

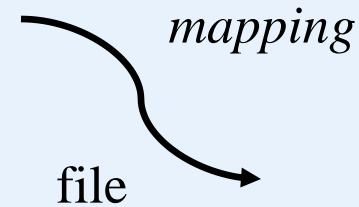
Outline: Hashing (5.9, 5.10, 3rd. ed.; 13.8, 4th, 5th ed.; 17.8, 6th ed.)

- external hashing
- static hashing & dynamic hashing
- hash function
 - mathematical function that maps a key to a bucket address
 - collisions
 - collision resolution scheme
 - open addressing
 - chaining
 - multiple hashing
- linear hashing

Mapping a table into a file

Employee

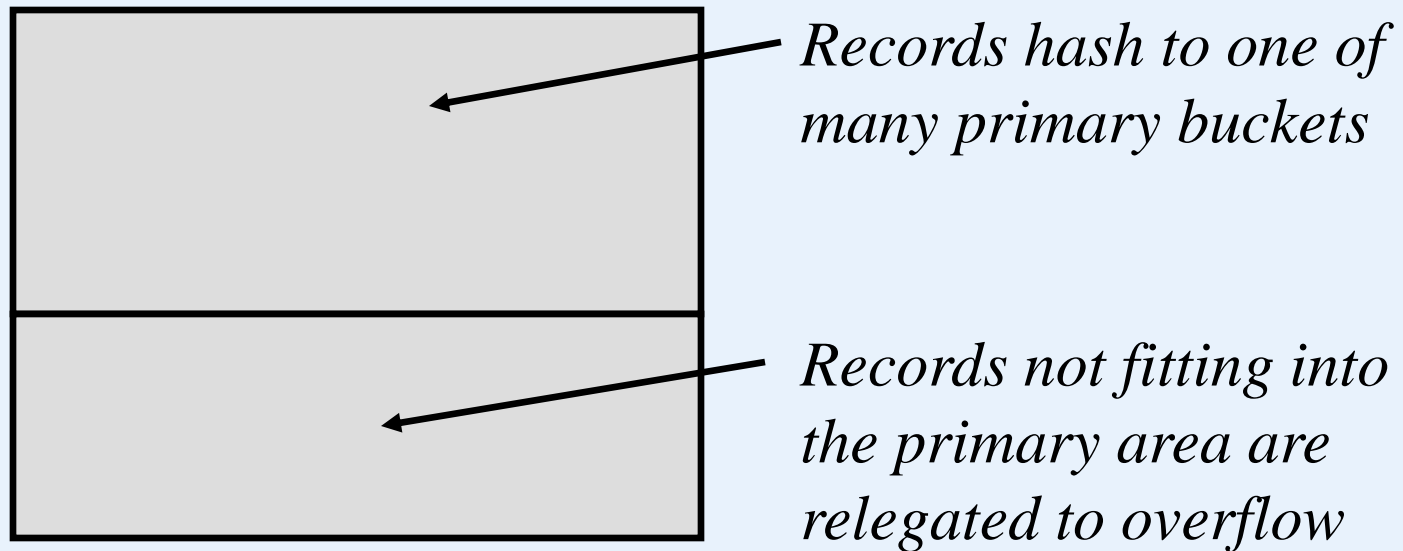
<u>ssn</u>	name	bdate	sex	address	salary
				



- Block (or page)
 - access unit of operating system
 - block size: range from 512 to 4096 bytes
- Bucket
 - access unit of database system
 - A bucket contains one or more blocks.
- A file can be considered as a collection of buckets.
Each bucket has an address.

External Hashing

- Consider a file comprising a primary area and an overflow area



- Common implementations are *static* - the number of primary buckets is fixed - and we expect to need to reorganize this type of files on a regular basis.

External Hashing

- Consider a static hash file comprising M primary buckets
- We need a hash function that maps the key onto $\{0, 1, \dots, M-1\}$
- If M is prime and Key is numeric then

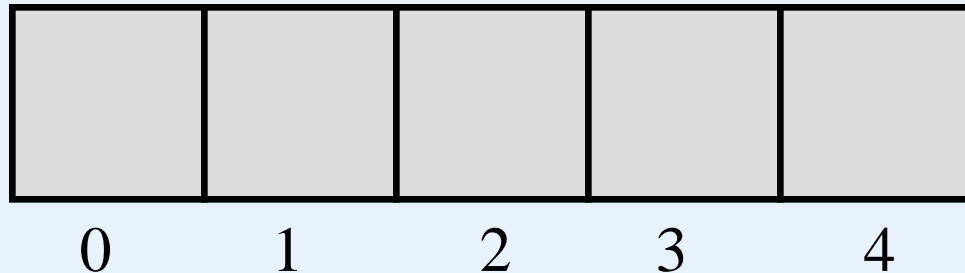
$$Hash(Key) = Key \bmod M$$

can work well

- A collision may occur when more than one records hash to the same address
- We need a collision resolution scheme for overflow handling because the number of collisions for one primary bucket can exceed the bucket capacity
 - open addressing
 - chaining

Overflow handling

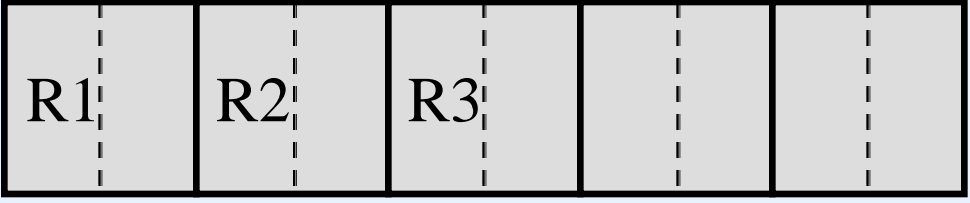
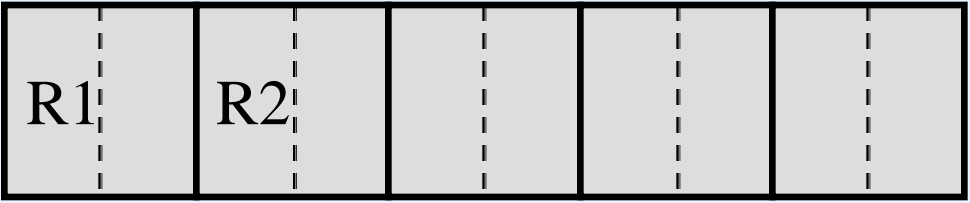
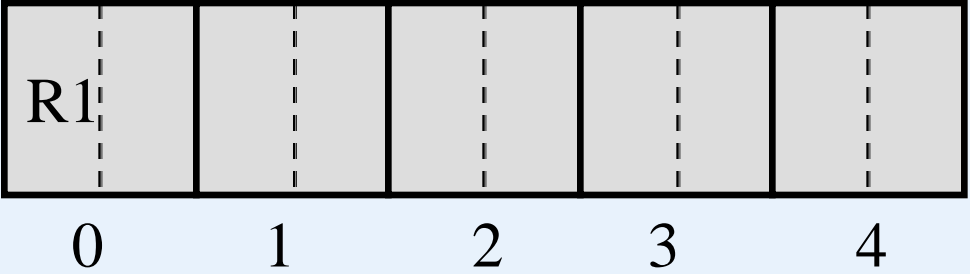
- Open addressing
 - subsequent buckets are examined until an open record position is found
 - no need for an overflow area
 - consider records being inserted R1, R2, R3, R4, R5, R6, R7 with bucket capacity of 2 and hash values 0, 1, 2, 1, 1, 0, 3



How do we handle retrieval, deletion?

