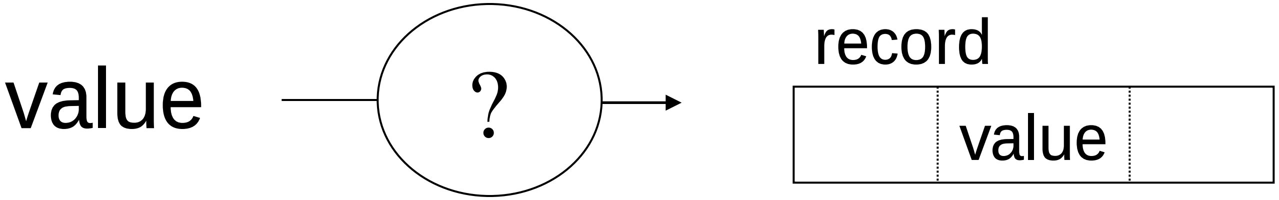
CSCD43: Database Systems Technology

Lecture 7

Wael Aboulsaadat

Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.


value
?
record
value

Topics

- Conventional Indexes

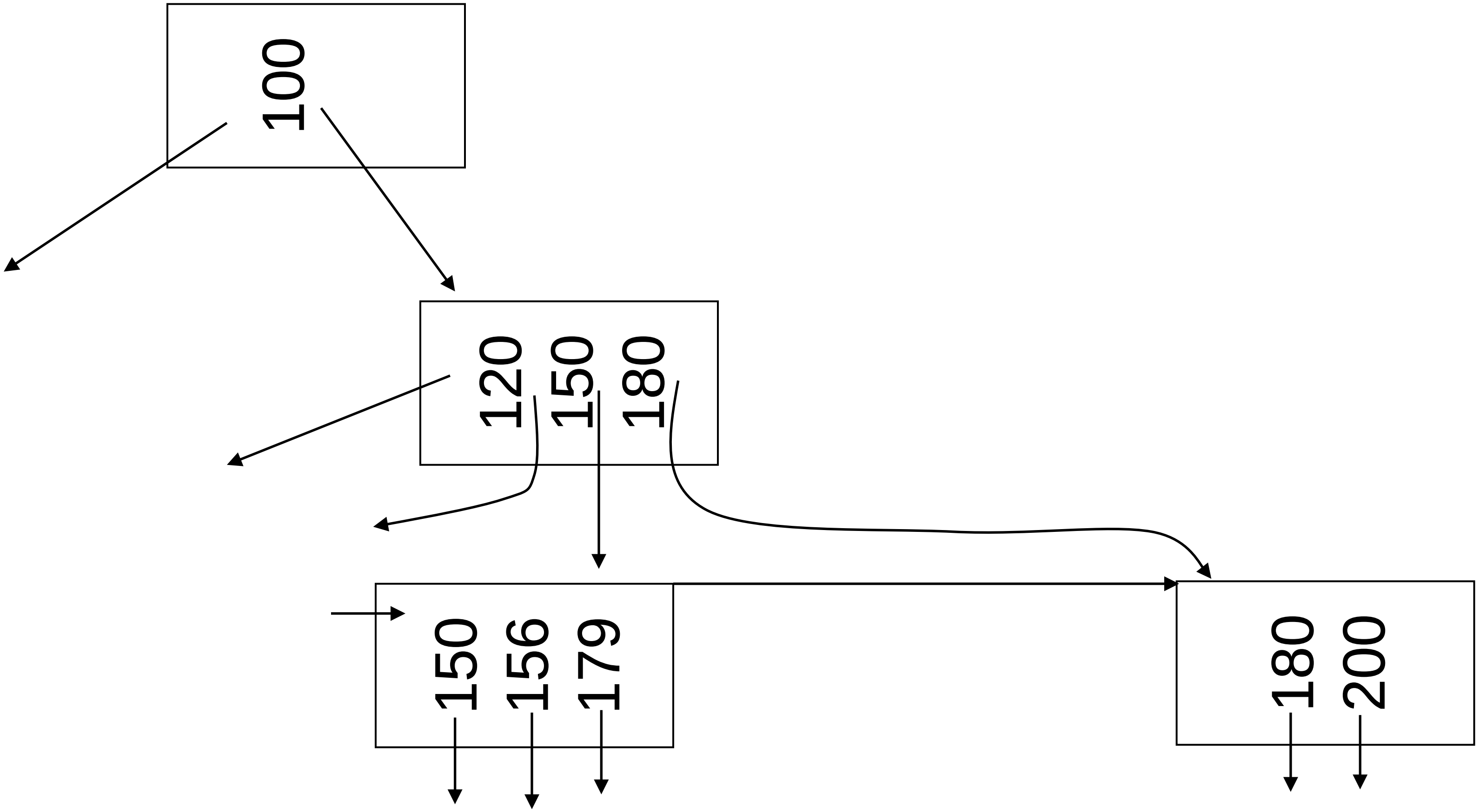
