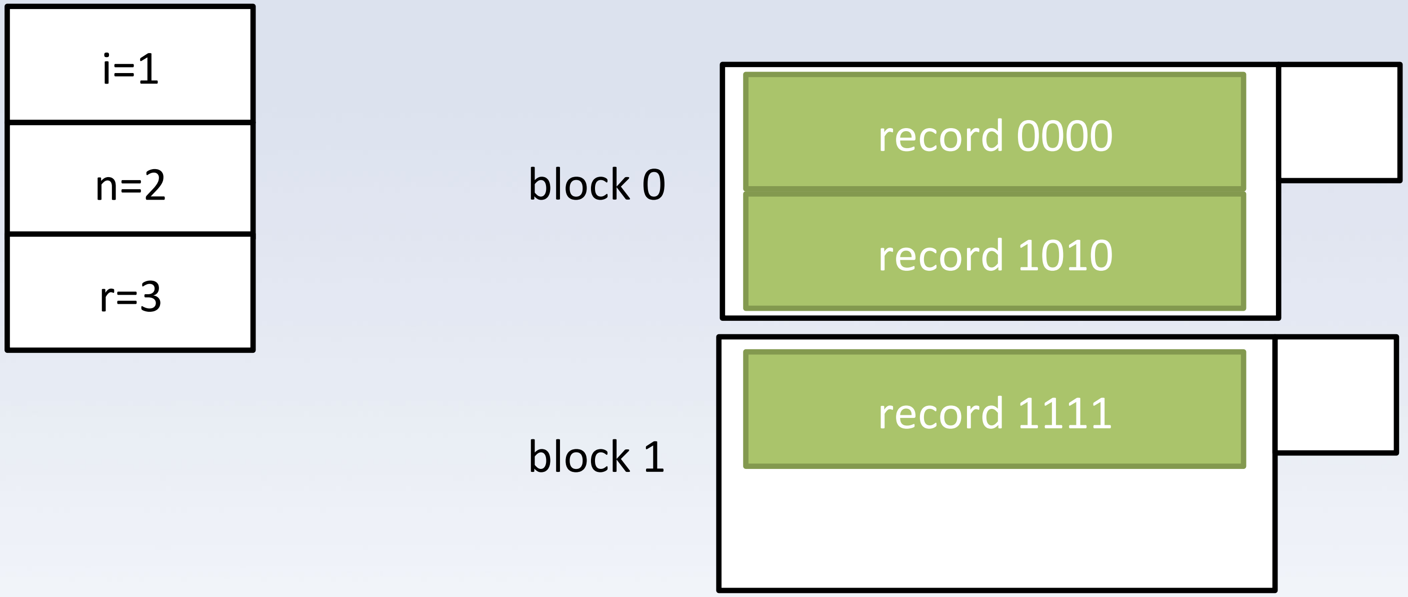


r/n=1.5

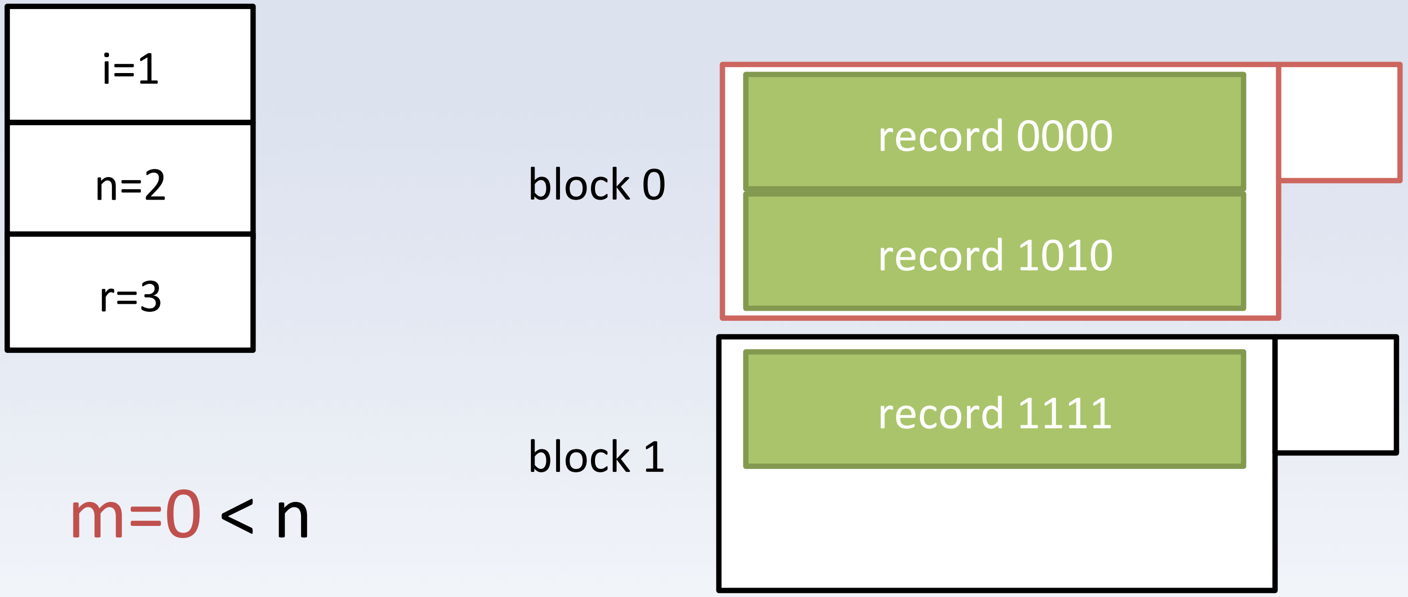
Linear Hash Lookup

- Interpret the last i blocks of the key as an integer m
- Find block m
  - won't exist if m>=n
  - switch the high order bit of m to 0
  - look in that block