File Organizations

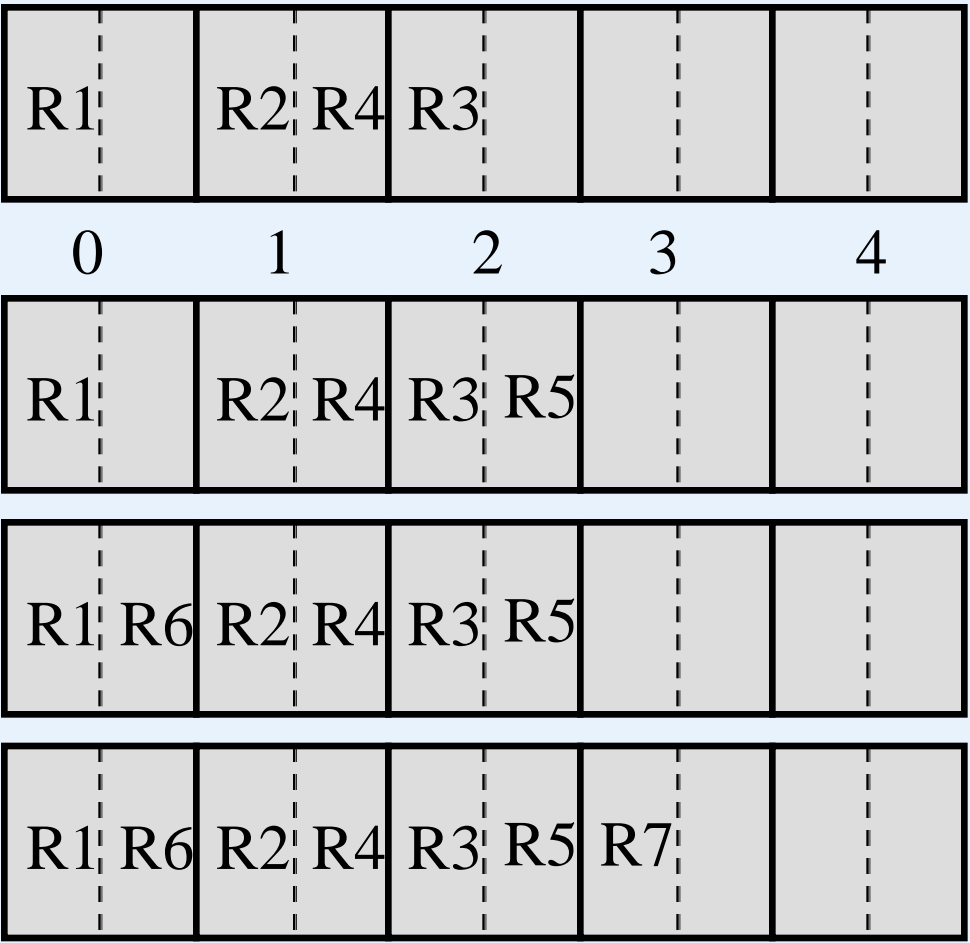
- consider records being inserted R1, R2, R3, R4, R5, R6, R7 with bucket capacity of 2 and hash values 0, 1, 2, 1, 1, 0, 3



File Organizations

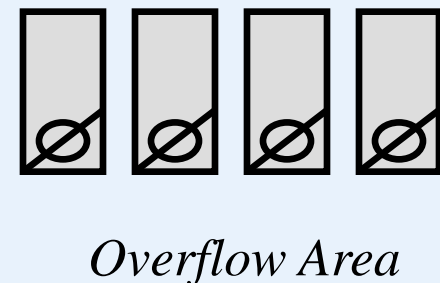
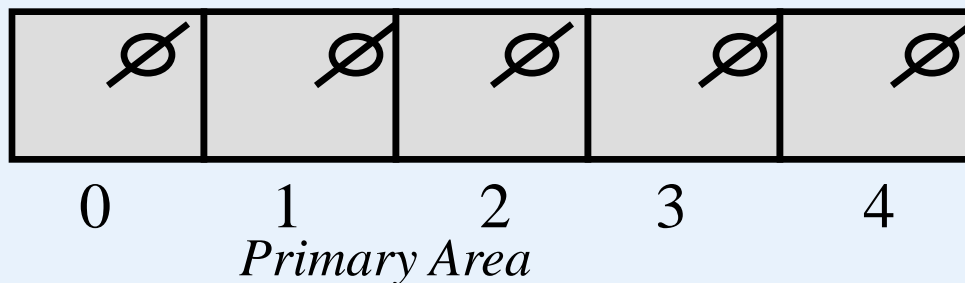
R1, R2, R3, R4, R5, R6, R7

hash values: 0, 1, 2, 1, 1, 0, 3



Overflow handling

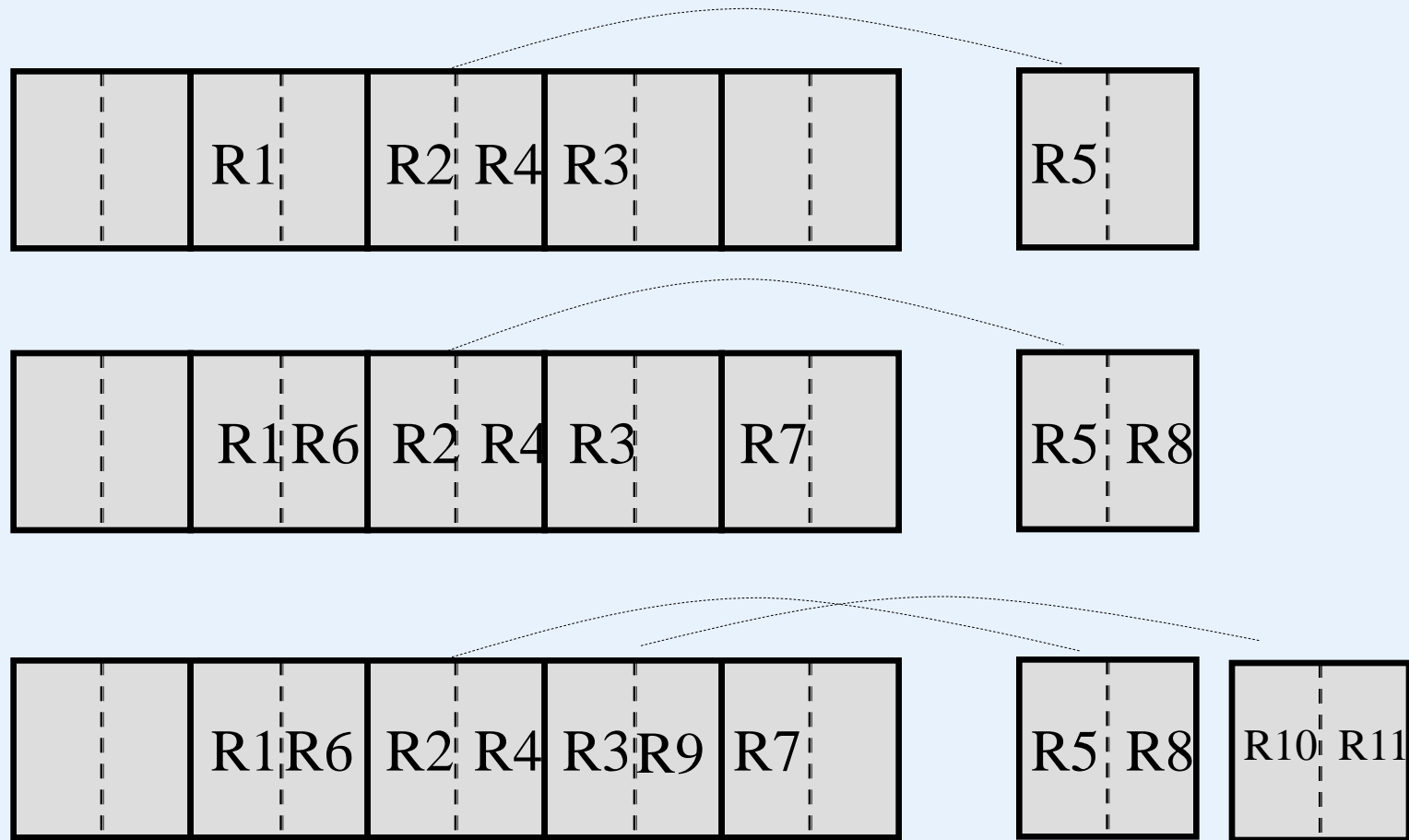
- Chaining
 - a pointer in the primary bucket points to the first overflow record
 - overflow records for one primary bucket are chained together
 - consider records being inserted R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11.
 - with bucket capacity of 2 and hash values 1, 2, 3, 2, 2, 1, 4, 2, 3, 3, 3.
 - deletions?



