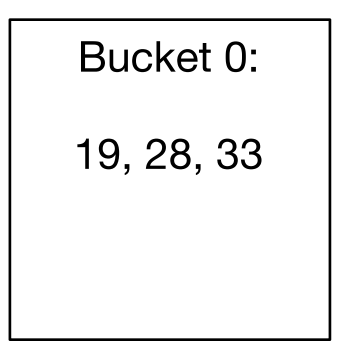
Linear Hashing

Linear Hashing

- Extensible Hashing:
  - Uses a lot of metadata to reflect history of splitting
    - But only splits buckets when they are needed
  - Linear Hashing
    - Splits buckets in a predefined order
    - Minimal meta-data
    - Sounds like a horrible idea, but …

Linear Hashing

- Assume a hash function that creates a large string of bits
  - We start using these bits as we extend the address space
  - Start out with a single bucket, Bucket 0
  - All items are located in Bucket 0



Items with keys 19, 28, 33

Linear Hashing

- Eventually, this bucket will overflow
  - E.g. if the load factor is more than 2
  - Bucket 0 splits
  - All items in Bucket 0 are rehashed:
    - Use the last bit in order to determine whether the item goes into Bucket 0 or Bucket 1
    - Address is $h_1(c) = c \pmod 2$

Linear Hashing

- After the split, the hash table has two buckets:

Bucket 0:	Bucket1:
28	19, 33

- After more insertions, the load factor again exceeds 2

Bucket 0:	Bucket1:
28, 40	11, 19, 33

Linear Hashing

- Again, the bucket splits.
  - But it has to be Bucket 0

Bucket 0:	Bucket1:	Bucket 2:
28, 40	11, 19, 33	

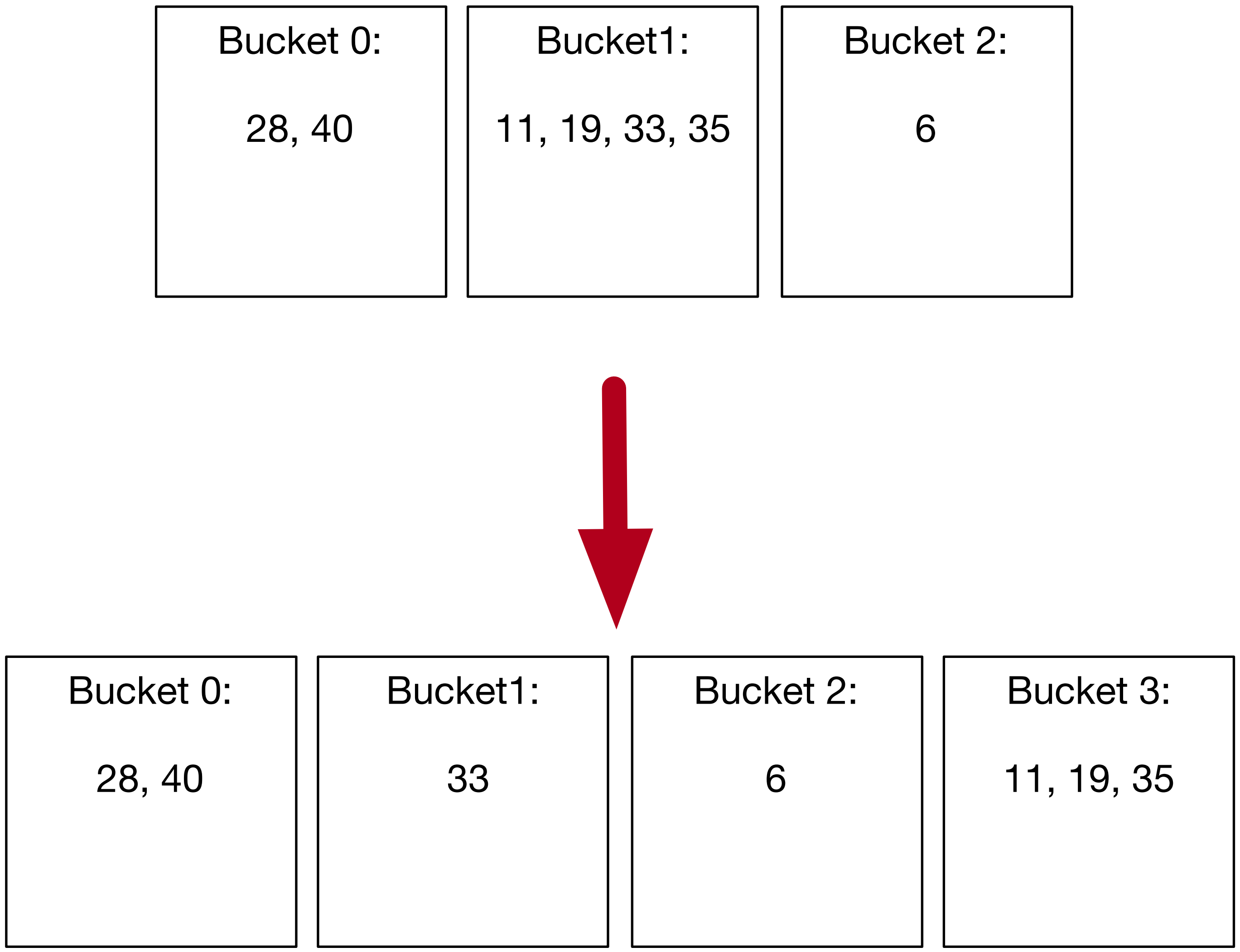
  - For the rehashing, we now use two bits, i.e.