Example: Lookup 1010



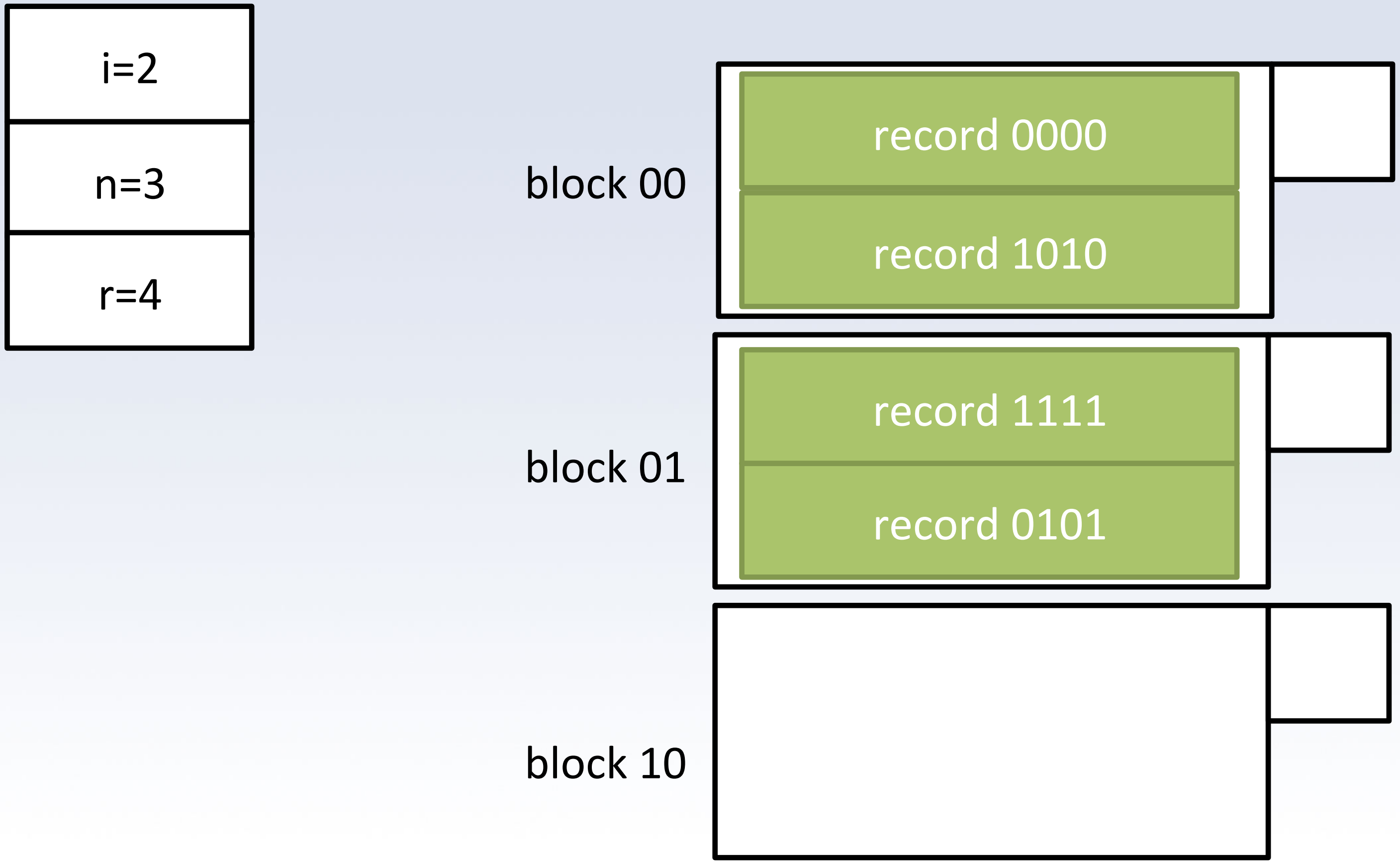
Example: Lookup 1010



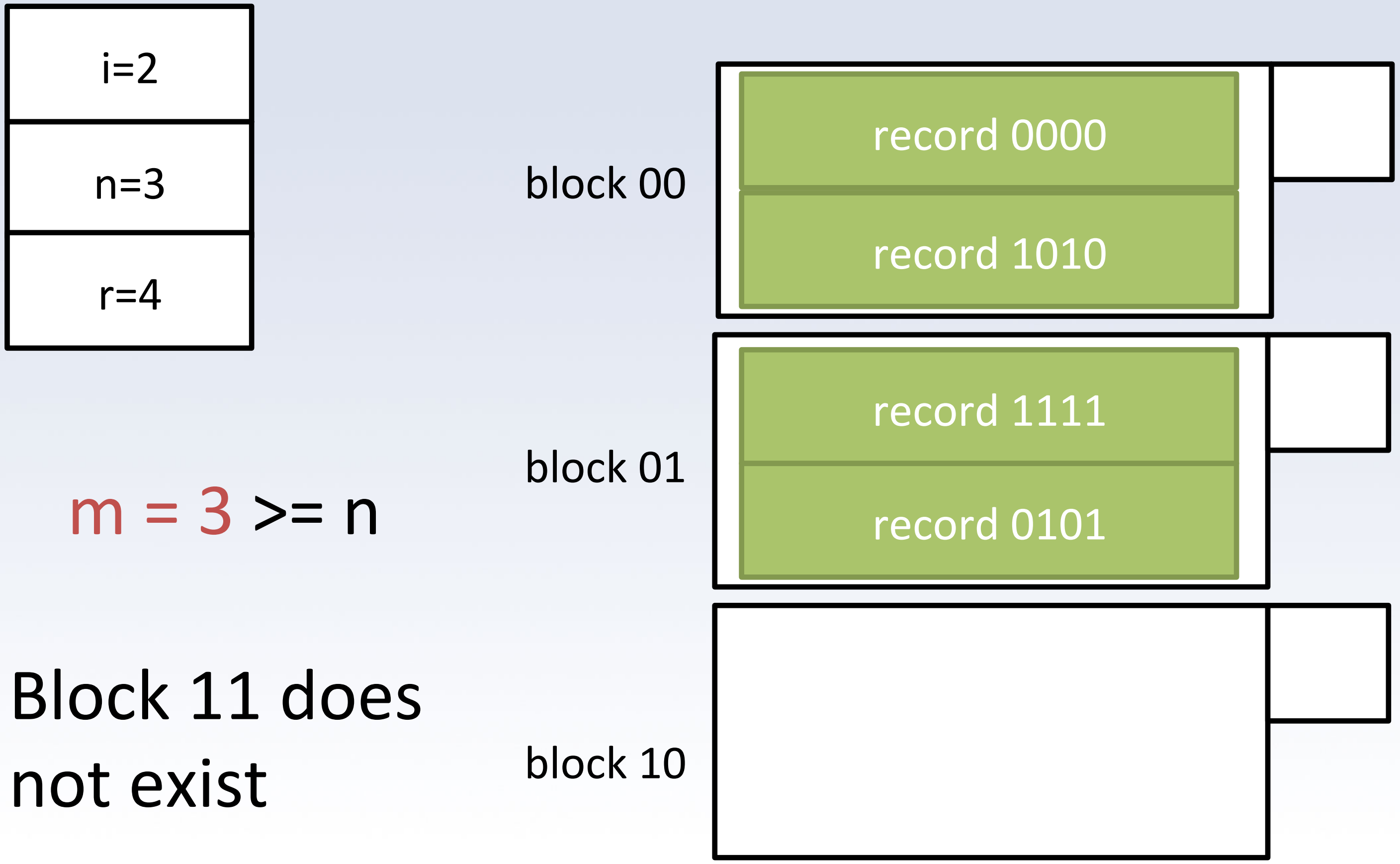
m=0 < n

Block 0 exists

Example: Lookup 1111



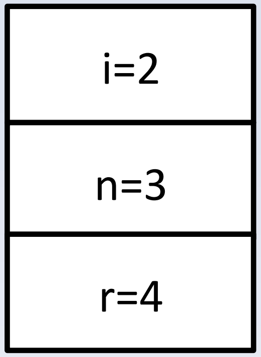
Example: Lookup 1111



m = 3 >= n

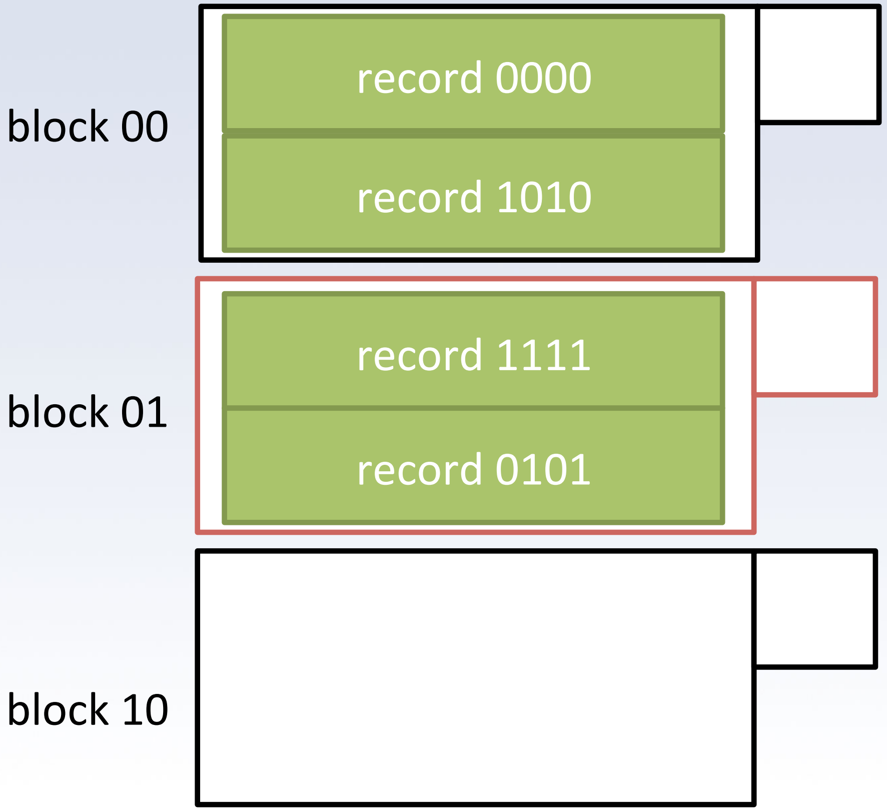
Block 11 does not exist

Example: Lookup 1111



m = 3 >= n

Look in block 01 instead

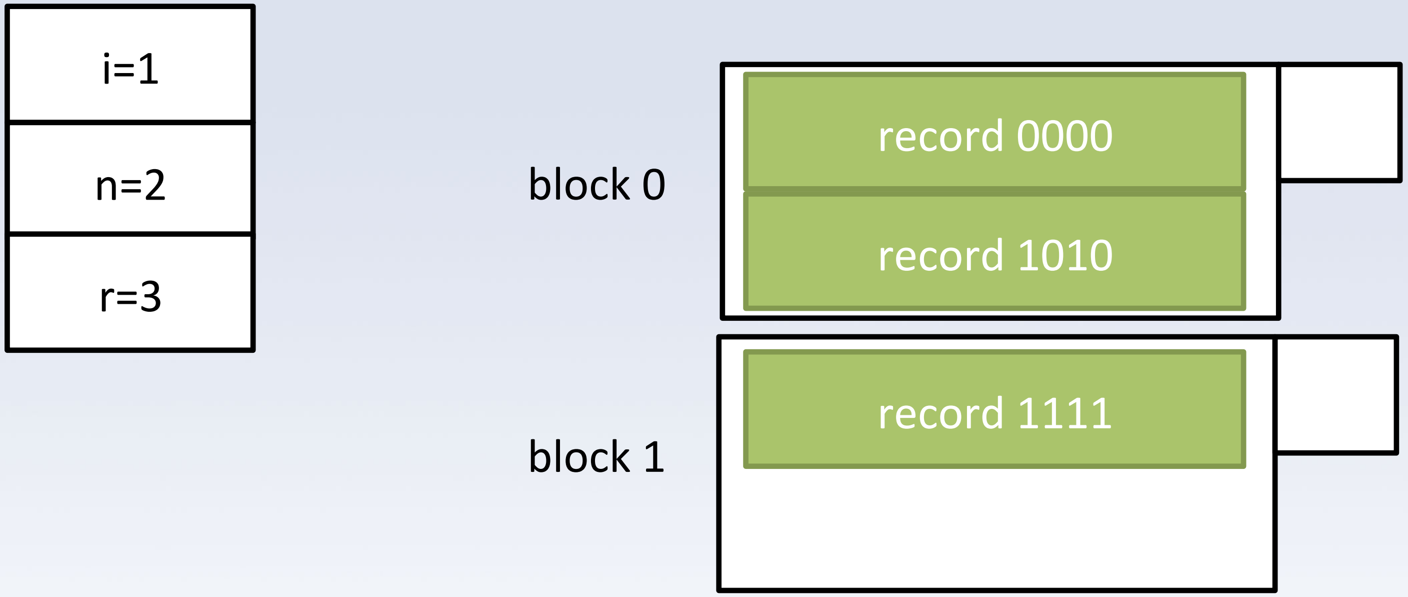


illinois.edu

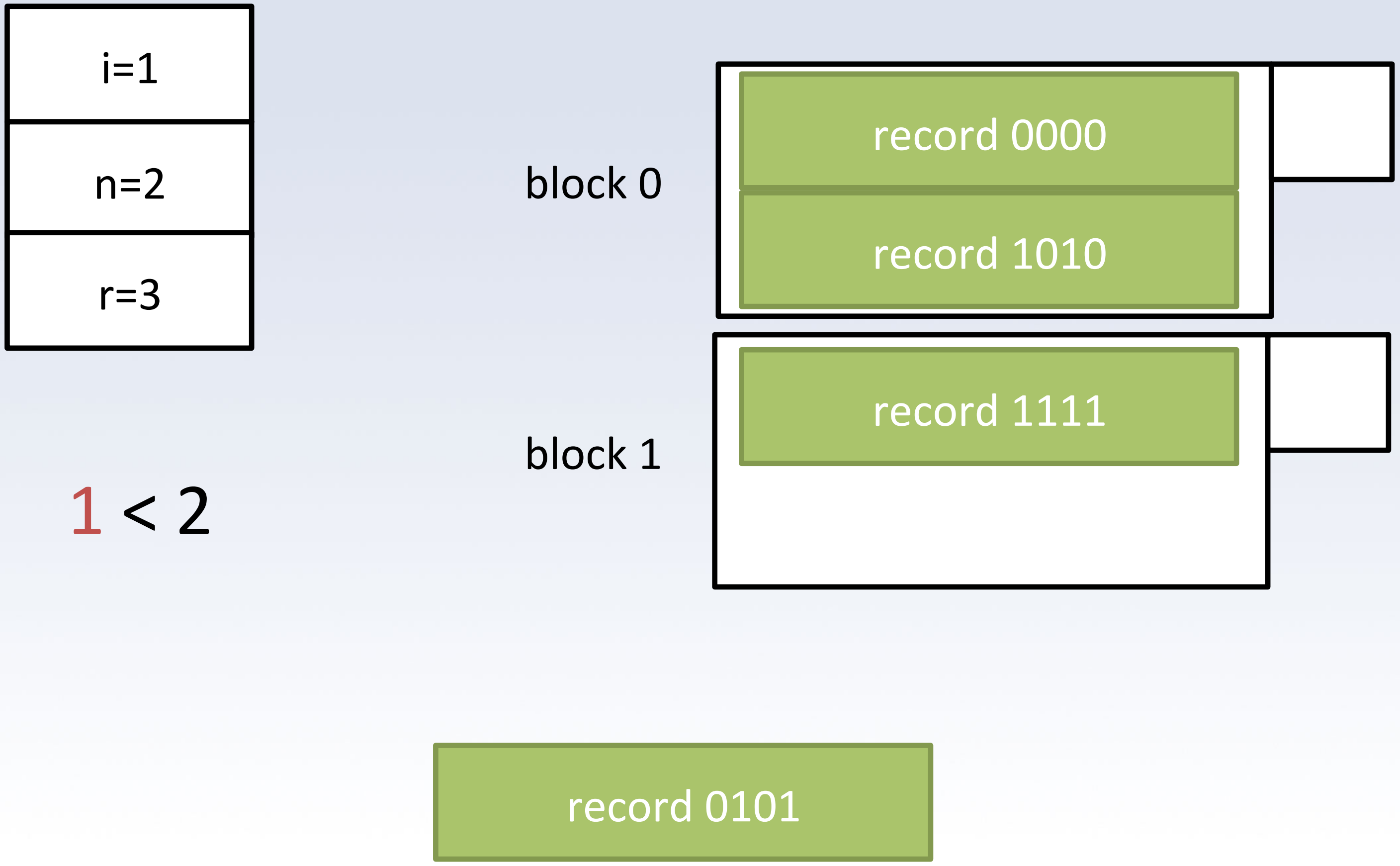
Linear Hash Insertion

- Lookup correct bucket
  - If room, insert. If not, create overflow.
  - If r/n is too big, add a new bucket
    - New bucket number is 1x. Split bucket 0x.
    - If n is too big, increment i