→ B-trees

- Hashing Schemes
- Bitmap Indexes

(c) Insert key = 160

n=3

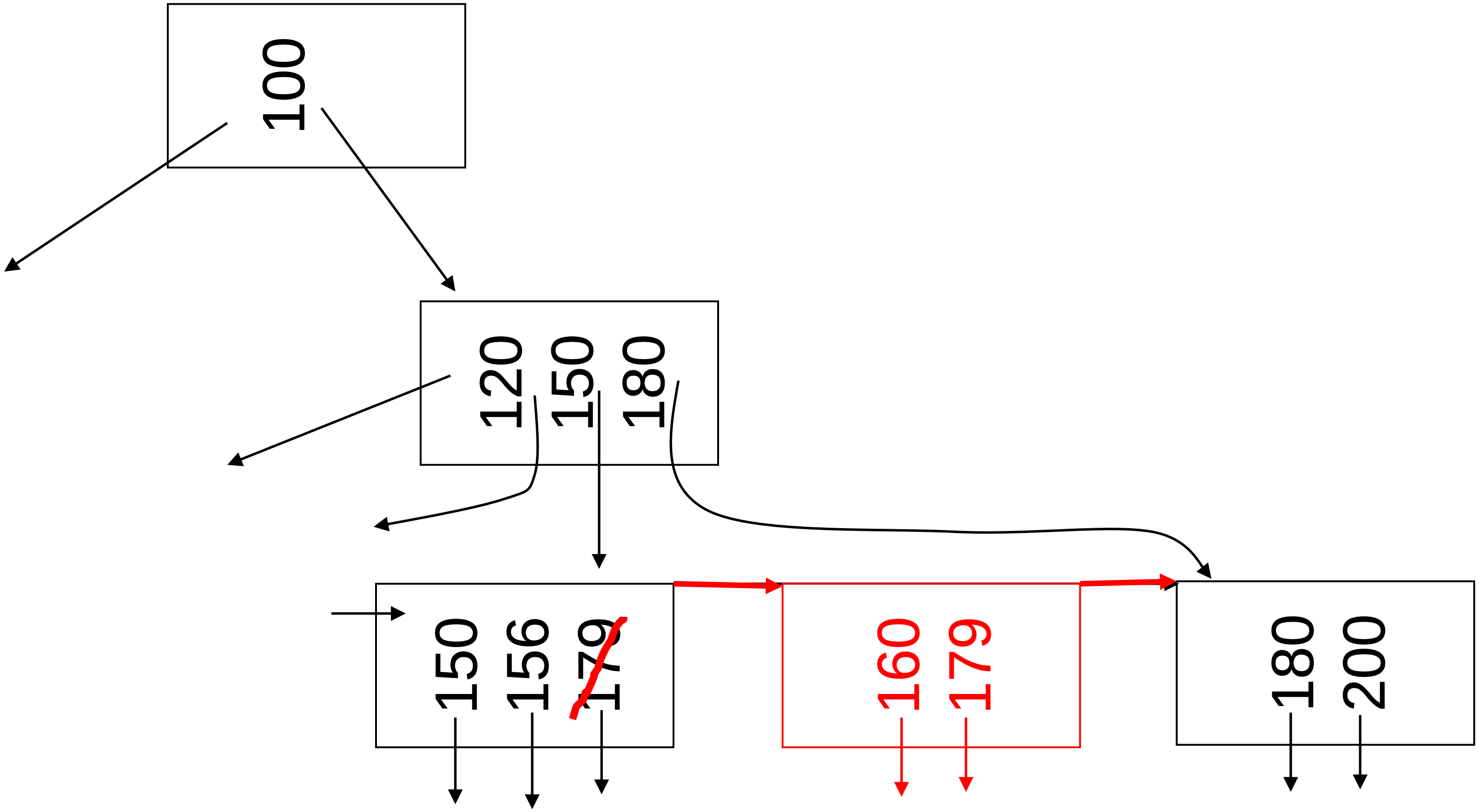
(n+1)/2



(c) Insert key = 160

n=3

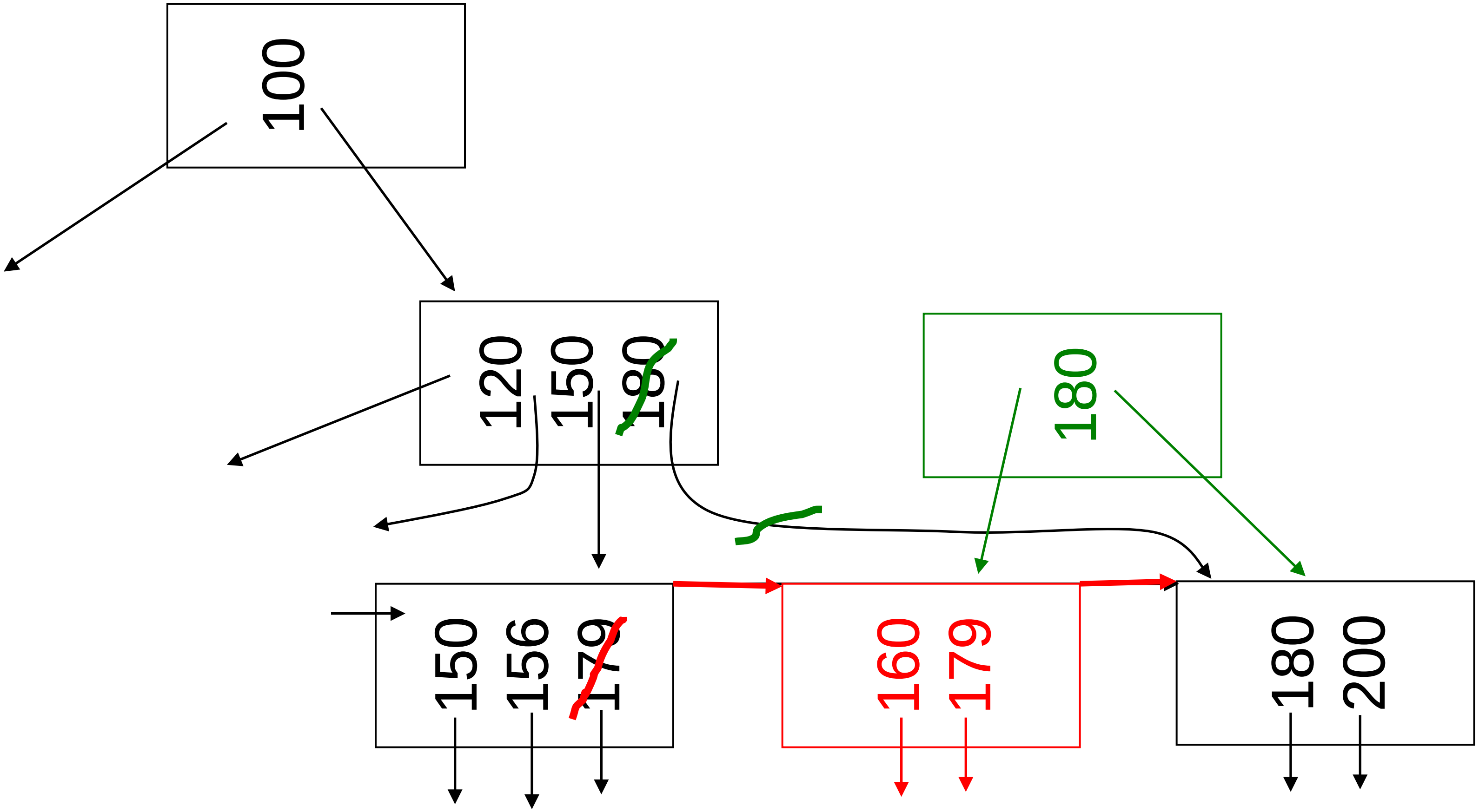
(n+1)/2



(c) Insert key = 160

n=3