$$h_2(c) = c \pmod 4$$

    - But only for those items in Bucket 0

Linear Hashing

- After some more insertions, Bucket 1 will split



Linear Hashing

- The state of a linear hash table is described by the number N of buckets
  - The level l is the number of bits that are being used to calculate the hash
  - The split pointer s points to the next bucket to be split
  - The relationship is

$$N = 2^l + s$$

    - This is unique, since always $s < 2^l$

Linear Hashing

- Addressing function
  - The address of an item with key c is calculated by

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

  - This reflects the fact that we use more bits for buckets that are already split

Linear Hashtable Evolution

$N = 1 = 2^0 + 0$

```
Number of buckets: 1
Split pointer: 0
Level: 0
```

```
def address(c):
    a = hash(c) % 2**l
    if a < s:
        a = hash(c) % 2**(l+1)
    return a
```

Bucket 0:

19, 28, 33

Linear Hashtable Evolution

$N = 2 = 2^1 + 0$

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

```
Number of buckets: 2
Split pointer: 0
Level: 1
```

Bucket 0:	Bucket1:
28	19, 33

```
Add items with hashes 40 and 11
This gives an overflow and we split Bucket 0
```

Linear Hashtable Evolution

$N = 3 = 2^1 + 1$

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
     a = hash(c) % 2**(l+1)
   return a
```

```
Number of buckets: 3
Split pointer: 1
Level: 1
```

Bucket 0:	Bucket1:
28, 40	11, 19, 33

```
split Bucket 0
Create Bucket 2
Use new hash function on items in Bucket 0
```

Bucket 0:	Bucket1:	Bucket 2:
28, 40	11, 19, 33	

```
No items were moved
```

Linear Hashtable Evolution

$N = 3 = 2^1 + 1$

```
Number of buckets: 3
Split pointer: 1
Level: 1
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:
28, 40	11, 19, 33	