Example: Insert 0101

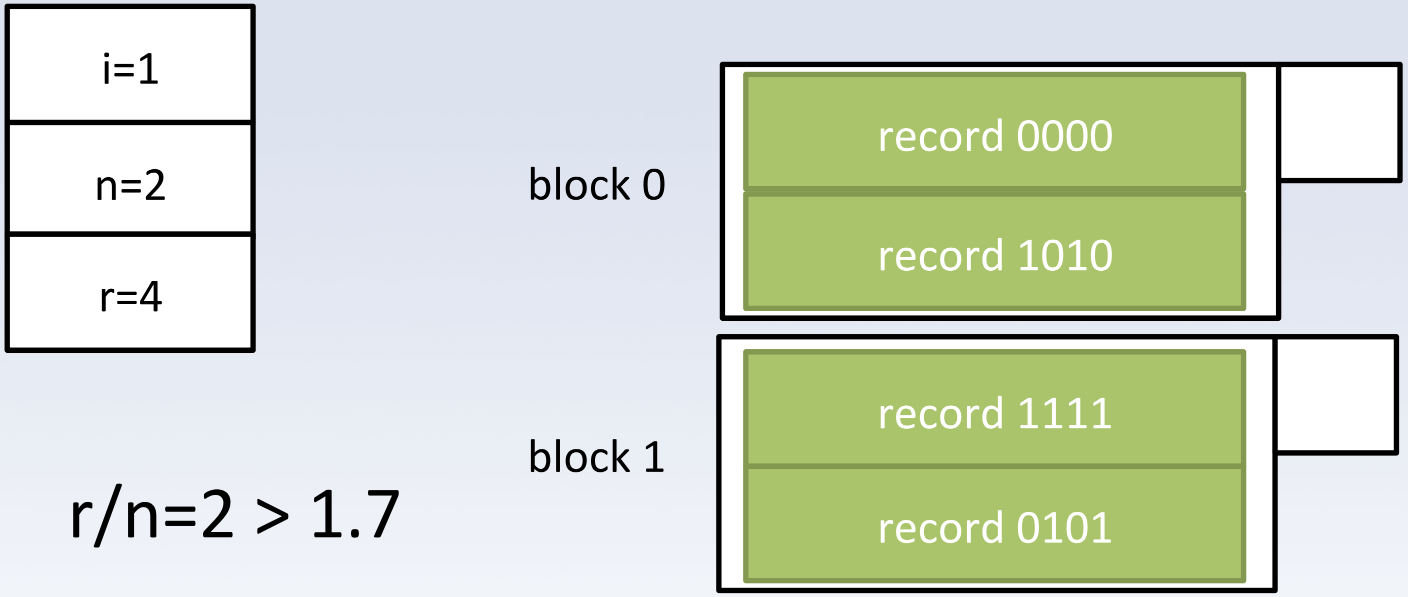


Example: Insert 0101



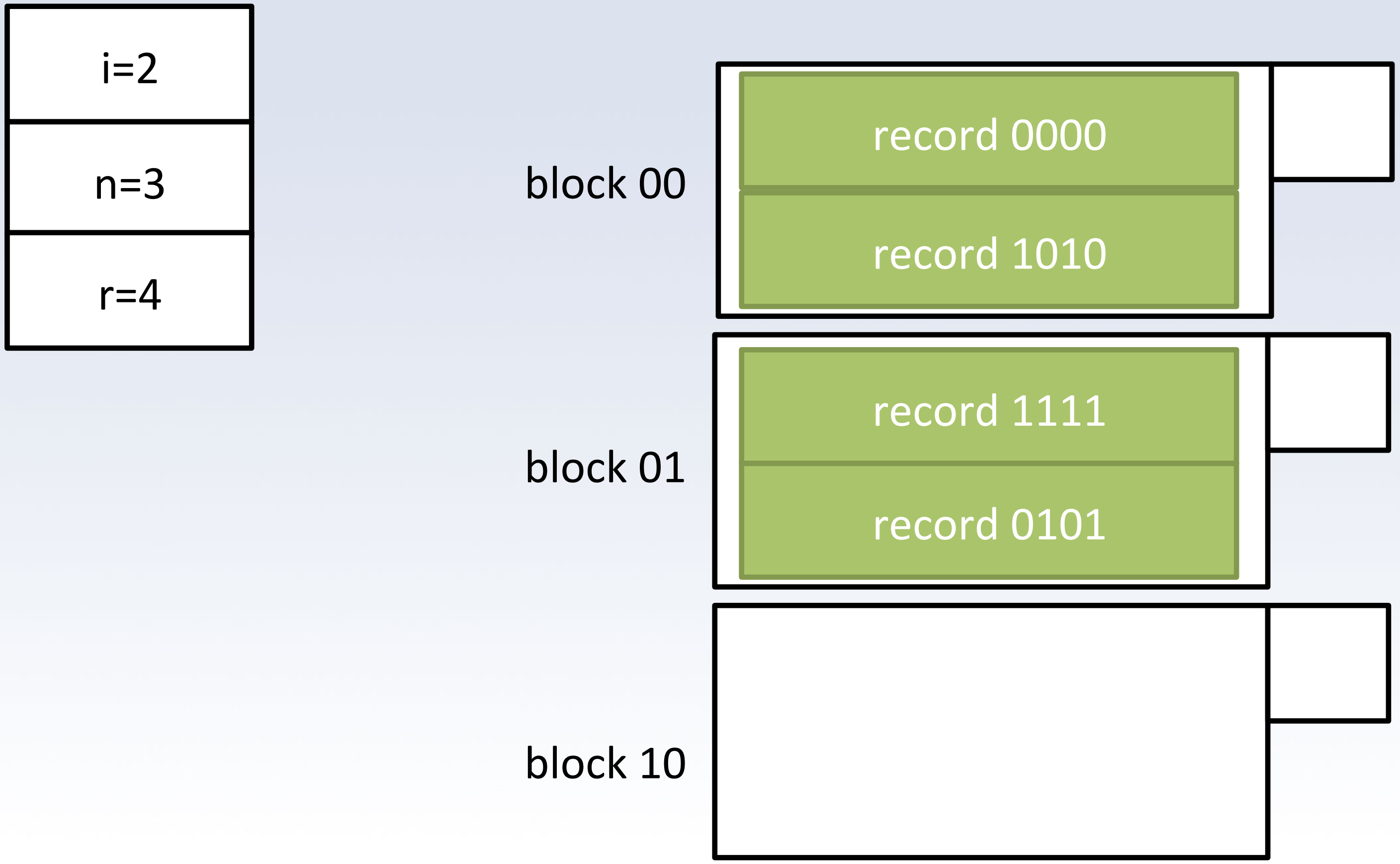
illinois.edu

Example: Insert 0101

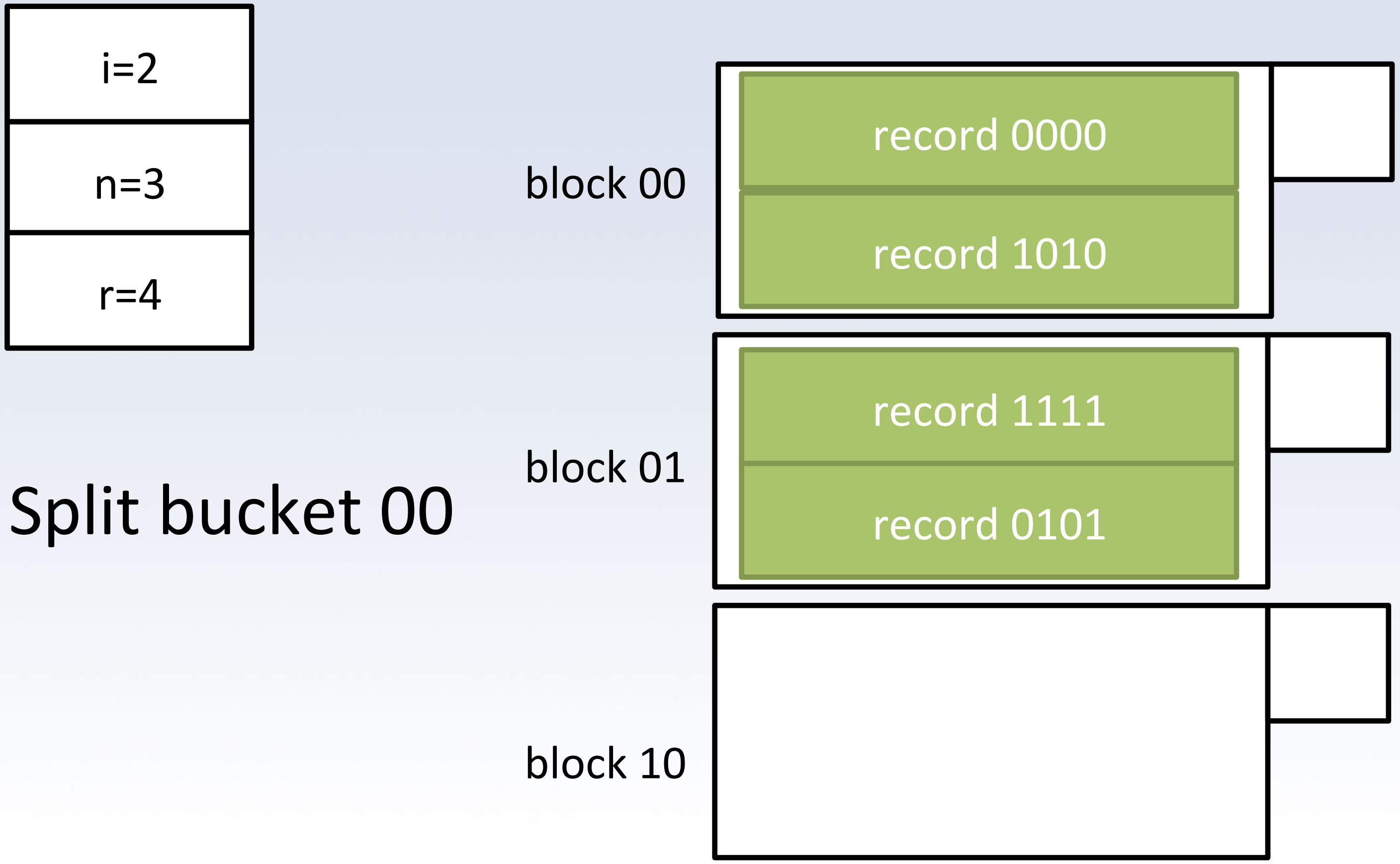


r/n=2 > 1.7

Example: Insert 0101



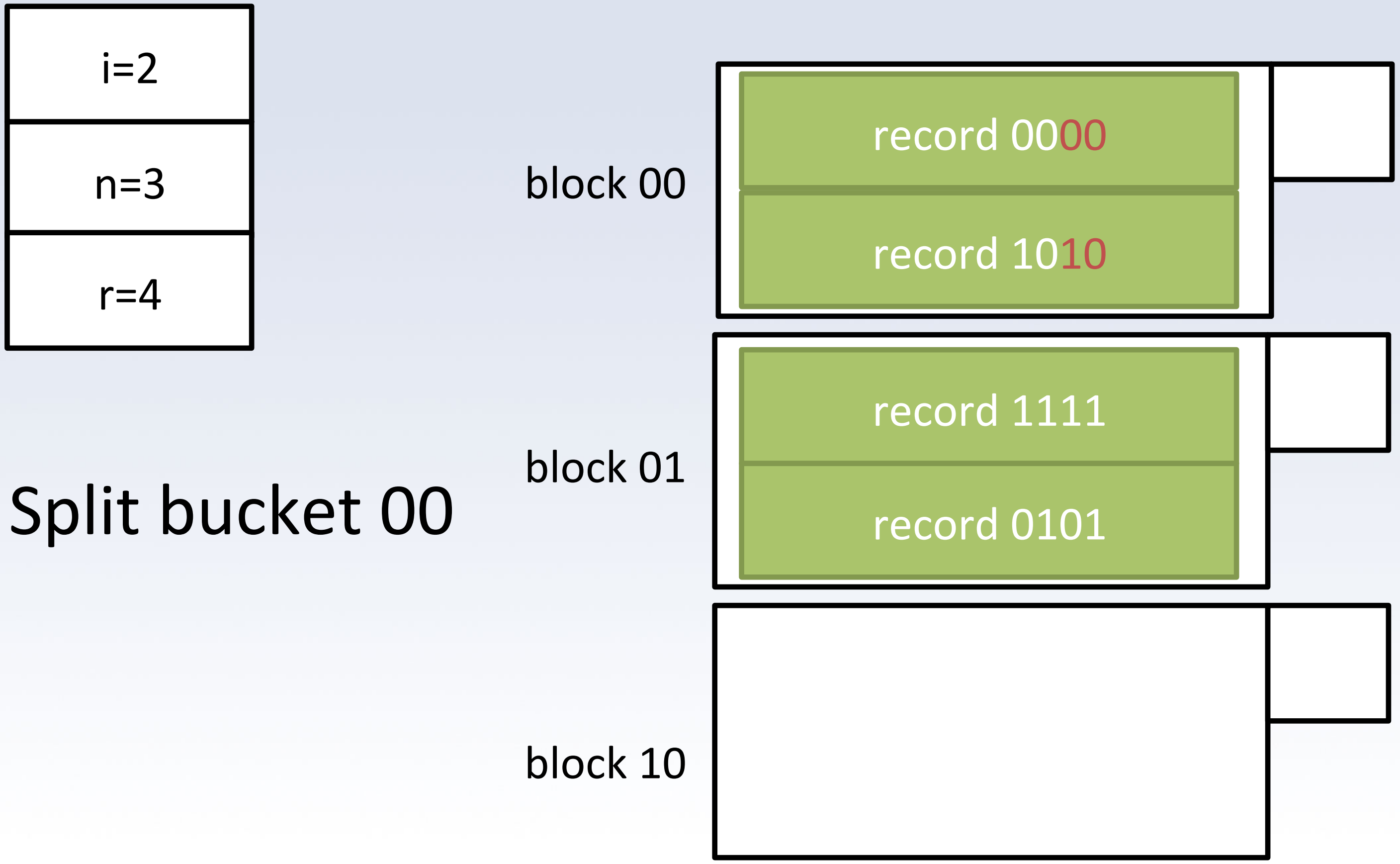