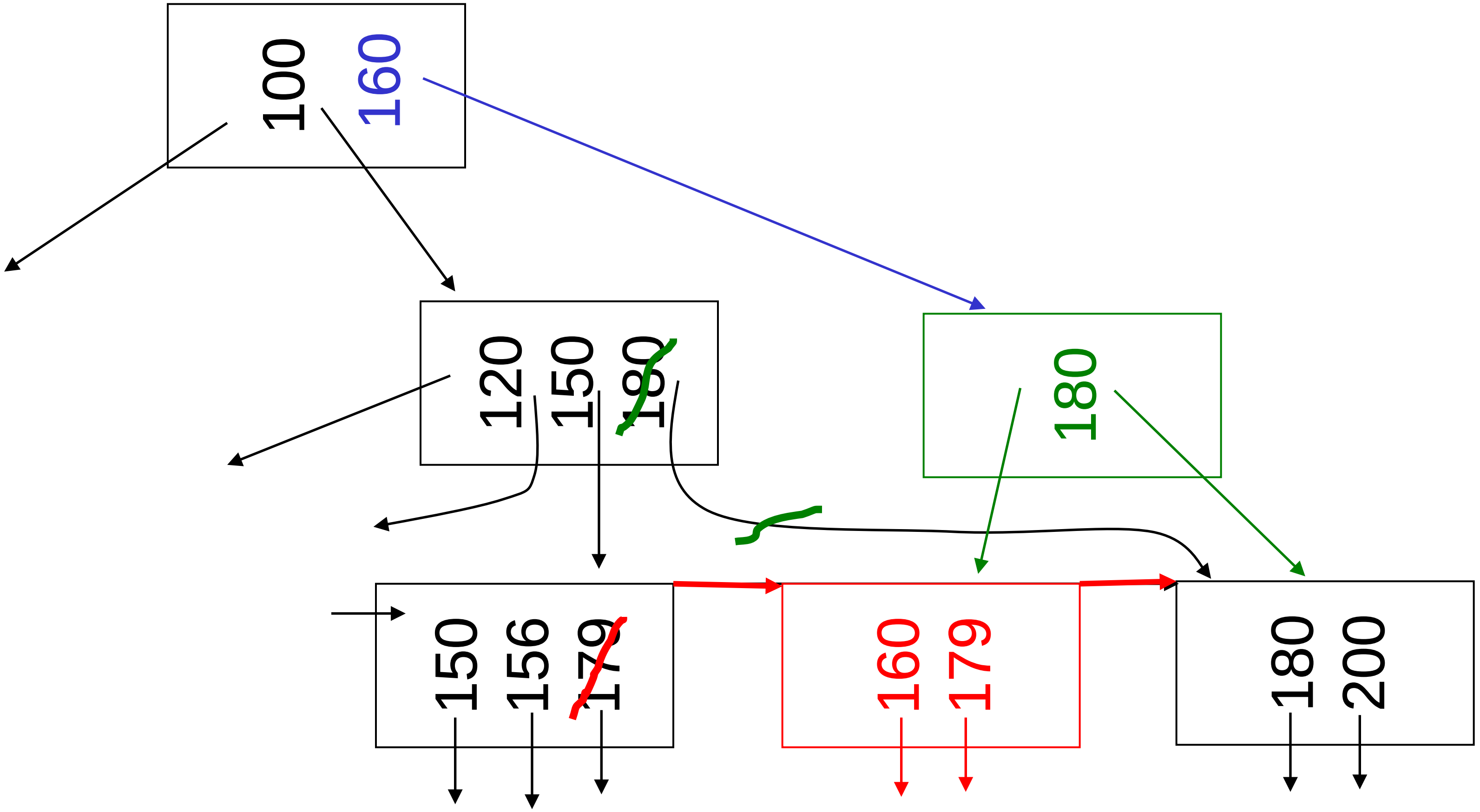
(n+1)/2



(c) Insert key = 160

n=3

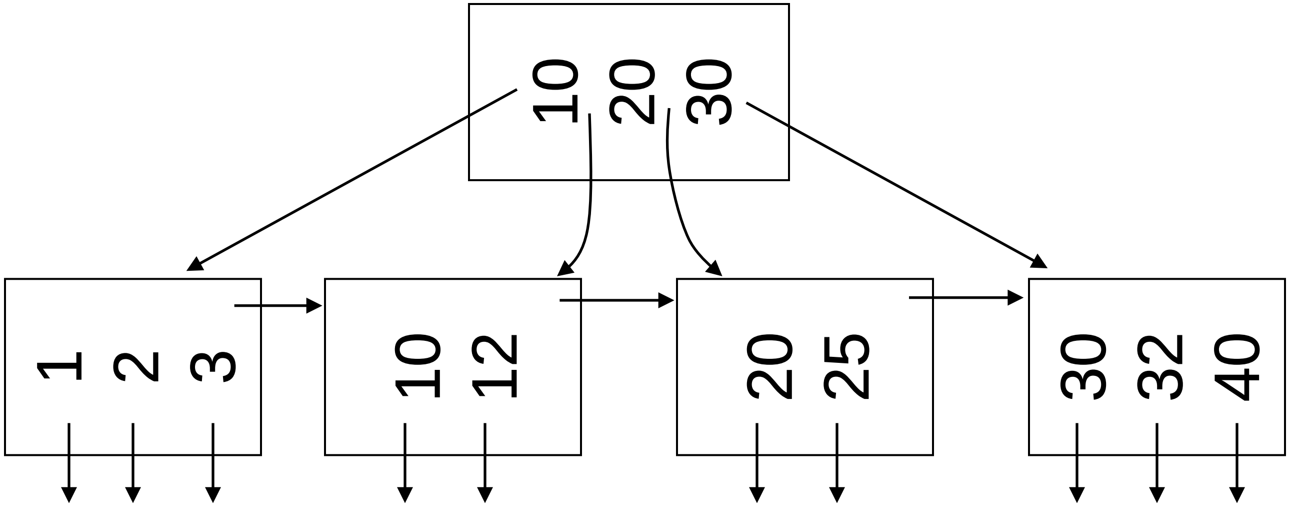
(n+1)/2



(d) New root, insert 45

n=3

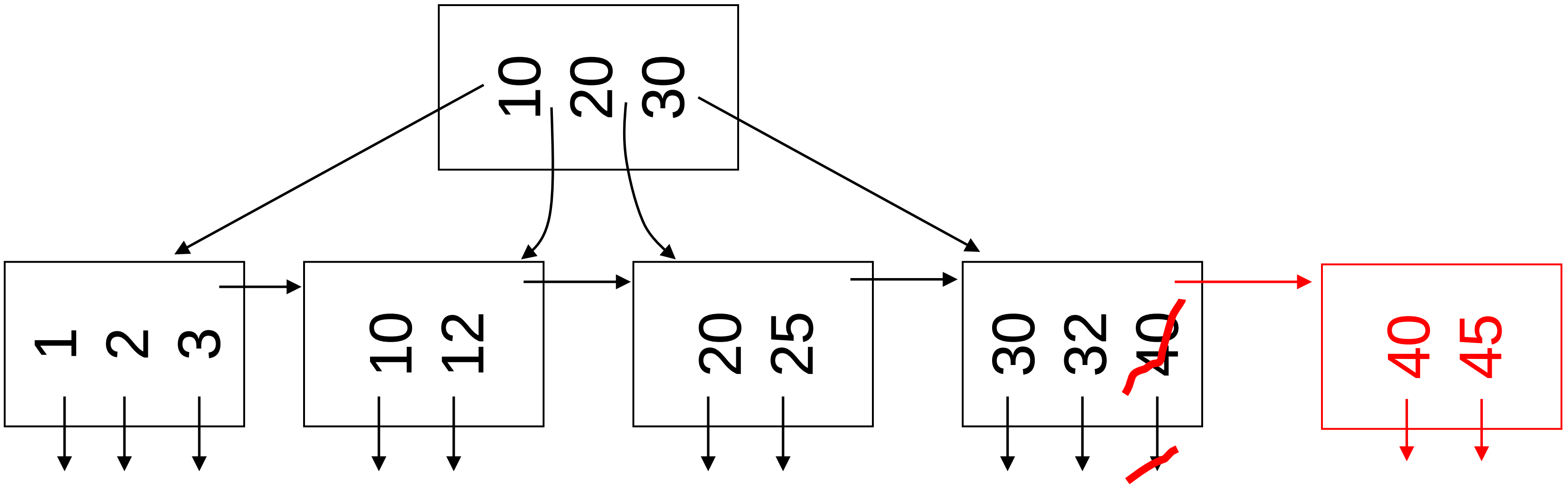
(n+1)/2



(d) New root, insert 45

n=3