Add items 6, 35

Bucket 0:	Bucket1:	Bucket 2:
28, 40	11, 19, 33, 35	6

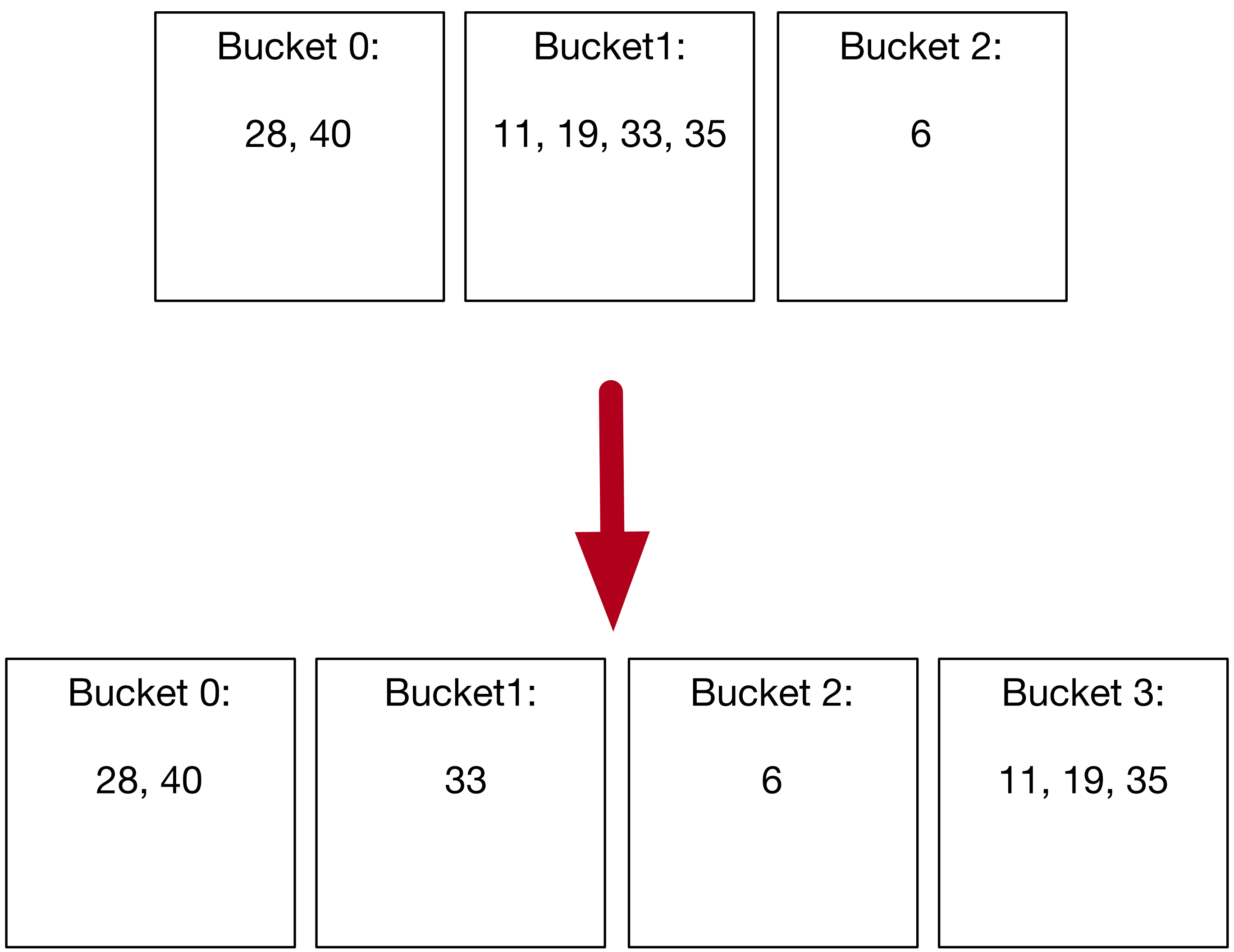
Because of overflow, we split Bucket 1

Linear Hashtable Evolution

$$N = 4 = 2^2 + 0$$

```
Number of buckets: 4
Split pointer: 0
Level: 2
```

```
def address(c):
    a = hash(c) % 2**l
    if a < s:
        a = hash(c) % 2**(l+1)
    return a
```



Linear Hashtable Evolution

$N = 4 = 2^2 + 0$

```
Number of buckets: 4
Split pointer: 0
Level: 2
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:
28, 40	33	6	11, 19, 35

```
Now add keys 8, 49
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:
28, 40, 8	33, 49	6	11, 19, 35