Example: Insert 0101



Split bucket 00

illinois.edu

Example: Insert 0101

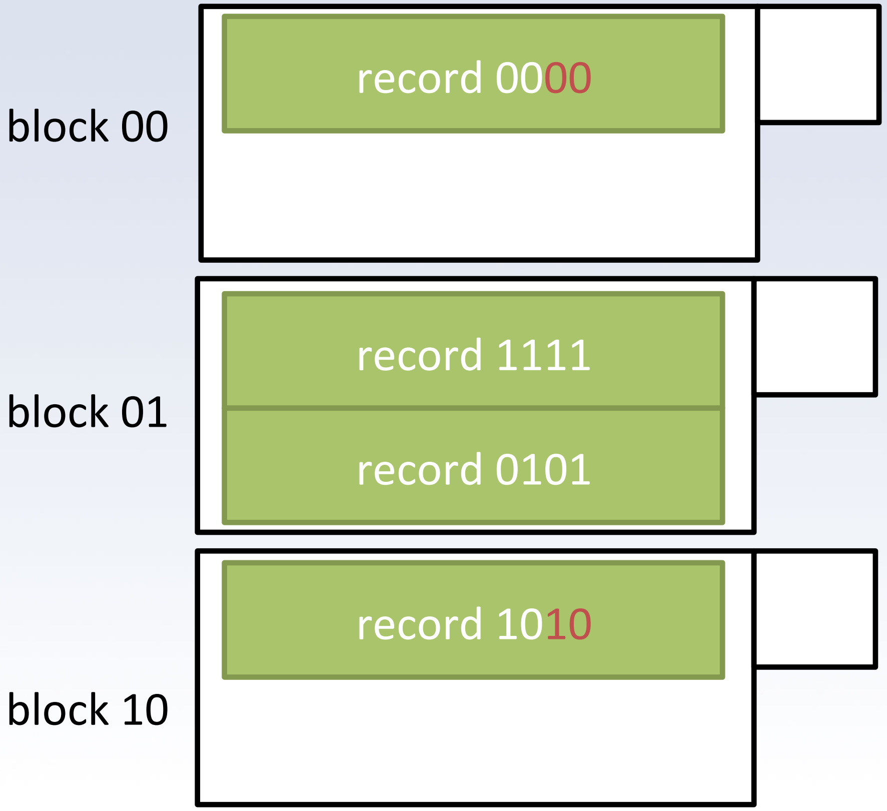


Split bucket 00

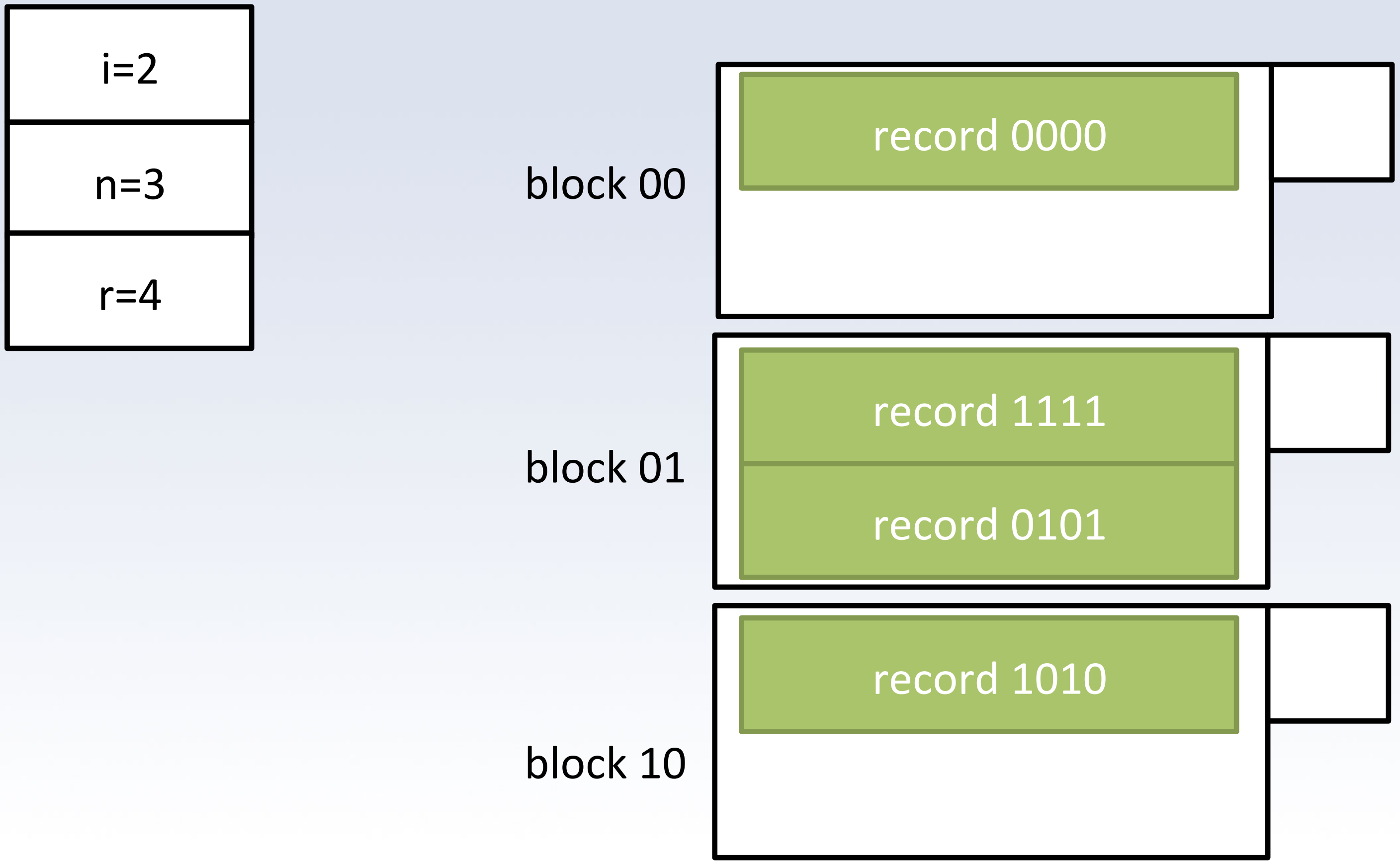
Example: Insert 0101

i=2
n=3
r=4

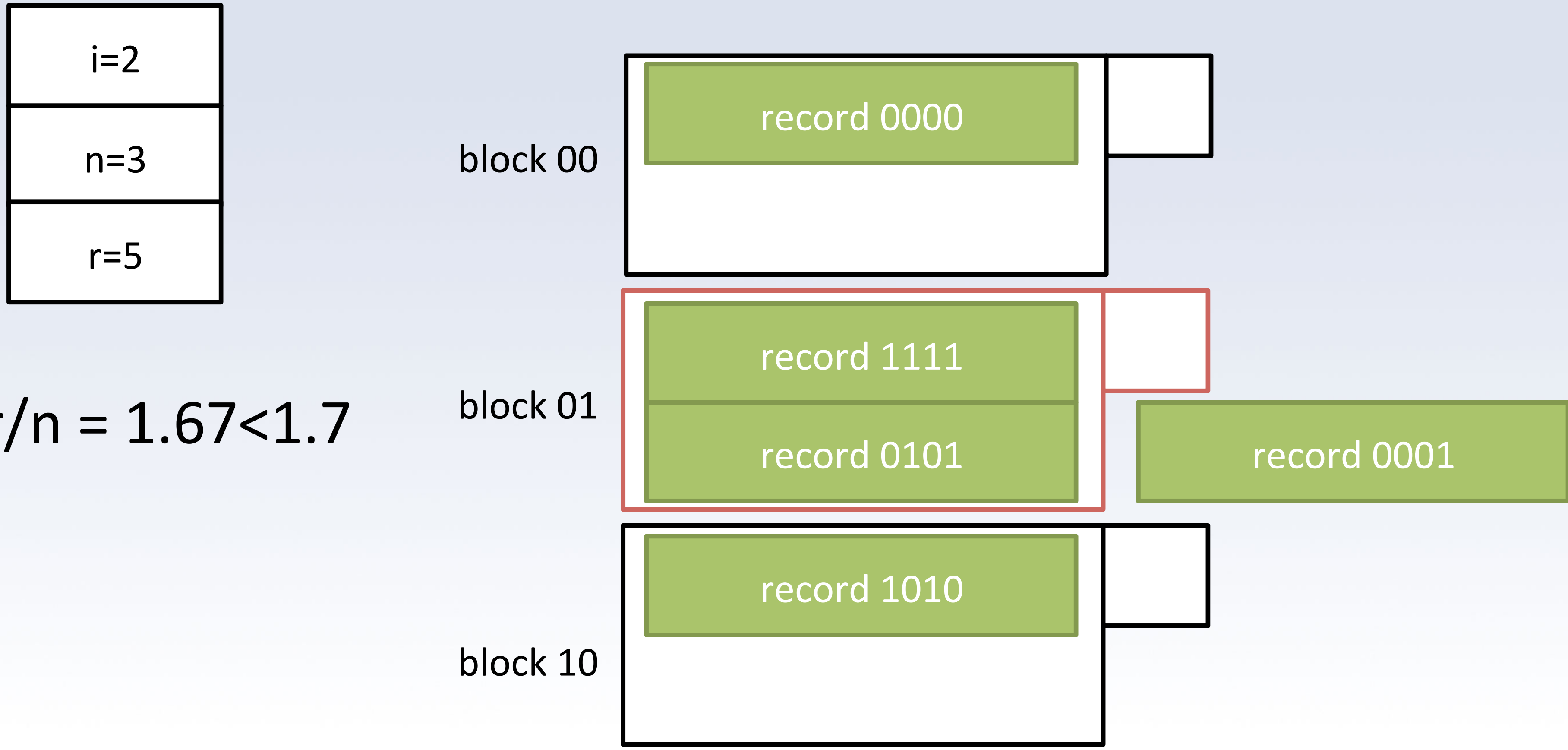
Split bucket 00



Example: Insert 0001