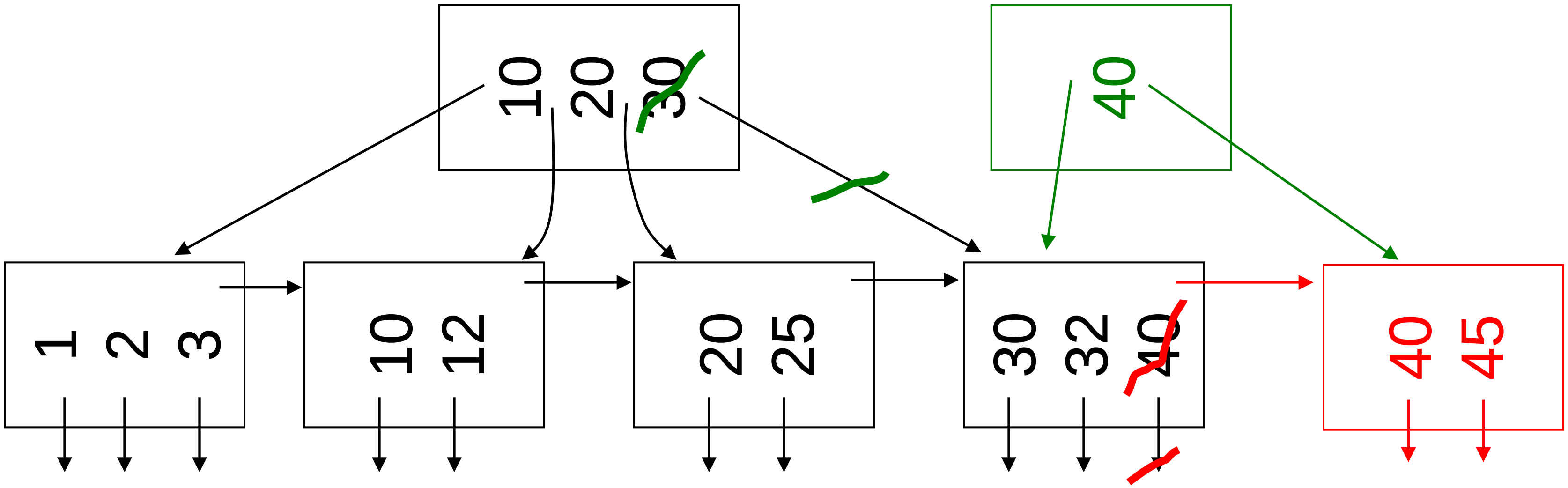
(n+1)/2



(d) New root, insert 45

n=3

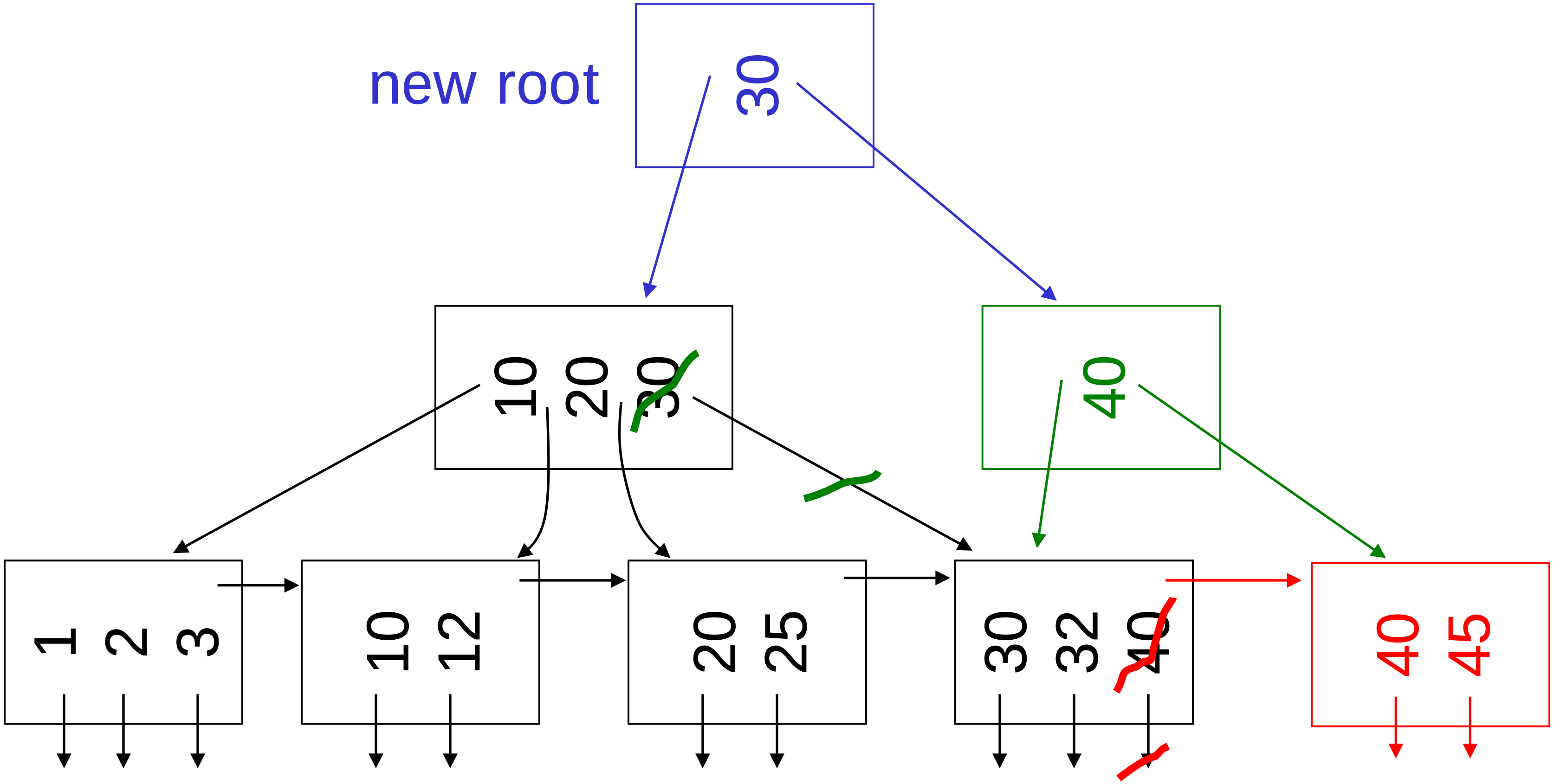
(n+1)/2



(d) New root, insert 45

n=3

(n+1)/2



Deletion from B+tree

(a) Simple case - no example

(b) Coalesce with neighbor (sibling)