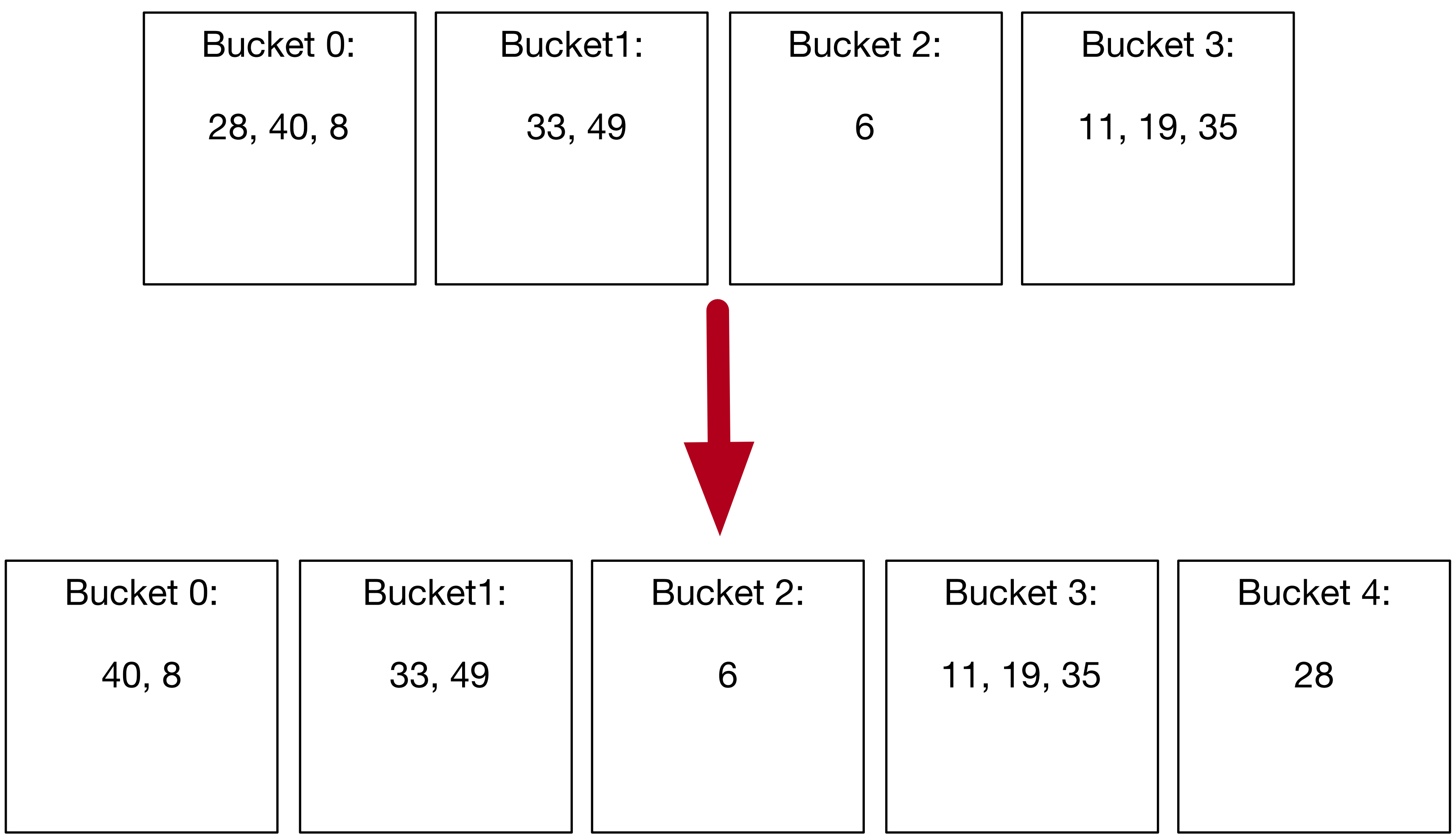
```
Creates an overflow!
Need to split!
```

Linear Hashtable Evolution

$$N = 5 = 2^2 + 1$$

```
Number of buckets: 1
Split pointer: 1
Level: 2
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```




```
Create Bucket 4.
Rehash Bucket 0.
```

Linear Hashtable Evolution

$N = 5 = 2^2 + 1$

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

```
Number of buckets: 5
Split pointer: 1
Level: 2
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:
40, 8	33, 49	6	11, 19, 35	28

Add keys 9, 42

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:
40, 8	9, 33, 49	6, 42	11, 19, 35	28

Creates an overflow!
Need to split!