File Organizations

R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11

1, 2, 3, 2, 2, 1, 4, 2, 3, 3, 3



Overflow handling

- Multiple Hashing
 - when collision occurs a next hash function is tried to find an unfilled bucket
 - eventually we would resort to chaining
 - note that open addressing can suffer from poor performance due to islands of full buckets occurring and having a tendency to get even longer - using a second hash function helps avoid that problem

Linear Hashing

- A dynamic hash file:
 - grows and shrinks gracefully
- initially the hash file comprises M primary buckets numbered $0, 1, \dots, M-1$
- the hashing process is divided into several phases (phase 0, phase 1, phase 2, ...). In phase j , records are hashed according to hash functions $h_j(\text{key})$ and $h_{j+1}(\text{key})$
- $h_j(\text{key}) = \text{key} \bmod (2^j * M)$

phase 0: $h_0(\text{key}) = \text{key} \bmod (2^0 * M)$, $h_1(\text{key}) = \text{key} \bmod (2^1 * M)$

phase 1: $h_1(\text{key}) = \text{key} \bmod (2^1 * M)$, $h_2(\text{key}) = \text{key} \bmod (2^2 * M)$

phase 2: $h_2(\text{key}) = \text{key} \bmod (2^2 * M)$, $h_3(\text{key}) = \text{key} \bmod (2^3 * M)$

... ..

Linear Hashing

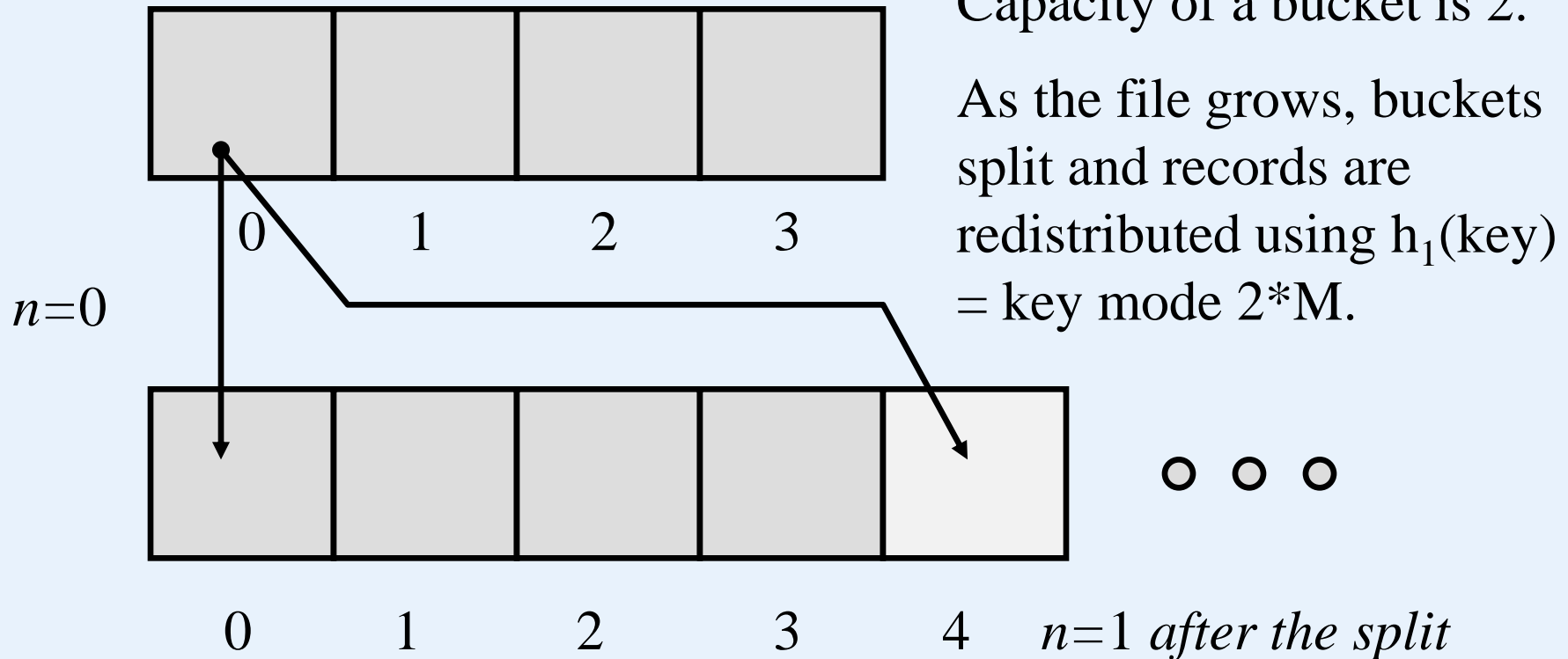
- $h_j(\text{key})$ is used first; to split, use $h_{j+1}(\text{key})$
- splitting a bucket means to redistribute the records into two buckets: the original one and a new one. In phase j , to determine which ones go into the original while the others go into the new one, we use $h_{j+1}(\text{key}) = \text{key} \bmod 2^{j+1} * M$ to calculate their address.
- splitting buckets
 - splitting occurs according to a specific rule such as
 - an overflow occurring, or
 - the load factor reaching a certain value, etc.
- a split pointer keeps track of which bucket to split next
- split pointer goes from 0 to $2^j * M - 1$ during the j^{th} phase, $j = 0, 1, 2, \dots$

Linear Hashing

1. What is a phase?
2. When to split a bucket?
3. How to split a bucket?
4. What bucket will be chosen to split next?
5. How do we find a record inserted into a linear hashing file?

Linear Hashing, example

- initially suppose $M=4$
- $h_0(\text{key}) = \text{key} \bmod M$; i.e. $\text{key} \bmod 4$ (rightmost 2 bits)
- $h_1(\text{key}) = \text{key} \bmod 2*M$

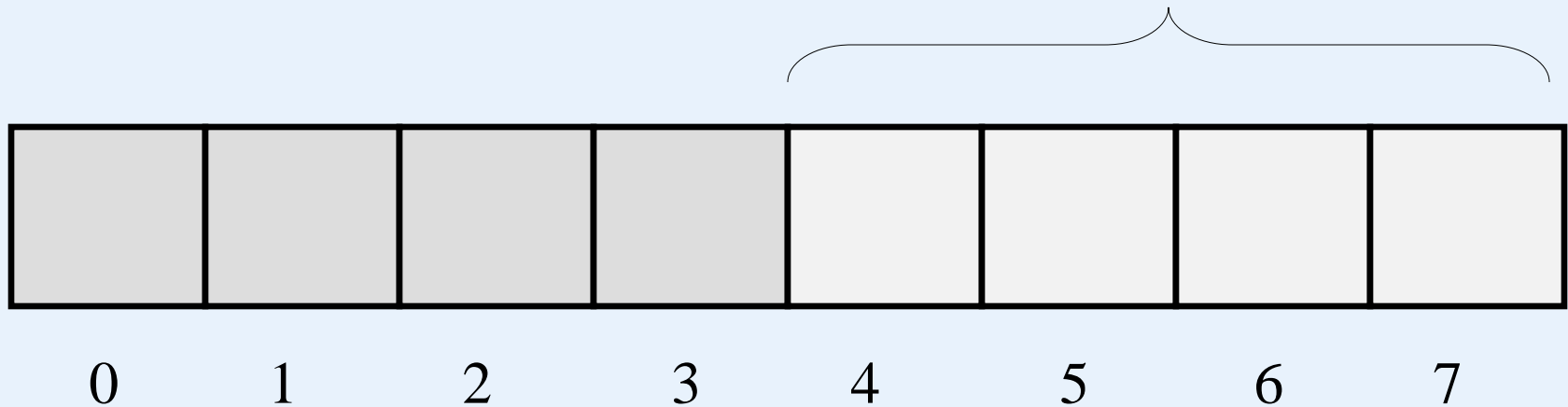


Linear Hashing, example

- collision resolution strategy: chaining
- split rule: if load factor > 0.70
- insert the records with key values:

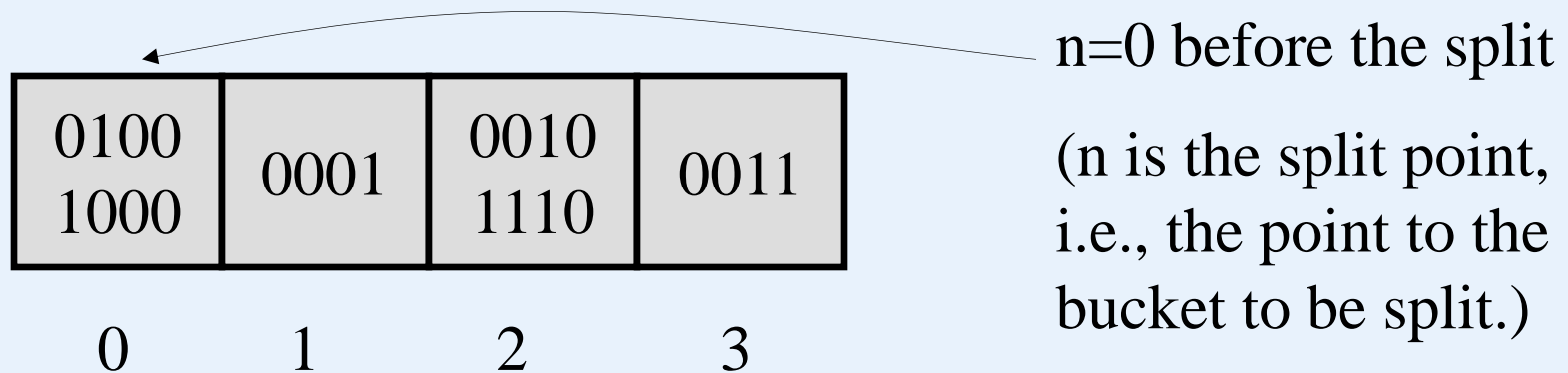
0011, 0010, 0100, 0001, 1000, 1110, 0101, 1010, 0111, 1100

Buckets to be added during the expansion



Linear Hashing, example

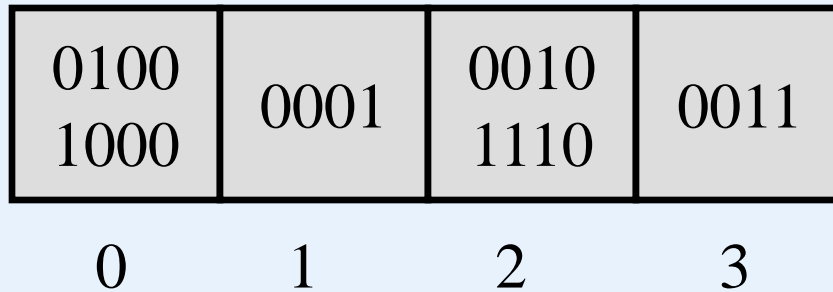
- when inserting the sixth record (using $h_0 = \text{Key} \bmod M$) we would have



0011, 0010, 0100, 0001, 1000, 1110, 0101, 1010, 0111, 1100

Linear Hashing, example

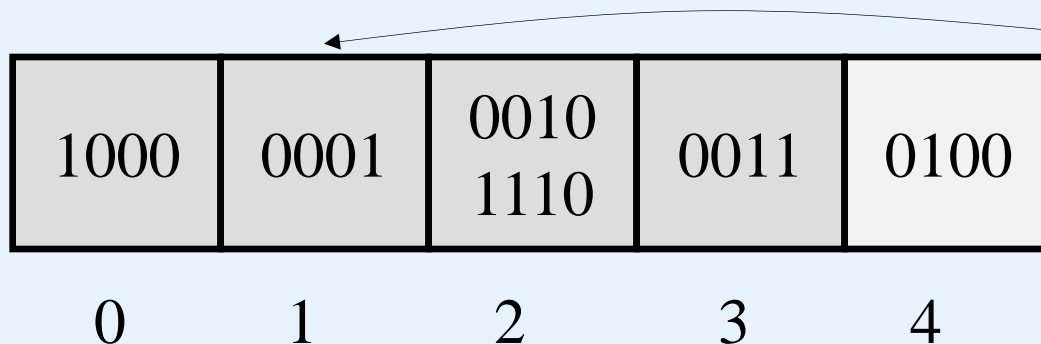
- when inserting the sixth record (using $h_0 = \text{Key} \bmod M$) we would have



$n=0$ before the split

(n is the point to the bucket to be split.)

- but the load factor $6/8 = 0.75 > 0.70$ and so bucket 0 must be split (using $h_1 = \text{Key} \bmod 2M$):

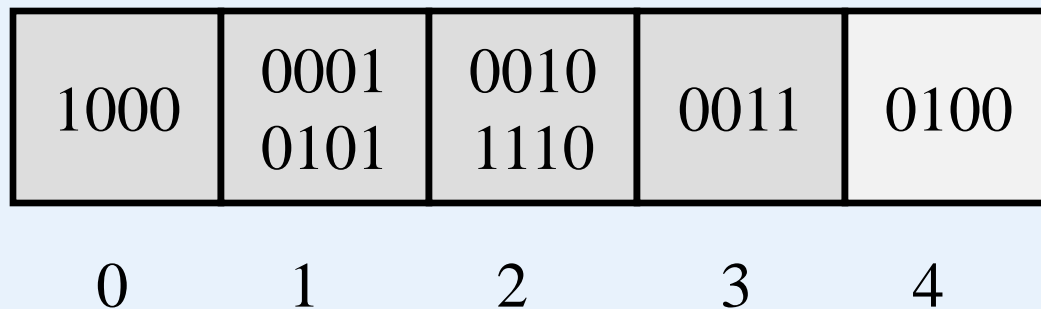
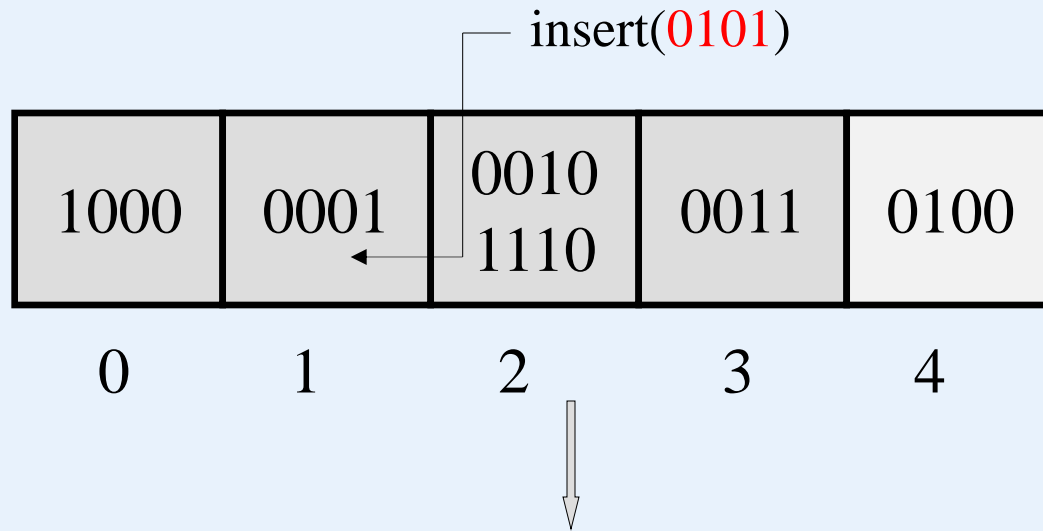