(c) Re-distribute keys

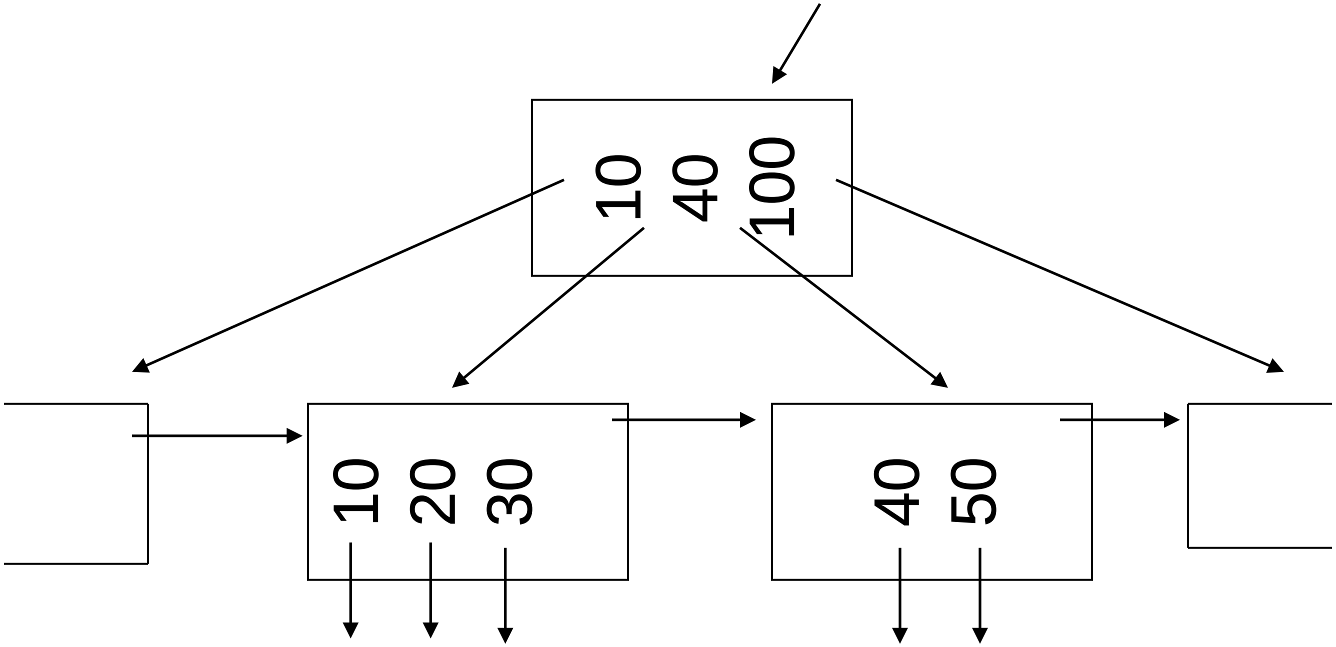
(d) Cases (b) or (c) at non-leaf

(b) Coalesce with sibling

- Delete 50

n=4

(n+1)/2

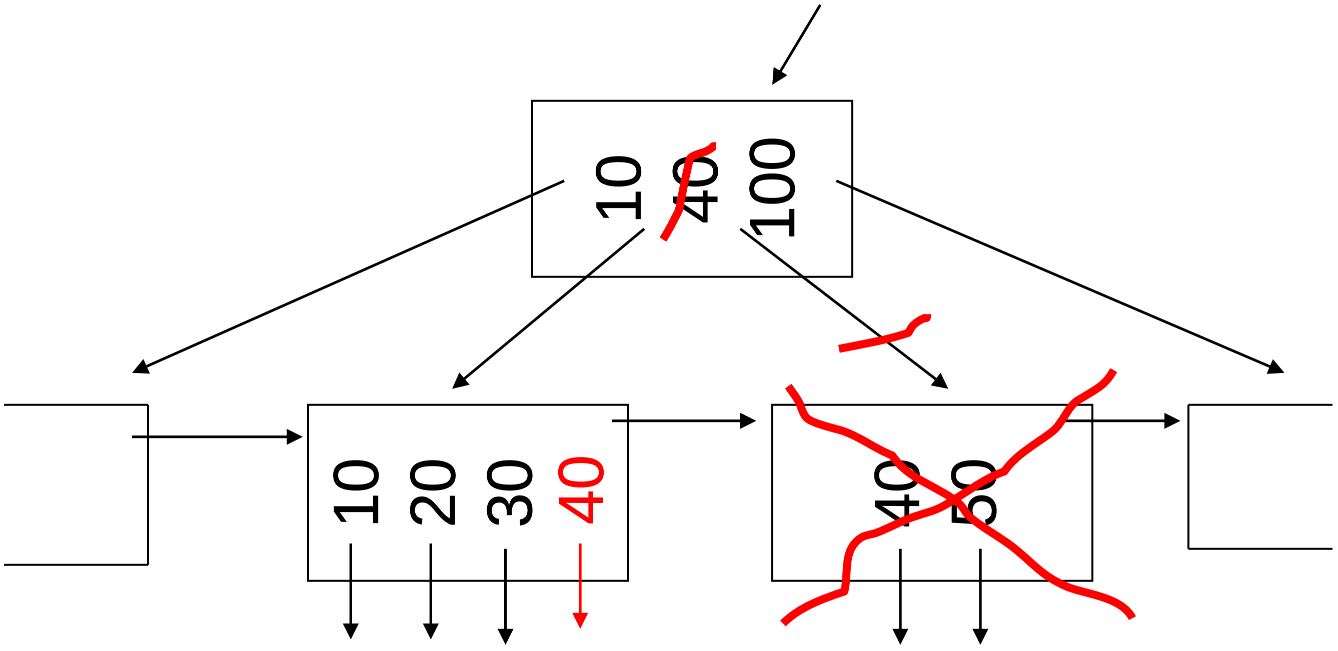