Linear Hashtable Evolution

$$N = 6 = 2^2 + 2$$

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

```
Number of buckets: 1
Split pointer: 2
Level: 2
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:
40, 8	9, 33, 49	6, 42	11, 19, 35	28

Split

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:
40, 8	9, 33, 49	6, 42	11, 19, 35	28	

No item actually moved, but average load factor is now again under 2.

Linear Hashtable Evolution

$$N = 6 = 2^2 + 2$$

```
Number of buckets: 6
Split pointer: 2
Level: 2
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
     a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:
40, 8	9, 33, 49	6, 42	11, 19, 35	28	

`add 5,10`

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:
40, 8	9, 33, 49	6, 10, 42	11, 19, 35	28	5

Linear Hashtable Evolution

$$N = 7 = 2^2 + 3$$

```
Number of buckets: 7
Split pointer: 3
Level: 2
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:
40, 8	9, 33, 49	6, 10, 42	11, 19, 35	28	5

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:
40, 8	9, 33, 49	10, 42	11, 19, 35	28	5	6

Linear Hashtable Evolution

$N = 7 = 2^2 + 3$

```
Number of buckets: 7
Split pointer: 3
Level: 2
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
     a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:
40, 8	9, 33, 49	10, 42	11, 19, 35	28	5	6

`add 92, 74`

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:
40, 8	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5	6

Linear Hashtable Evolution

$N = 8 = 2^3 + 0$

```
Number of buckets: 8
Split pointer: 0
Level: 3
```

```
def address(c):
    a = hash(c) % 2**l
    if a < s:
        a = hash(c) % 2**(l+1)
    return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:
40, 8	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5	6

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:
40, 8	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5	6	

Linear Hashtable Evolution

$$N = 8 = 2^3 + 0$$

```
Number of buckets: 8
Split pointer: 0
Level: 3
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:
40, 8	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5	6	

`add 13, 54`

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:
	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5, 13	6, 54	

Linear Hashtable Evolution

$$N = 9 = 2^3 + 1$$

```
Number of buckets: 9
Split pointer: 1
Level: 3
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:
	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5, 13	6, 54	

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:	Bucket 8:
	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5, 13	6, 54		40, 8

Linear Hashtable Evolution

$$N = 9 = 2^3 + 1$$

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

```
Number of buckets: 9
Split pointer: 1
Level: 3
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:	Bucket 8:
	9, 33, 49	10, 42, 74	11, 19, 35	28, 92	5, 13	6, 54		40, 8

`add 1, 81`

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:	Bucket 8:
	1, 9, 33, 49, 81	10, 42, 74	11, 19, 35	28, 92	5, 13	6, 54		40, 8

Linear Hashtable Evolution

$$N = 10 = 2^3 + 2$$

```
Number of buckets: 10
Split pointer: 2
Level: 3
```

```
def address(c):
   a = hash(c) % 2**l
   if a < s:
      a = hash(c) % 2**(l+1)
   return a
```

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:	Bucket 8:	Bucket 9:
	1, 33, 49, 81	10, 42, 74	11, 19, 35, 67, 99	28, 92	5, 13	6, 54	39	40, 8	9

Bucket 0:	Bucket1:	Bucket 2:	Bucket 3:	Bucket 4:	Bucket 5:	Bucket 6:	Bucket 7:	Bucket 8:	Bucket 9:	Bucket 10:
	1, 33, 49, 81		11, 19, 35, 67, 99	28, 92	5, 13	6, 54	39	40, 8	9	10, 42, 74

Linear Hashing

- Observations:
  - Buckets split in fixed order
    - 0, 0,1, 0, 1, 2, 3, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, …, 15, 0, …
    - Address calculation is modulo 2^l, i.e. the *l* least significant bits
    - Buckets 0, 1, …, s-1 and 2***l*, 2***l*+1, … *N*-1 are already split, they have on average half the size of the buckets *s*, *s*+1, …, 2***l*.