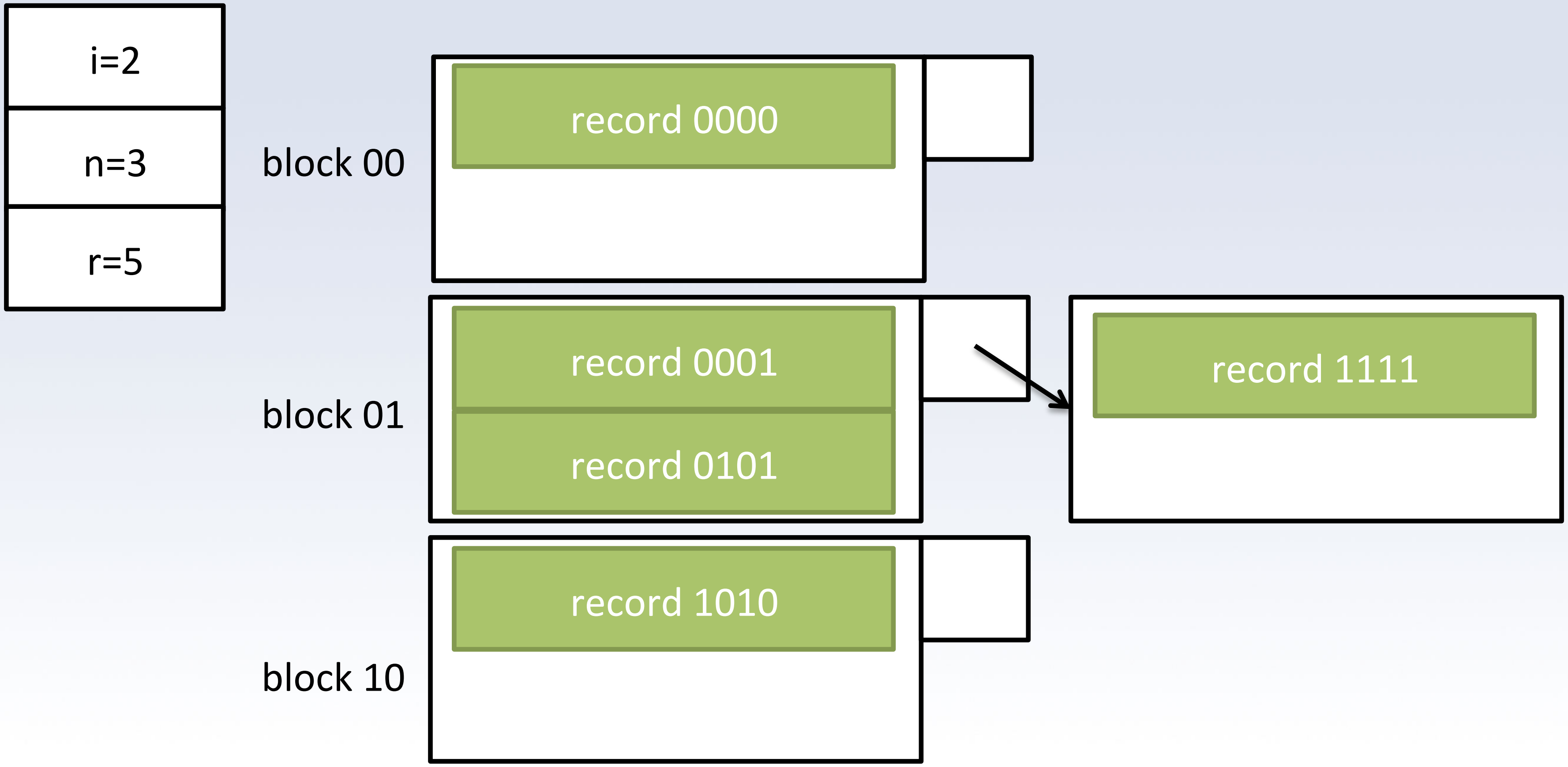
Example: Insert 0001



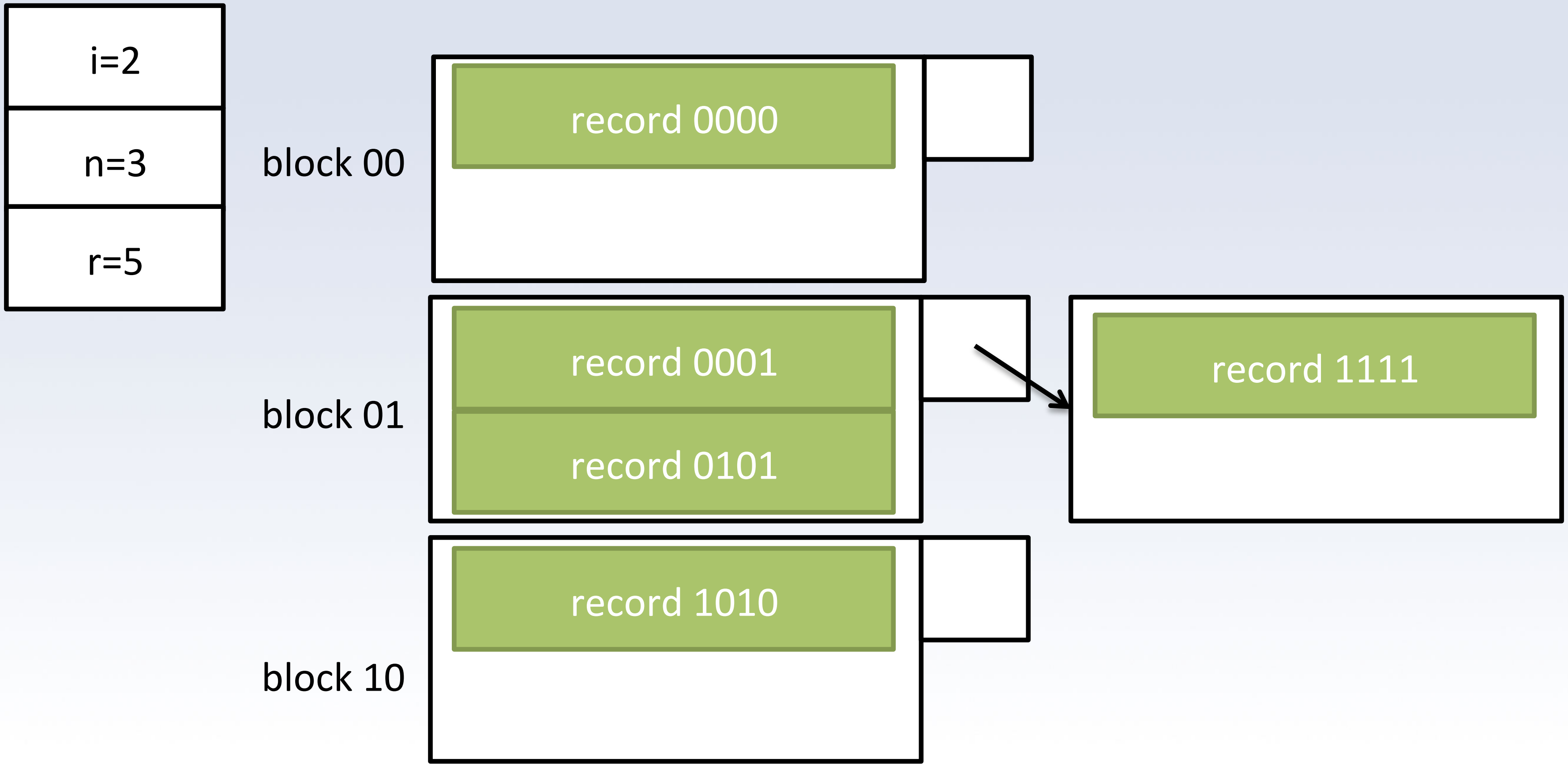
r/n = 1.67<1.7

illinois.edu

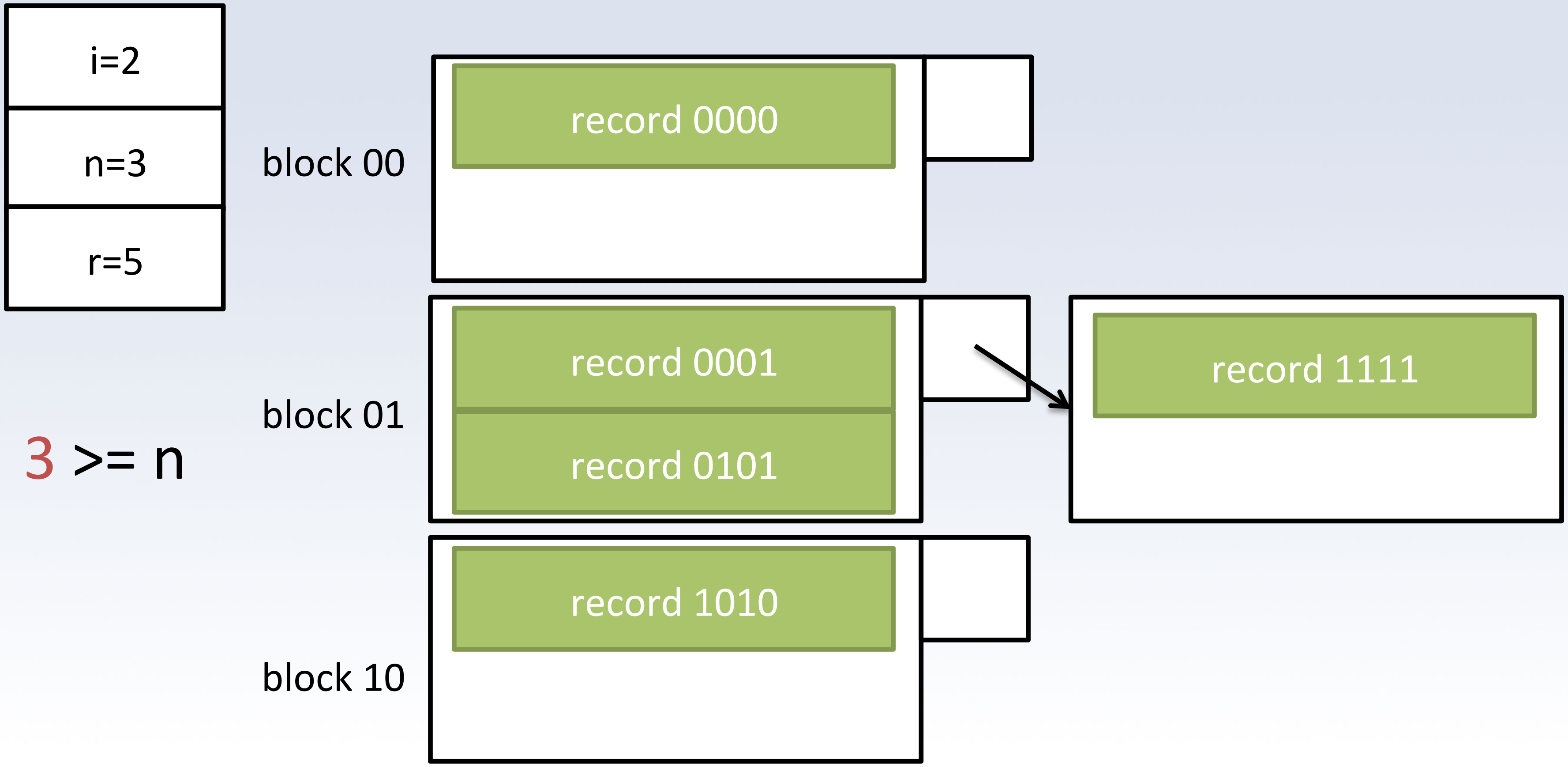
Example: Insert 0001



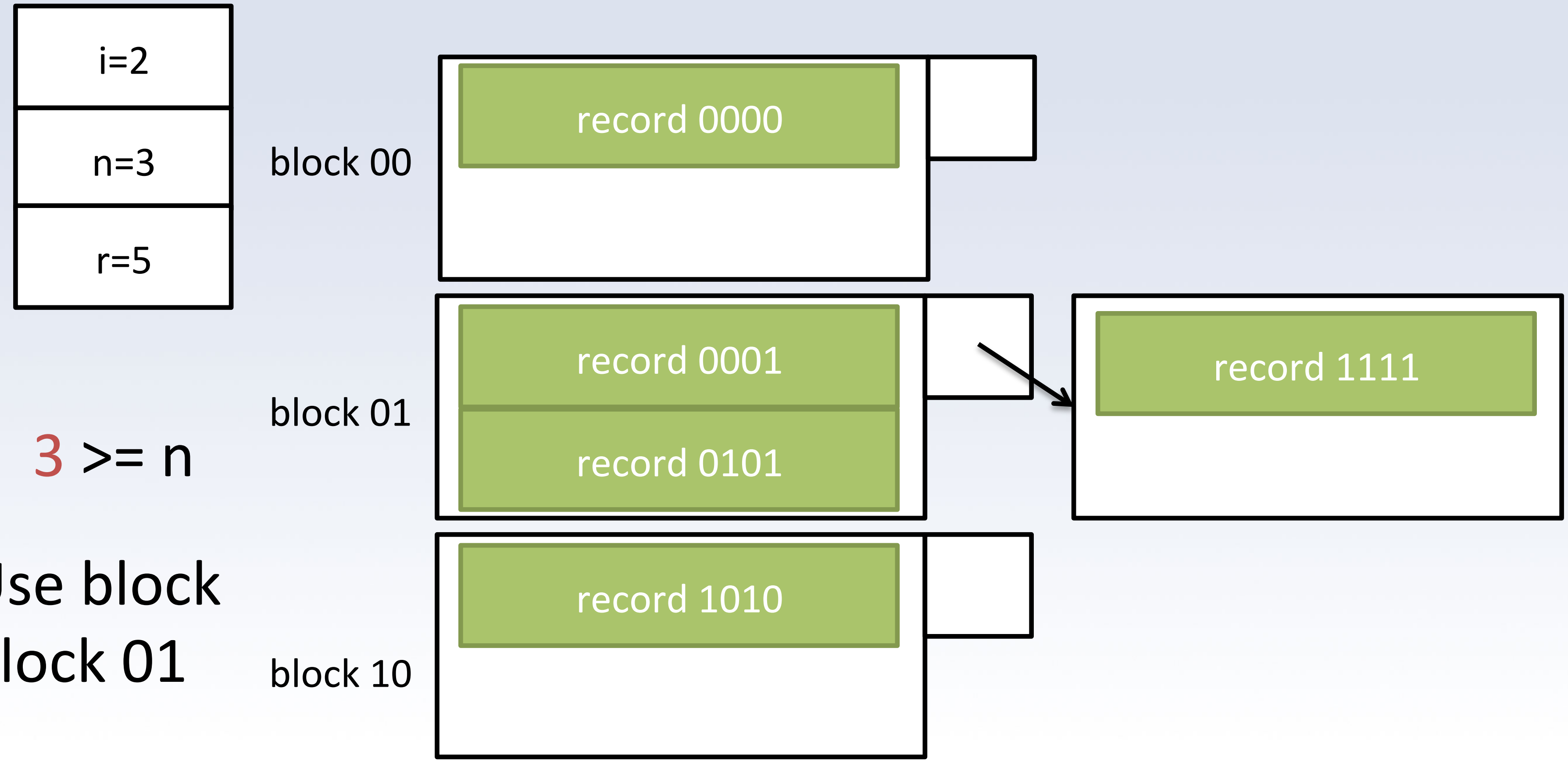
Example: Insert 0111



Example: Insert 0111