$n=1$ after the split

load factor: $6/10 = 0.6$

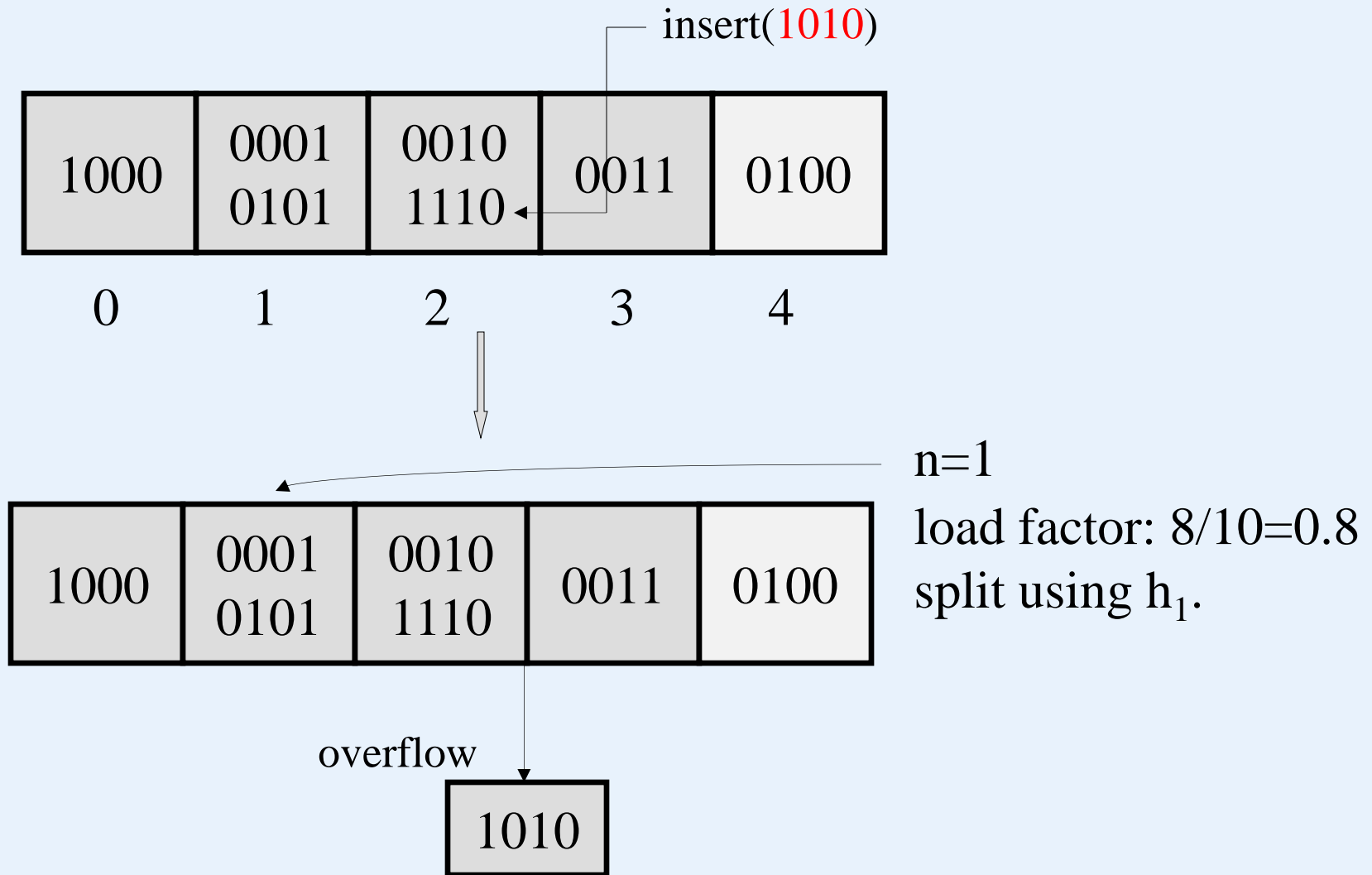
no split

Linear Hashing, example

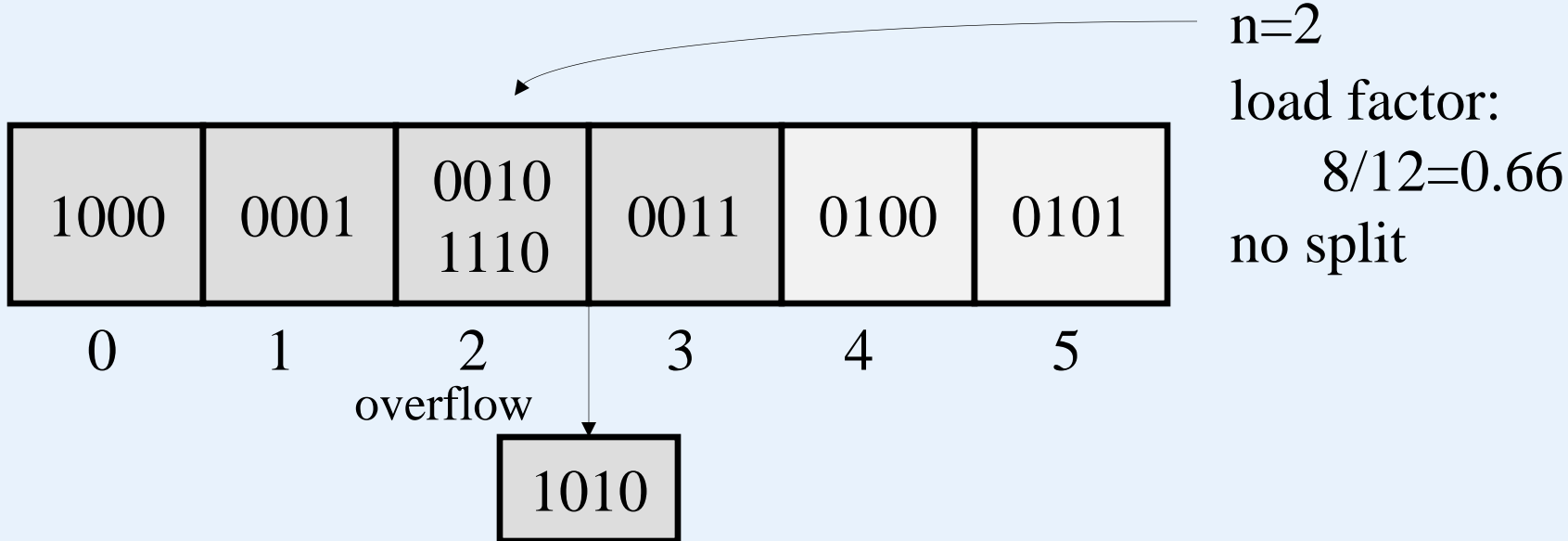


$n=1$
 load factor: $7/10=0.7$
 no split

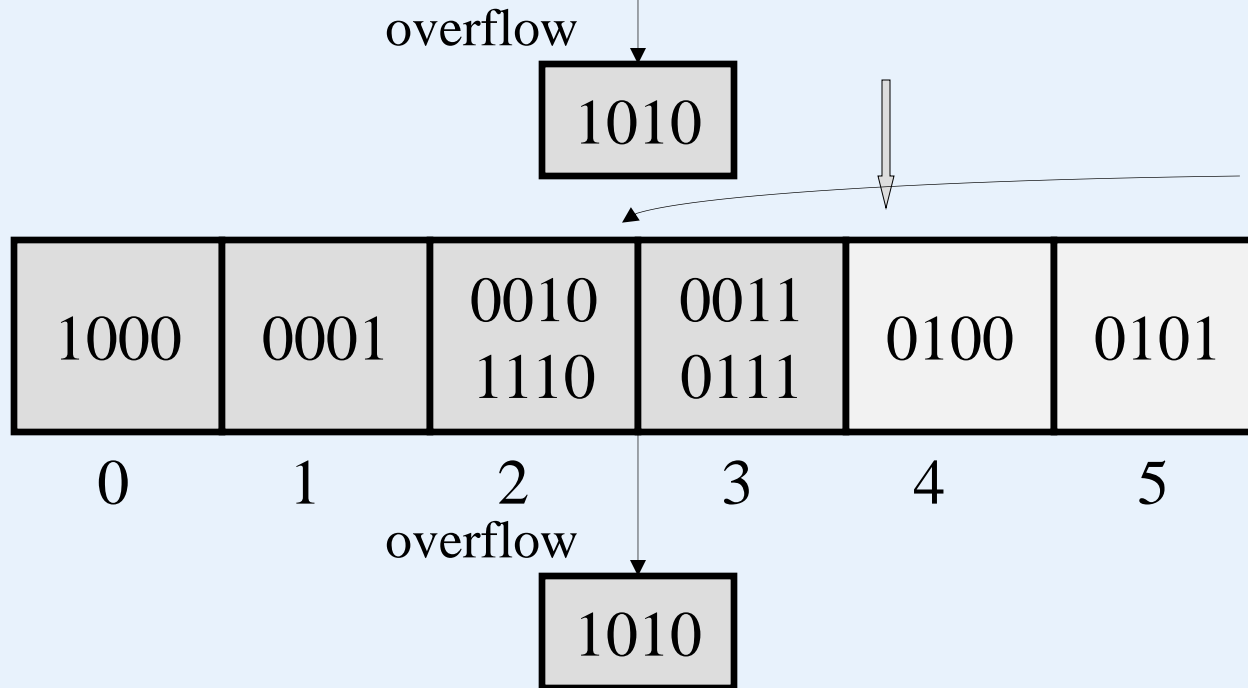
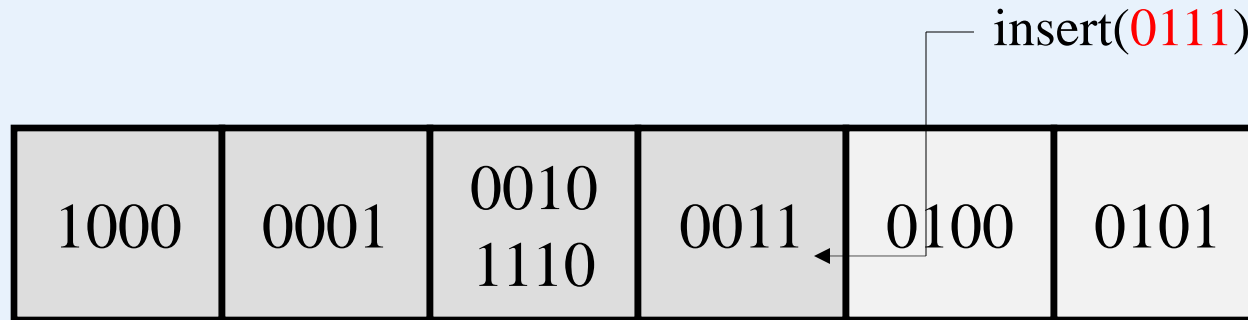
Linear Hashing, example