Linear Hashing

- Observations:
  - An overflowing bucket is not necessarily split immediately
  - Sometimes, a split leaves all keys in the splitting bucket or moves them all to the new bucket
- On average, a bucket will have α items in them

Linear Hashing Rules

- Buckets are numbered $0, 1, \ldots, N-1$
- Number of buckets N, split pointer s, and level l are related by
  - $N = 2^l + s$ (with $s < 2^l$)
  - Warning: There is a variant where we start with any number of buckets instead of one. This relationship then no longer holds.
- Address of record with key hashing to c is
  - $c \pmod{2^l}$ or $c \pmod{2^{l+1}}$

File State

- The file state of an LH file is determined by the number n of buckets
  - level l
  - split pointer s
  - Formula: $n = 2^l + s$ with l as high as possible, i.e. with $s \in \{0,1,\ldots,2^l - 1\}$

File State

- Clarification regarding the literature
  - The original LH scheme can start with any number of buckets
  - In this class, we are using the most common case

Exercise

- What is the level and the state of an LH file with 13 buckets?

Solution

- We write $13 = 2^3 + 5$
  - Level is $l = \lfloor \log_2(13) \rfloor = 3$
  - Split pointer is $s = 13 - l$

Exercise

- Where would the records with the following (randomly picked) keys be inserted?
- 82
- 27
- 37

Solution

- Level is 3, so we use first remainder modulo $2^3 = 8$ and $2^4 = 16$ second
- $82 \pmod 8 = 2$. Since $2 < 5$, we rehash:
  $82 \pmod{16} = 2$ and we insert into bucket 2
- $27 \pmod 8 = 3$. Since 3 < 5, we rehash:
  $27 \pmod{16} = 11$. We insert into bucket 11
- $37 \pmod 8 = 5$. Since $5 \not< 5$, we do not rehash but insert into bucket 5.

Exercise

- Where would the records with the following (randomly picked) keys be inserted?
- 48
- 60
- 63
- 71

Solution

- $48 \pmod{8} = 0$. Rehash: $48 \pmod{16} = 0$ and insert into bucket 0.
- $60 \pmod{8} = 4$. Rehash: $60 \pmod{16} = 12$ and insert into bucket 12.
- $63 \pmod{8} = 7$. Rehash not necessary. Insert into bucket 7.
- $71 \pmod{8} = 7$. No rehash is necessary.

Exercise

- Where would the records with the following (randomly picked) keys be inserted?
- 98
- 75
- 25
- 30

Solution

- $98 \pmod 8 = 2$. Rehash: $98 \pmod{16} = 2$. Insert into bucket 2
- $75 \pmod 8 = 3$. Rehash: $75 \pmod{16} = 11$. Insert into bucket 11
- $25 \pmod 8 = 1$. Rehash: $25 \pmod{16} = 9$. Insert into bucket 9.
- $30 \pmod 8 = 6$. Insert into bucket 6.

Exercise

- Give the level and split pointer values as an LH file moves from 6 buckets to 20

Solution

Nr o Buckets	Level	Split Ptr
6	2	2
7	2	3
8	3	0
9	3	1
10	3	2
11	3	3
12	3	4
13	3	5
14	3	6
15	3	7
16	4	0
17	4	1
18	4	2
19	4	3
20	4	4

Interpretation

- We can encapsulate the behavior of the level and split pointer into the following algorithm

```
def split(level, split_pointer):
     split_pointer += 1
     if split_pointer == 2**level:
          split_pointer = 0
          level += 1
     return (level, split_pointer)
```