(b) Coalesce with sibling

- Delete 50

n=4

(n+1)/2

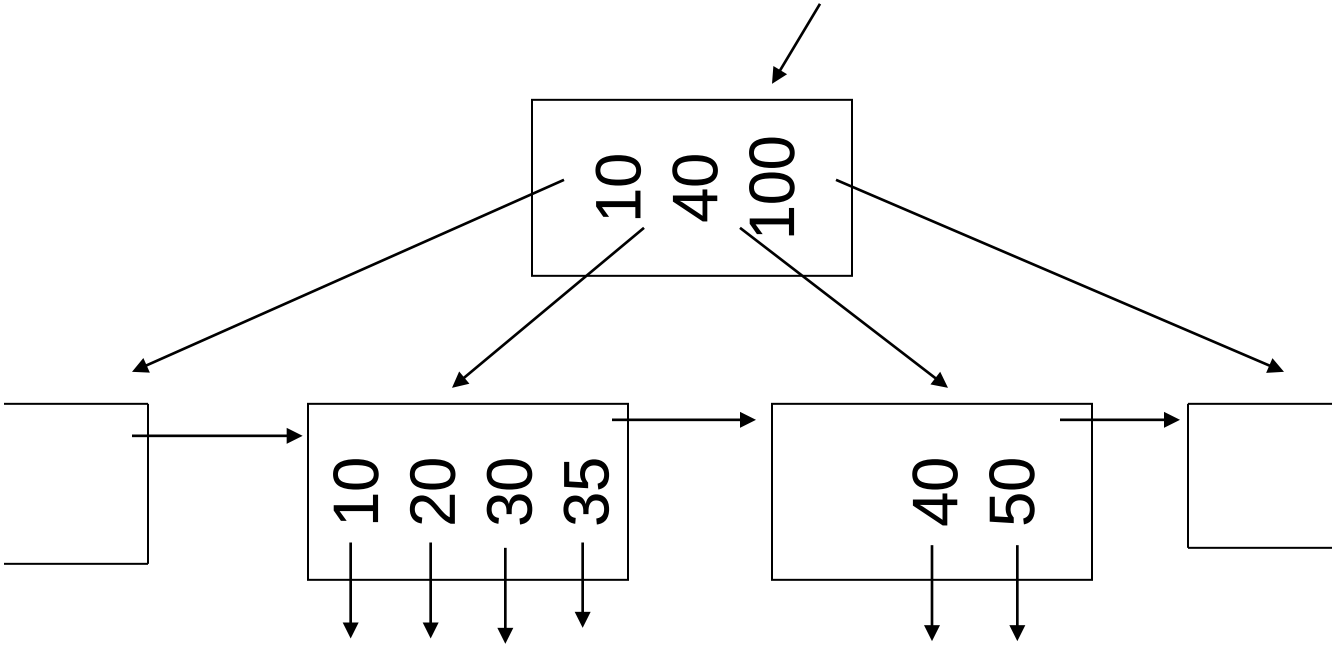


n=4

(n+1)/2

(c) Redistribute keys

– Delete 50

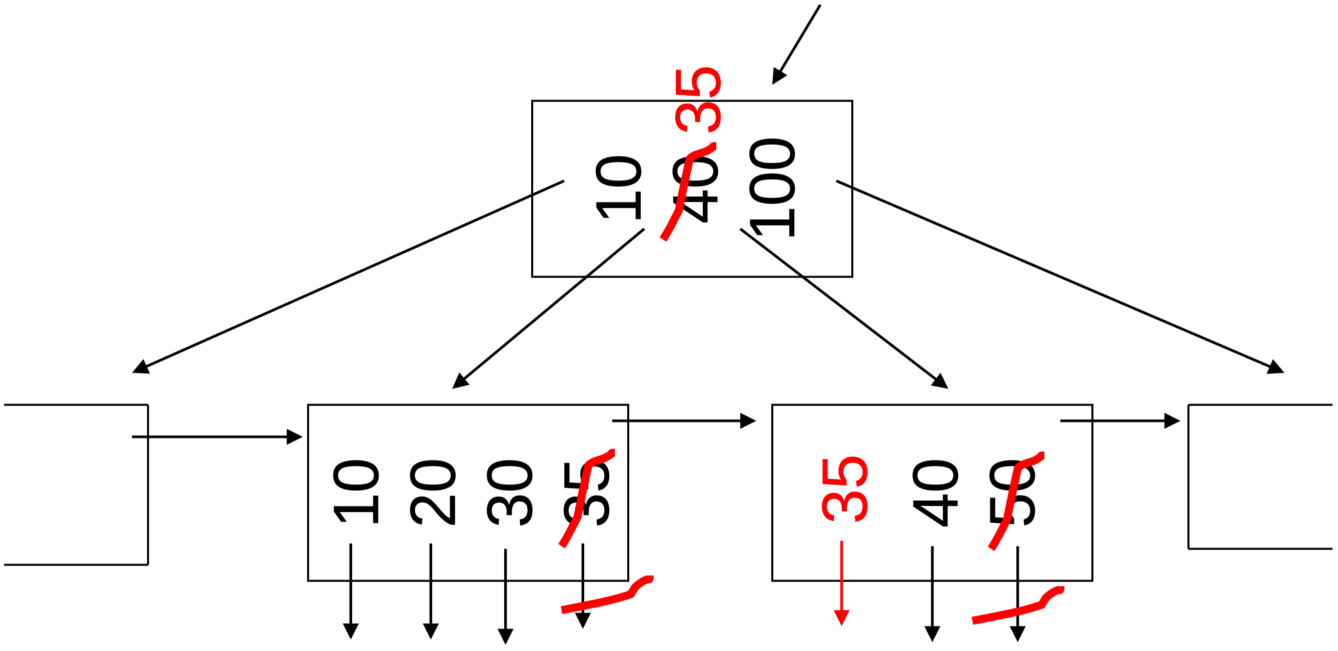


(c) Redistribute keys