Linear Hashing, example



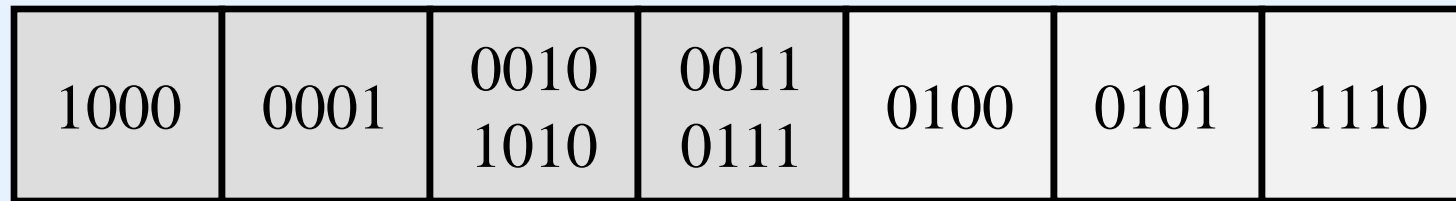
Linear Hashing, example



$n=2$

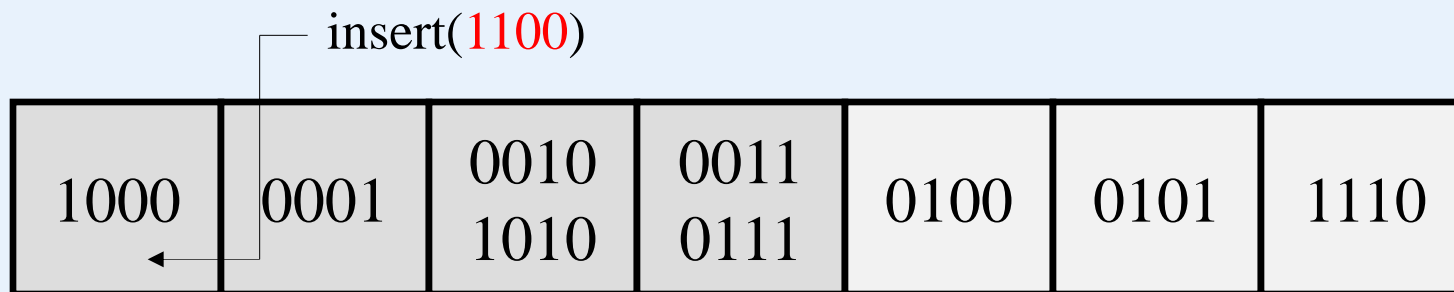
load factor:
 $9/12=0.75$
 split using h_1 .

Linear Hashing, example



↑
n=3

load factor: $9/14=0.642$
no split.



Linear Hashing, example

1000 1100	0001	0010 1010	0011 0111	0100	0101	1110
--------------	------	--------------	--------------	------	------	------

↑
n=3

load factor: $10/14=0.71$
split using h_1 .

1000 1100	0001	0010 1010	0011 0111	0100	0101	1110	0111
--------------	------	--------------	--------------	------	------	------	------

Linear Hashing, example

1000 1100	0001	0010 1010	0011	0100	0101	1110	0111
--------------	------	--------------	------	------	------	------	------

↑
n=4

load factor: $10/16=0.625$
no split.

- At this point, all the 4 (M) buckets are split. The size of the primary area becomes 2M. n should be set to 0. It begins a second phase.
- In the second phase, we will use h_1 to insert records and h_2 to split a bucket.
 - note that $h_1(K) = K \bmod 2M$ and $h_2(K) = K \bmod 4M$.

Linear Hashing including two Phases:

- collision resolution strategy: chaining
- split rule: load factor > 0.7
- initially $M = 4$ (M : size of the primary area)
- hash functions: $h_i(\text{key}) = \text{key} \bmod 2^i \times M$ ($i = 0, 1, 2, \dots$)
- bucket capacity = 2

Trace the insertion process of the following keys into a linear hashing file:

3, 2, 4, 1, 8, 14, 5, 10, 7, 24, 17, 13, 15.

The first phase – phase₀

- when inserting the sixth record we would have

4	1	2	3
8		14	
0	1	2	3

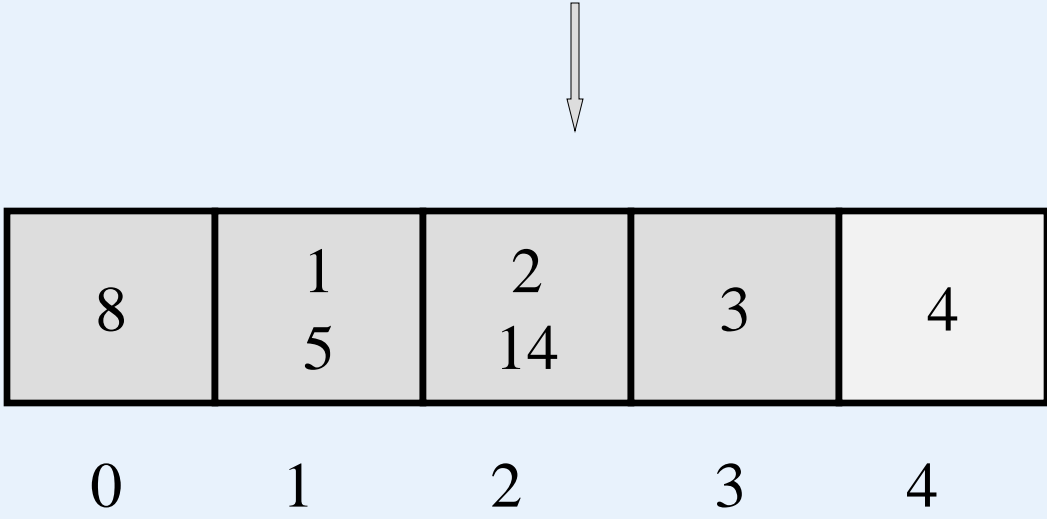
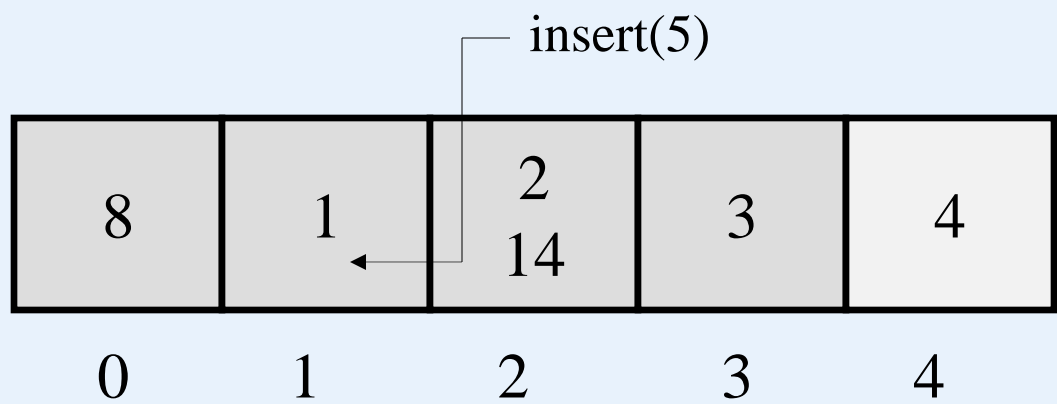
$n=0$ before the split
(n is the point to the bucket to be split.)