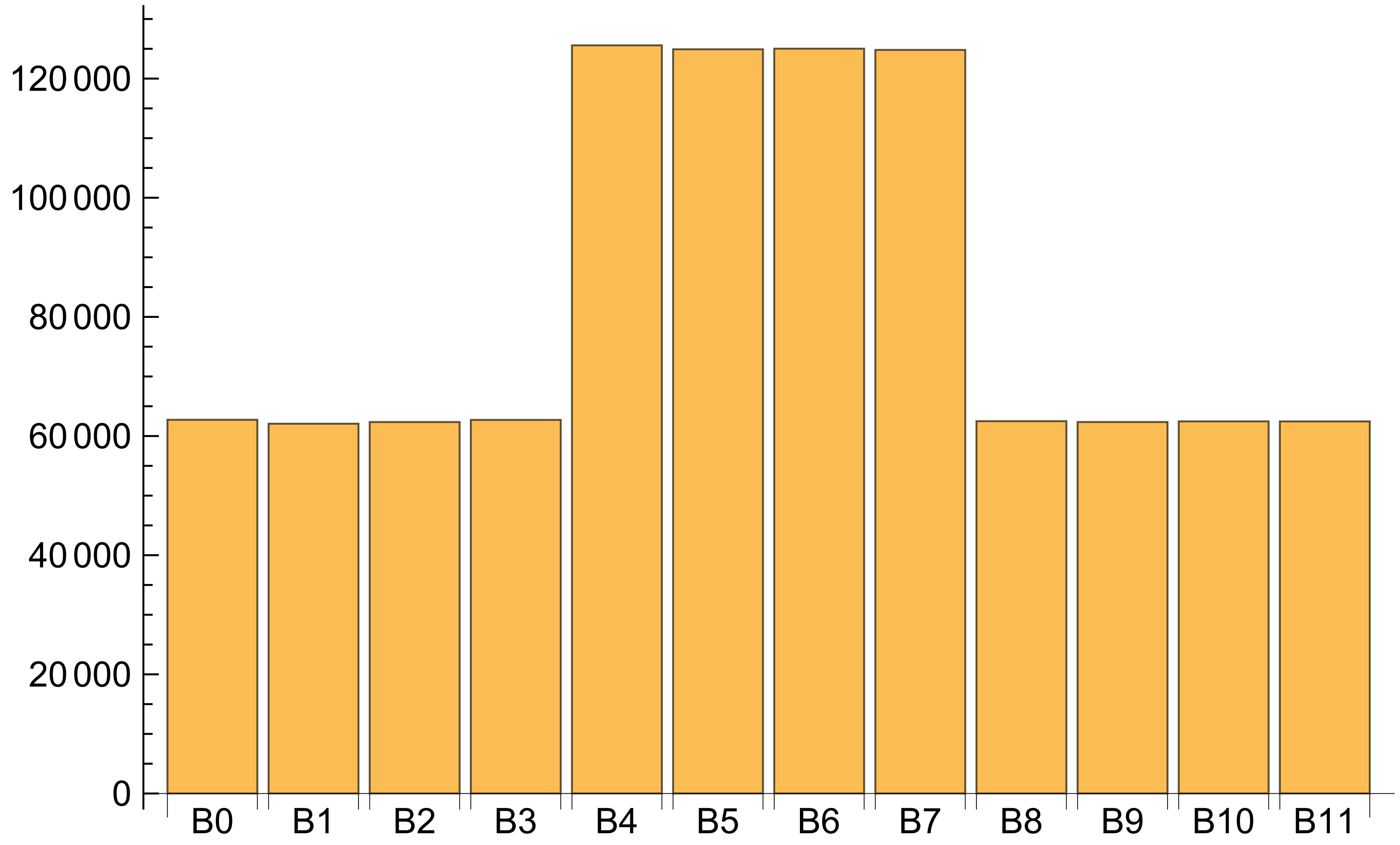
- We increment the split pointer
- If the split pointer equals 2^{level} then set the split pointer to zero and increment the level

Programming Exercise

- Using a programming platform of your choice, implement the LH addressing algorithm
- Insert 1000 records with key uniformly selected between 0 and $2^{32} - 1$ into an LH file with (a) 12 and (b) 25 buckets.
- Look at the size of the buckets.