Example: Insert 0111

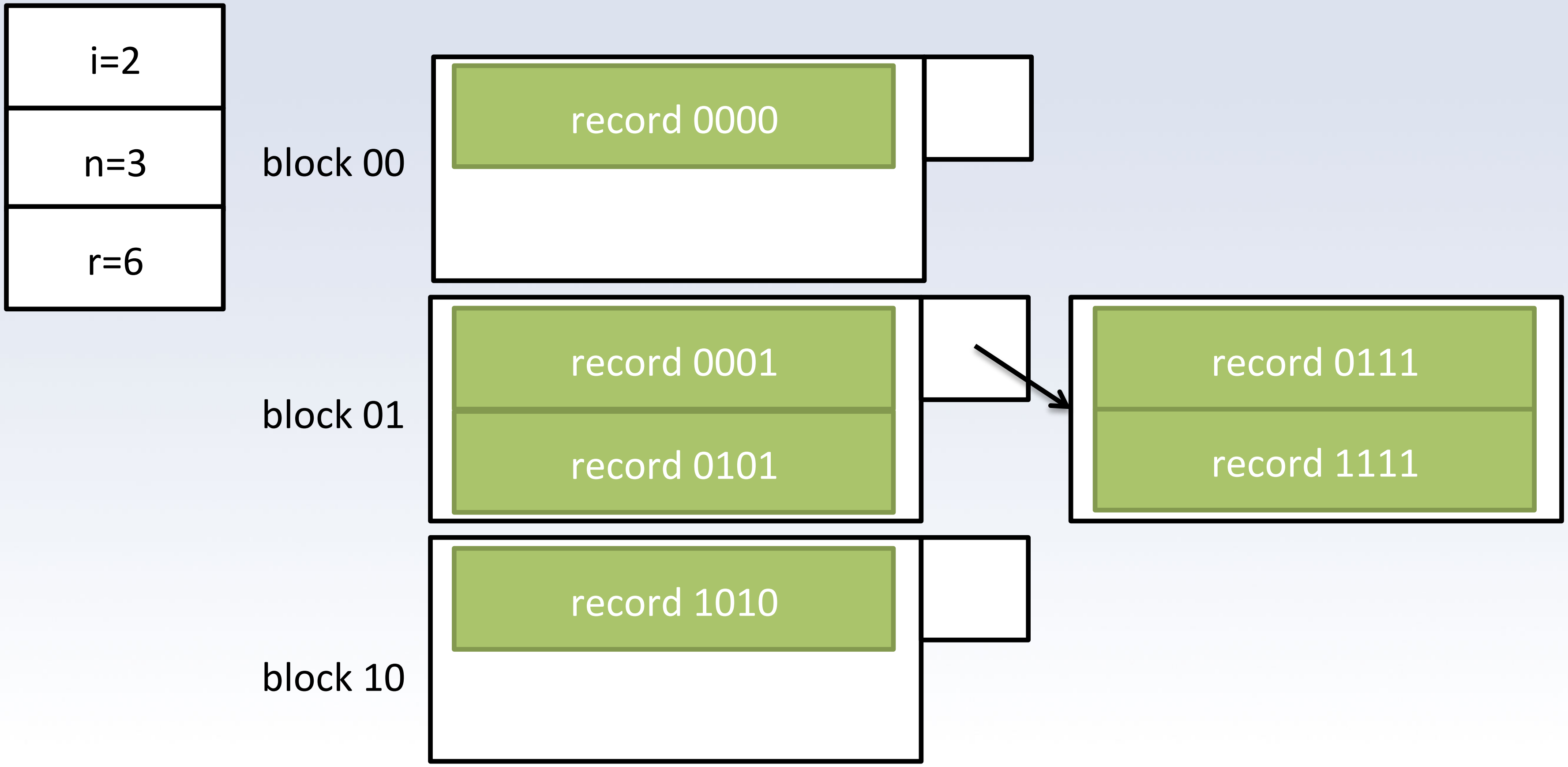


3 >= n

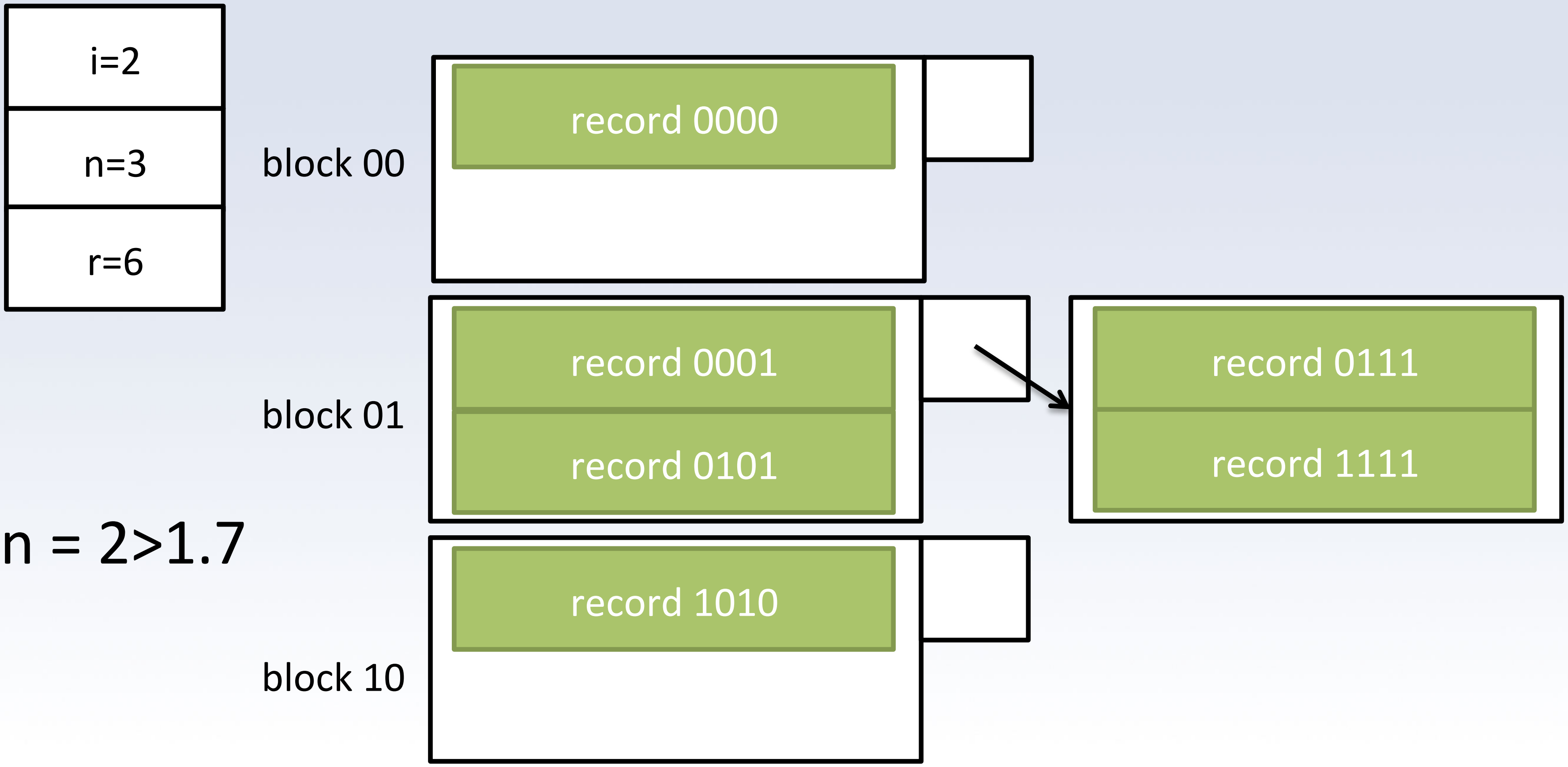
Use block block 01

illinois.edu

Example: Insert 0111

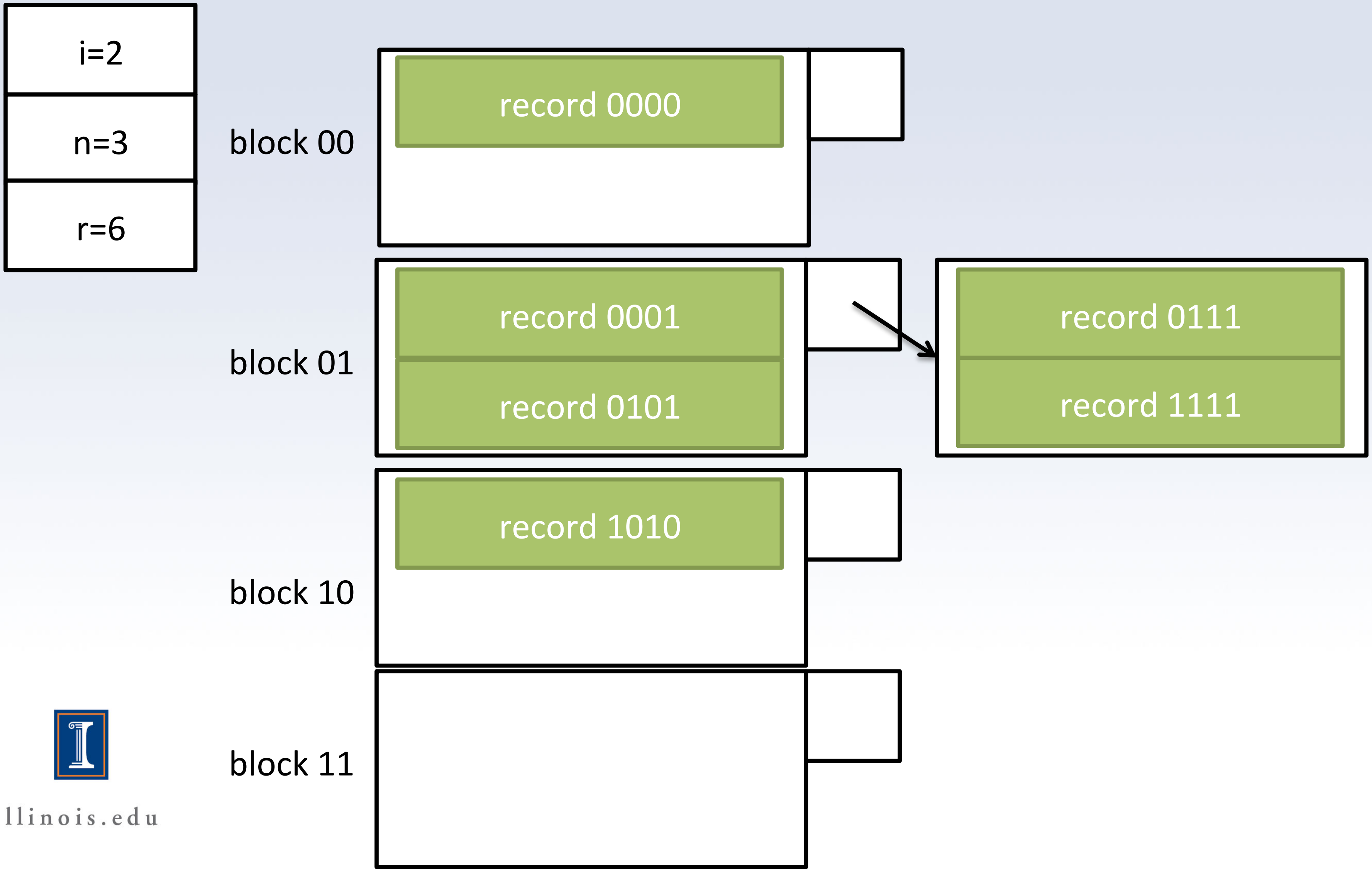


Example: Insert 0111

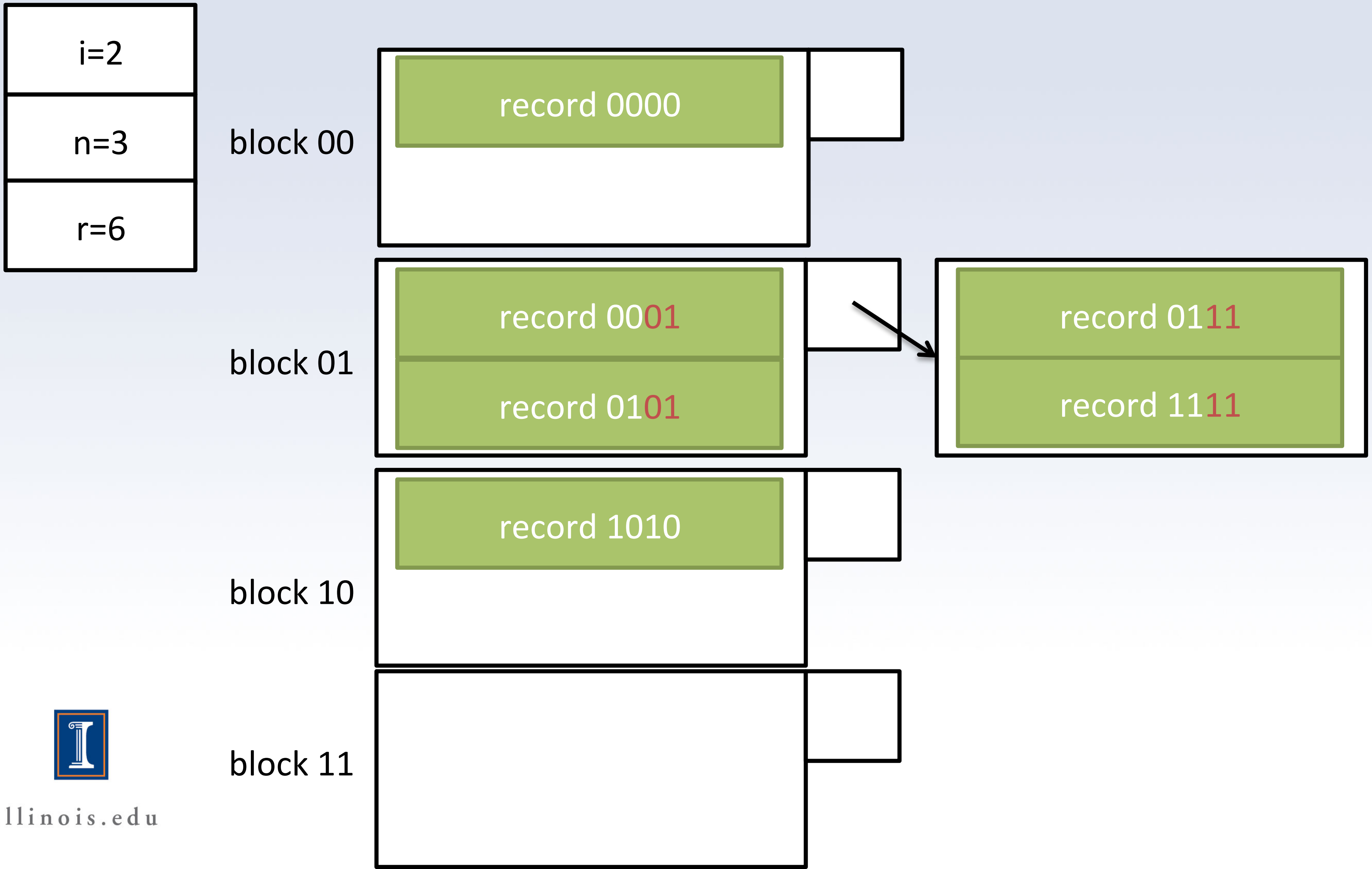


r/n = 2>1.7