– Delete 50

n=4

(n+1)/2

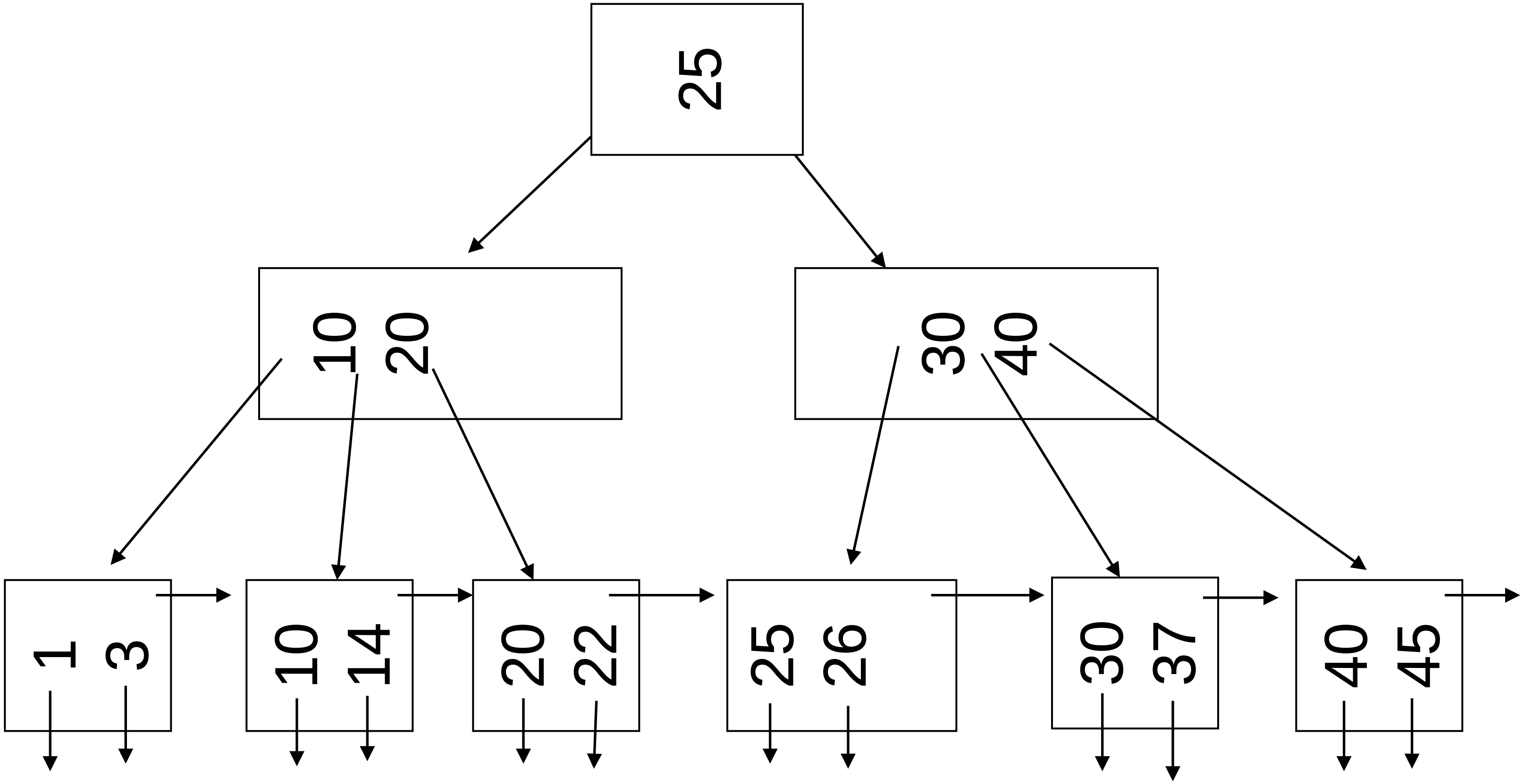


n=4

(n+1)/2

(d) Non-leaf coalese

– Delete 37

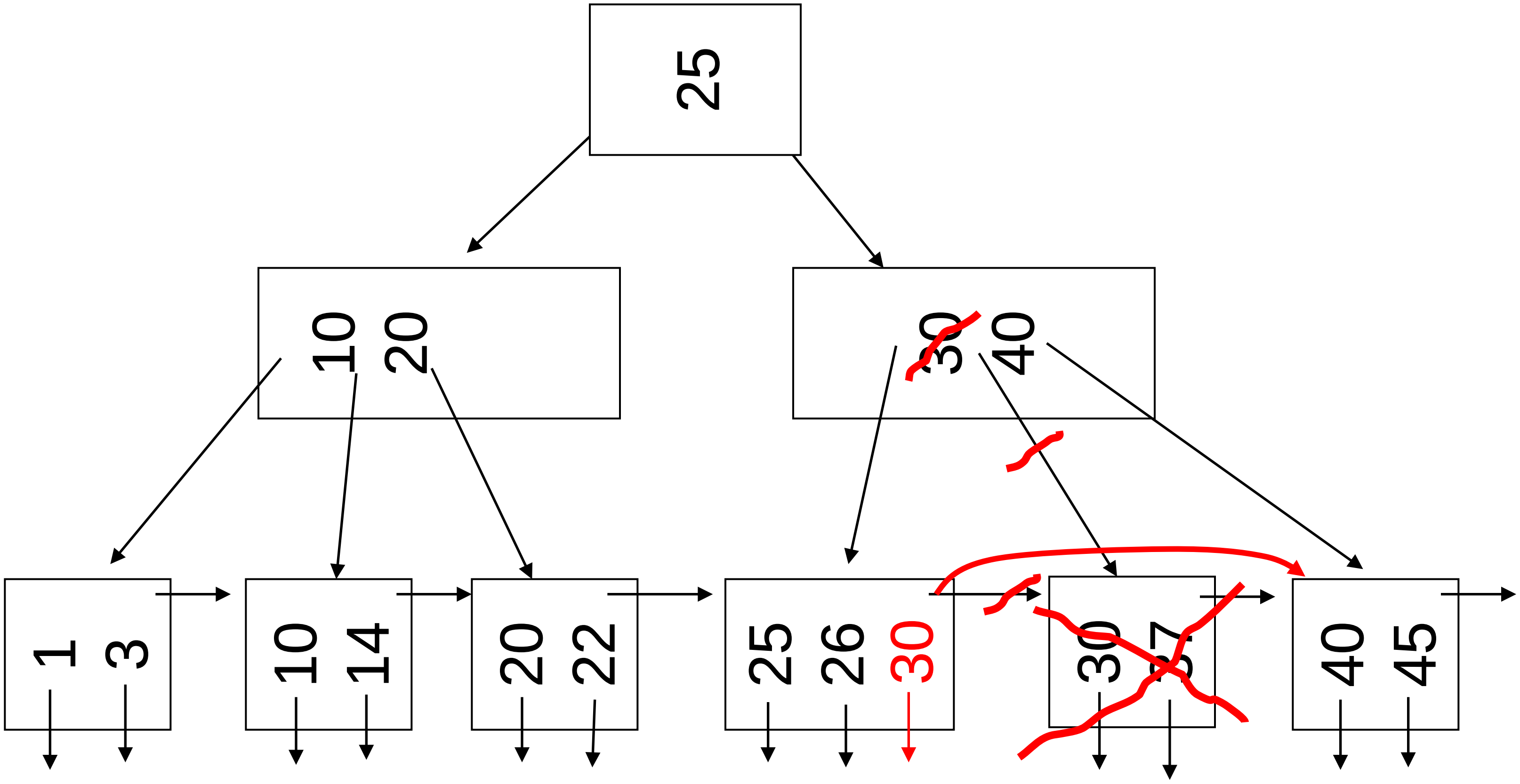


n=4

(n+1)/2

(d) Non-leaf coalese