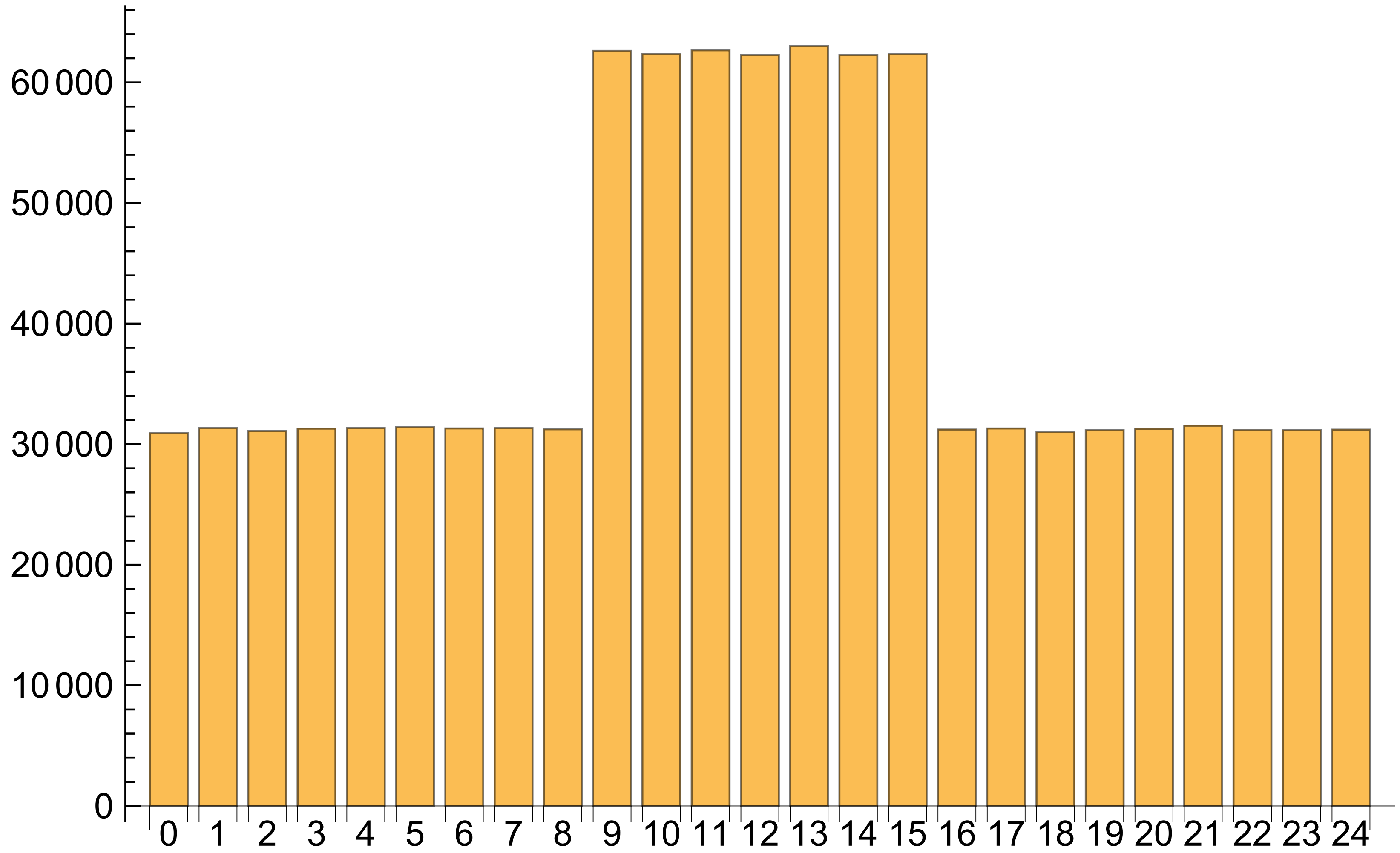
Solution

- I changed the number to 1,000,000
  - For 12 buckets:



Solution

- Here is the chart for 25 buckets



Interpretation

- Even with a perfect hash function, an LH file has buckets of equal size only if the number of buckets is a power of two.
- Otherwise, there are buckets already split in the current round and those not yet split.
  - The latter have about twice as many records