Example: Insert 0111

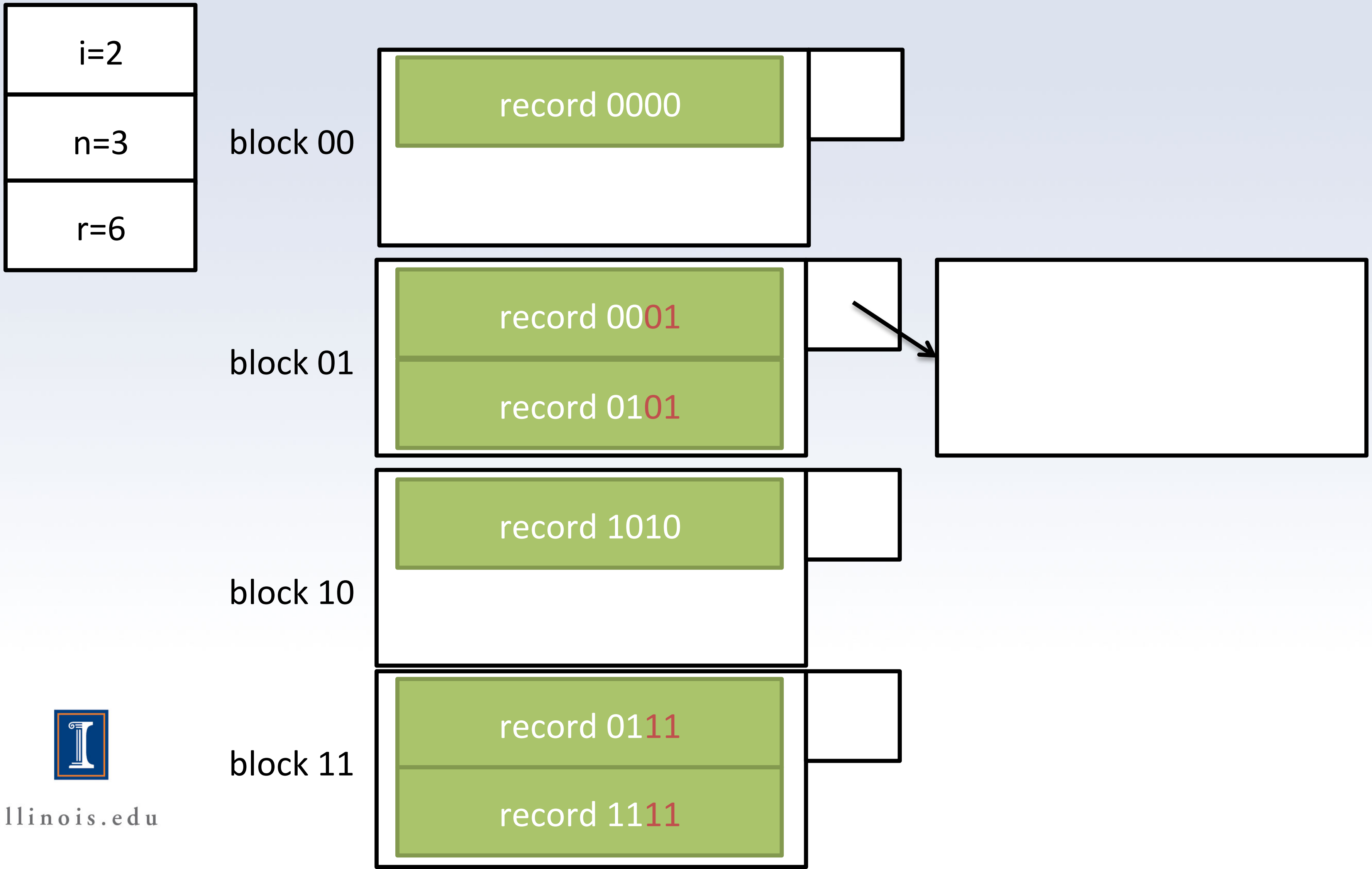


Example: Insert 0111

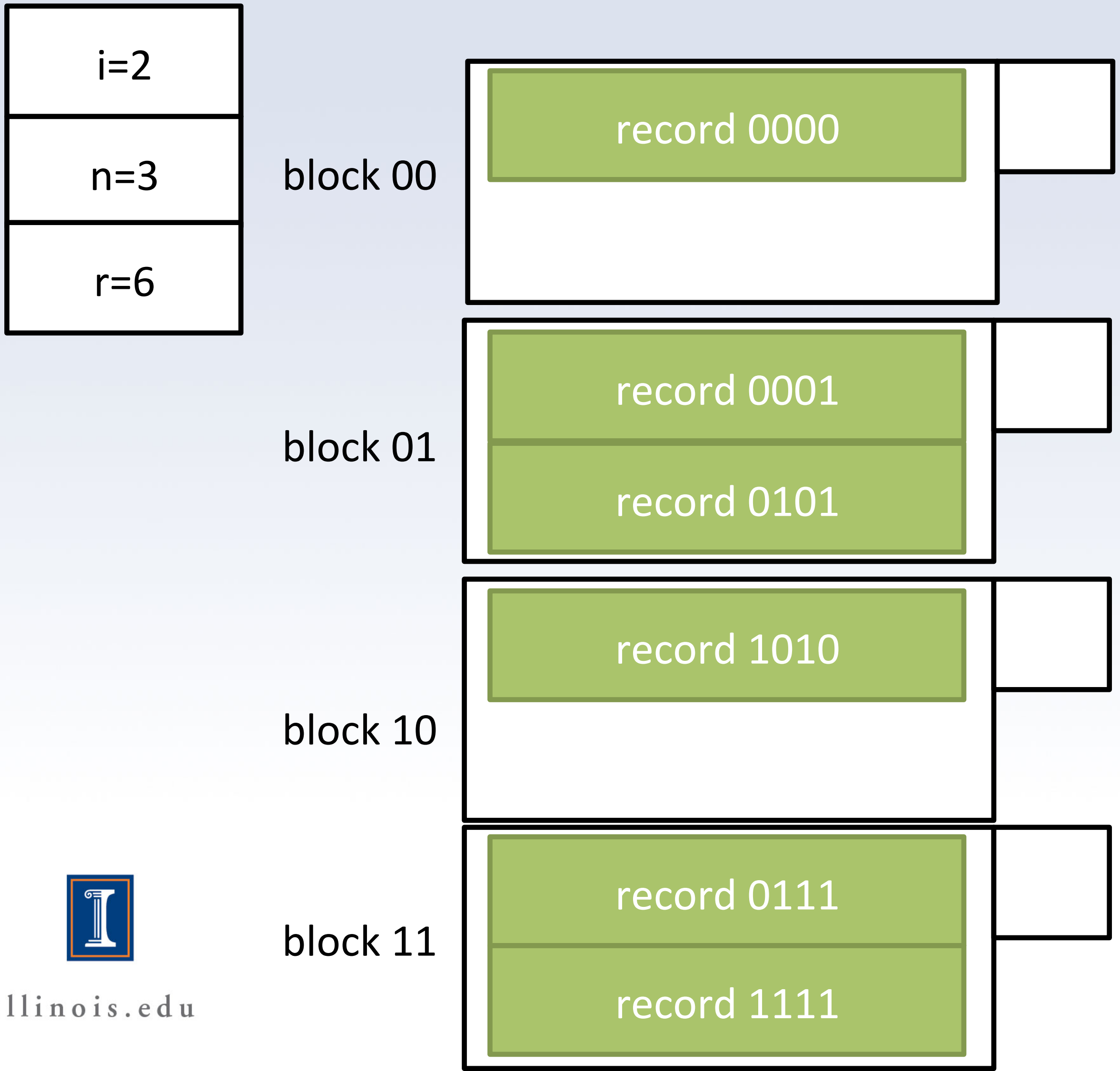


illinois.edu

Example: Insert 0111



Example: Insert 0111



illinois.edu

Hashing Strings

- Strings vary dramatically in length
- How can we hash them?
  - Use CRC
  - Use a rolling hash
  - Use cryptographic hash (e.g. MD5/SHA1)

Next time…

- We'll talk about
  - Multidimensional indexing (how Foursquare and other location based searching works)
  - Inverted indexing (how web search works)
  - Suffix arrays (how to index very large strings for search)