- but the load factor $6/8 = 0.75 > 0.70$ and so bucket 0 must be split (using $h_1 = \text{Key mod } 2M$):

8	1	2	3	4
		14		
0	1	2	3	4

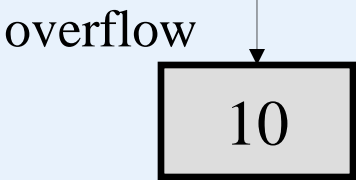
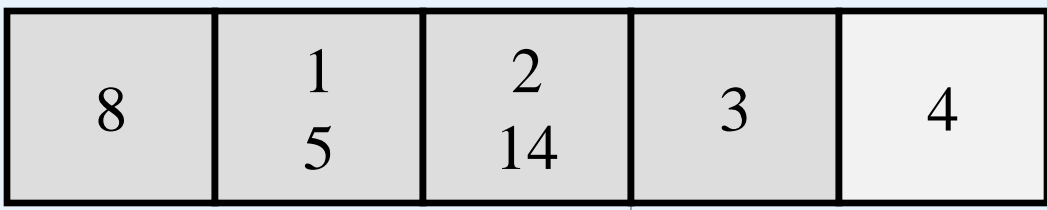
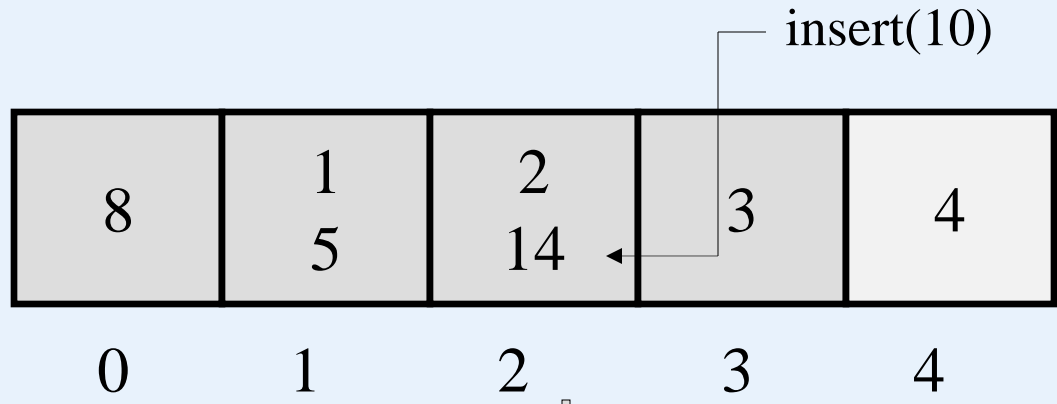
$n=1$ after the split
load factor: $6/10 = 0.6$
no split

File Organizations



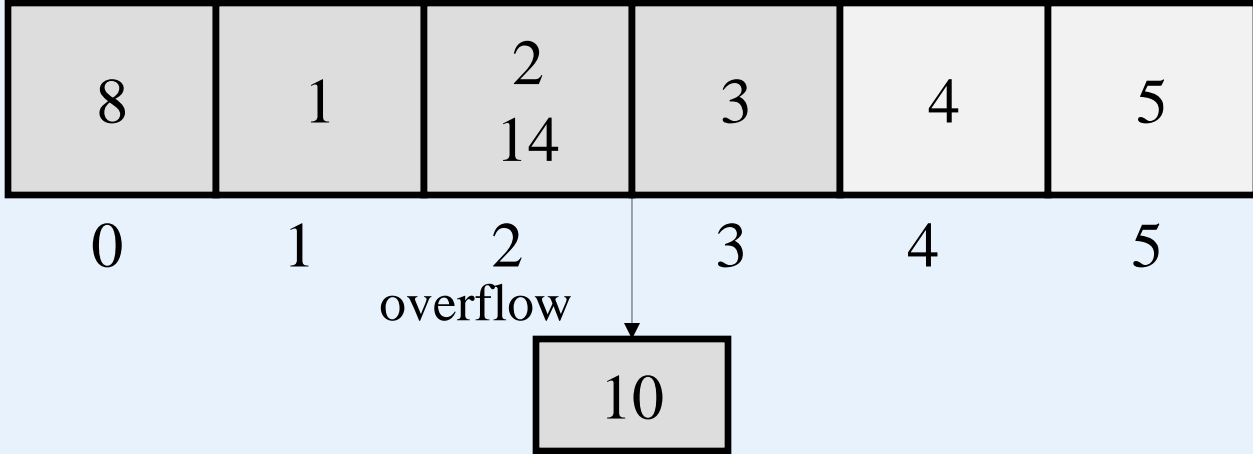
n=1
load factor: $7/10=0.7$
no split

File Organizations



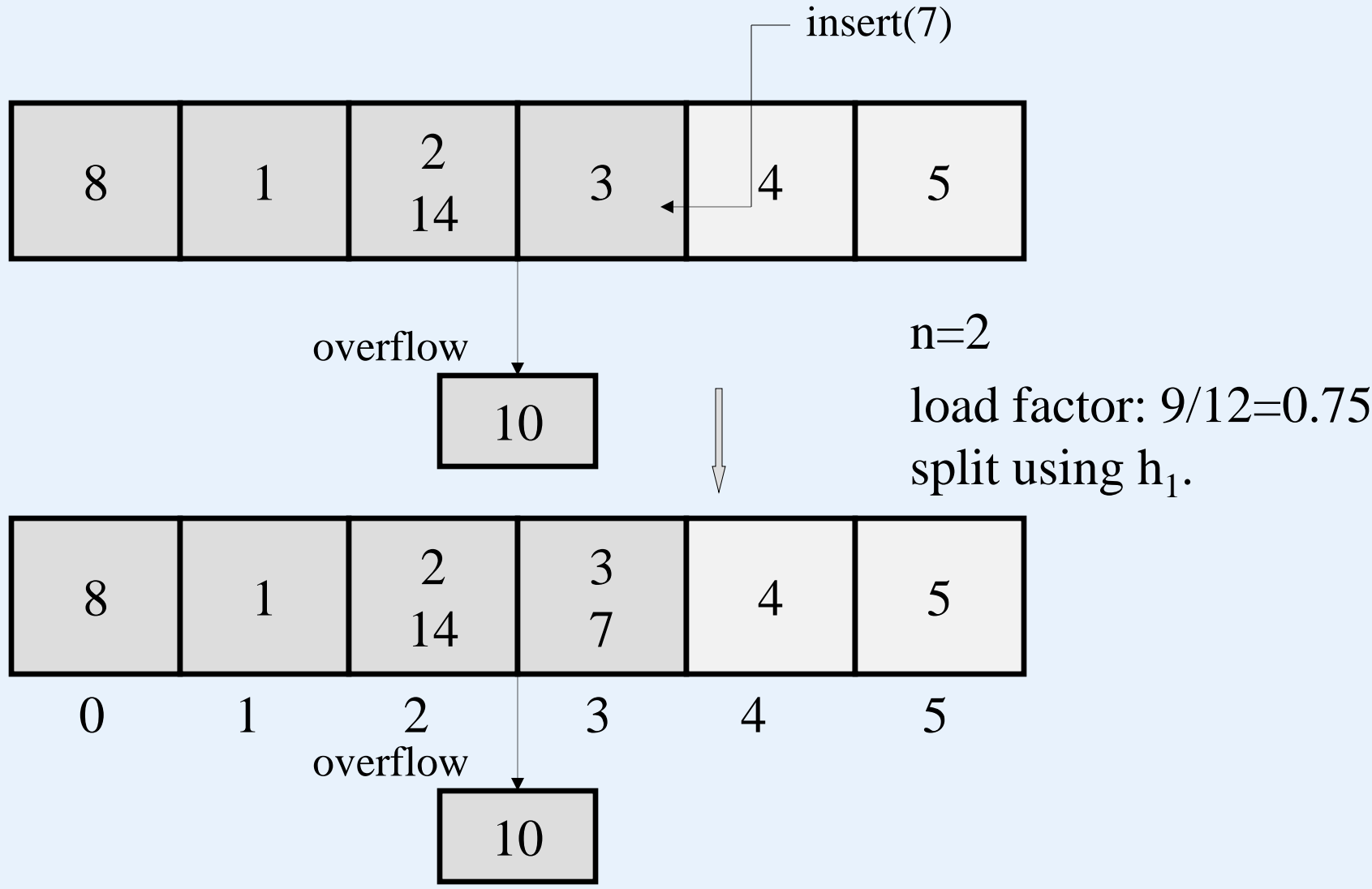
n=1
load factor: $8/10=0.8$
split using h_1 .

File Organizations

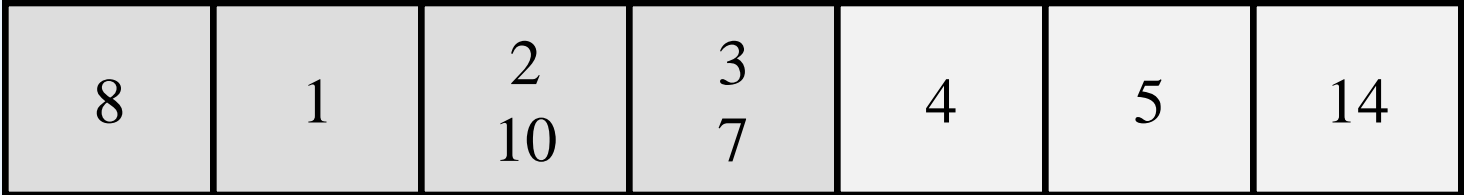


n=2
load factor: $8/12=0.66$
no split

File Organizations



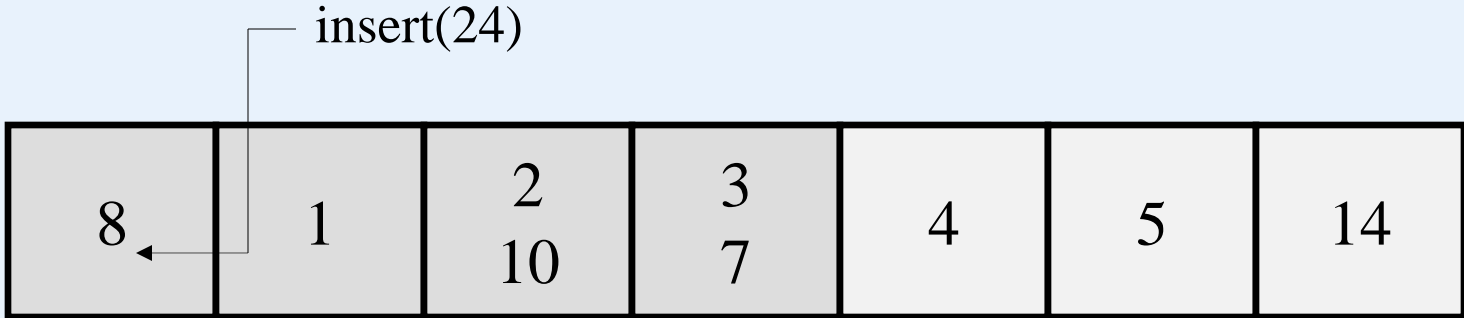
File Organizations



$n=3$

load factor: $9/14=0.642$

no split.



File Organizations

8 24	1	2 10	3 7	4	5	14
---------	---	---------	--------	---	---	----

n=3
load factor: $10/14=0.71$
split using h_1 .

8 24	1	2 10	3	4	5	14	7
---------	---	---------	---	---	---	----	---

File Organizations

8 24	1	2 10	3	4	5	14	7
---------	---	---------	---	---	---	----	---

n=4

The second phase – phase₁

n = 0; using $h_1 = \text{Key} \bmod 2M$ to insert and
 $h_2 = \text{Key} \bmod 4M$ to split.

insert(17)

8 24	1	2 10	3	4	5	14	7
---------	---	---------	---	---	---	----	---

File Organizations

8 24	1 17	2 10	3	4	5	14	7
---------	---------	---------	---	---	---	----	---

n=0
load factor: $11/16=0.687$
no split.

8 24	1 17	2 10	3	4	5	← insert(13)	14	7
---------	---------	---------	---	---	---	--------------	----	---

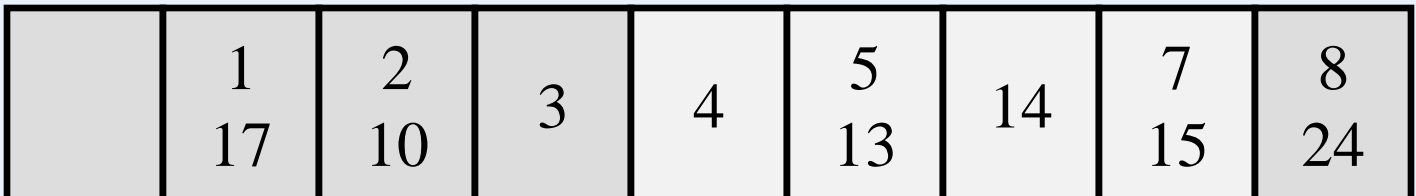
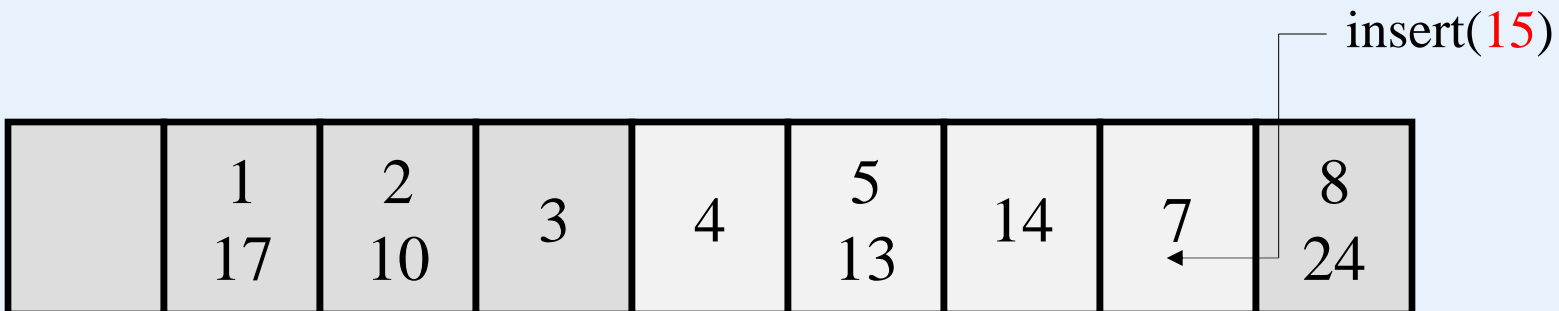
File Organizations

8 24	1 17	2 10	3	4	5 13	14	7
---------	---------	---------	---	---	---------	----	---

n=0
load factor: $12/16=0.75$
split bucket 0, using h_2 :
 $h_2 = \text{Key mod } 4M$

	1 17	2 10	3	4	5 13	14	7	8 24
--	---------	---------	---	---	---------	----	---	---------

File Organizations



n=1

load factor: $13/18=0.722$

split bucket 1, using h_2 .