– Delete 37

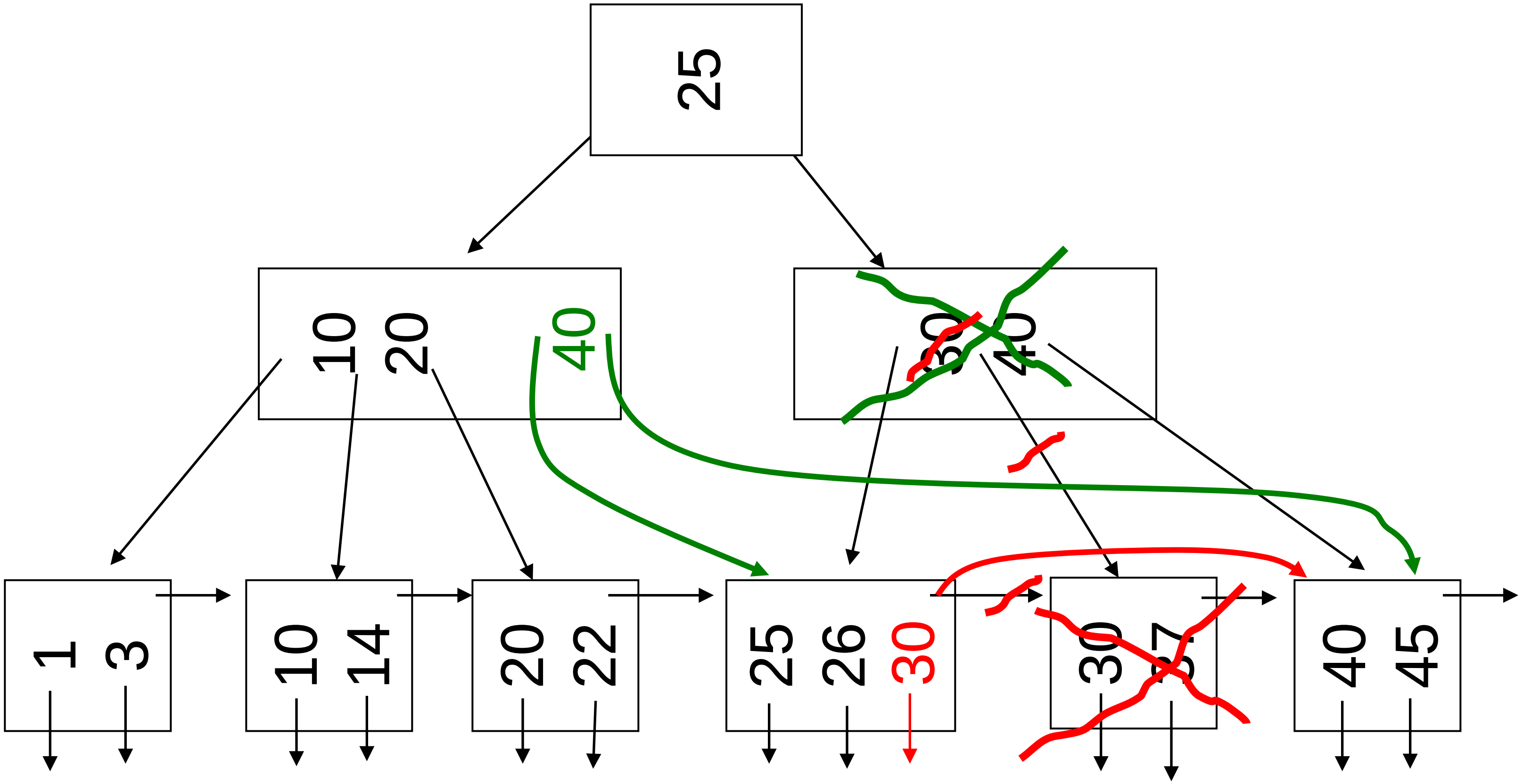


(d) Non-leaf coalese

n=4

(n+1)/2

- Delete 37

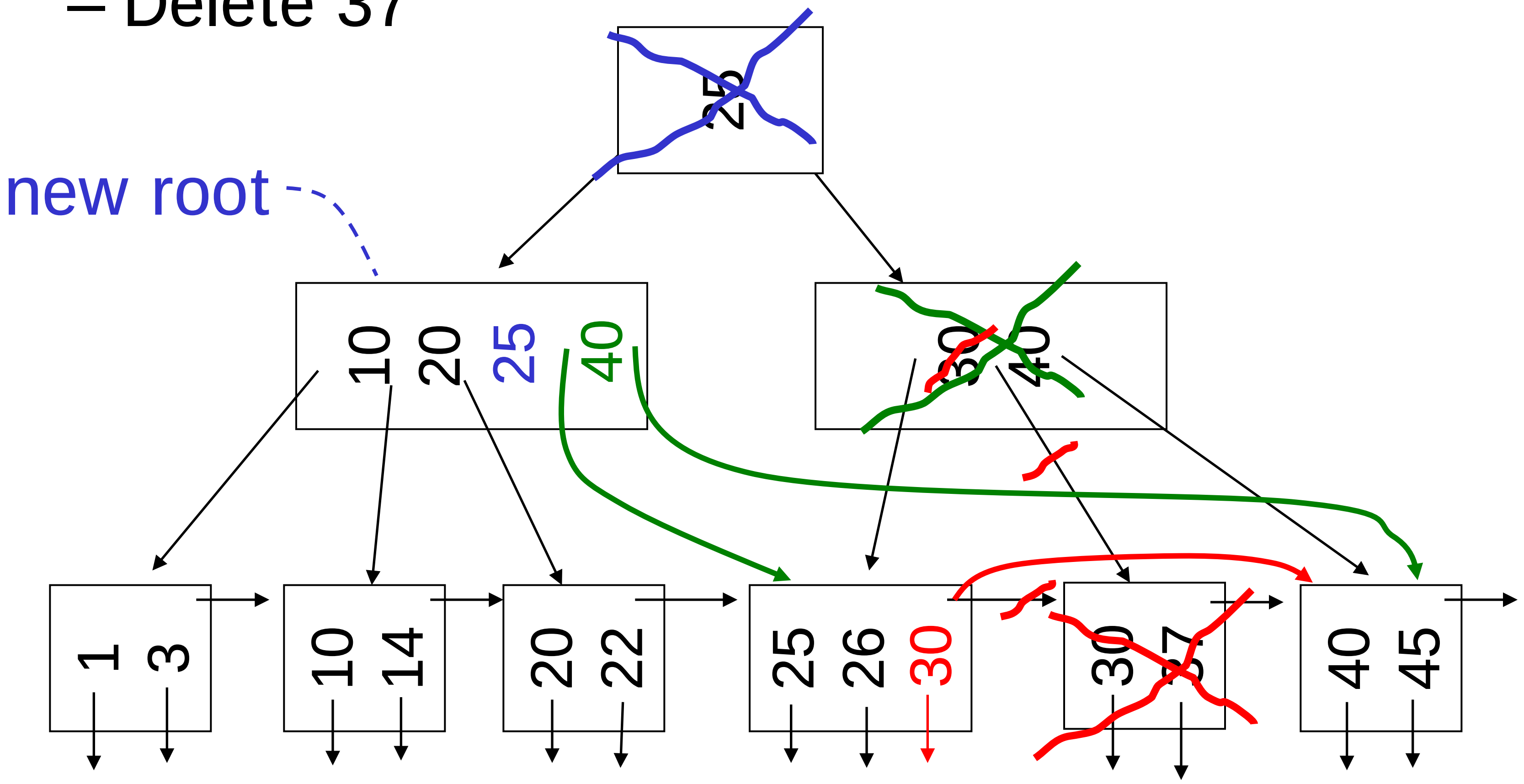


(d) Non-leaf coalese

– Delete 37

n=4

(n+1)/2



Topics

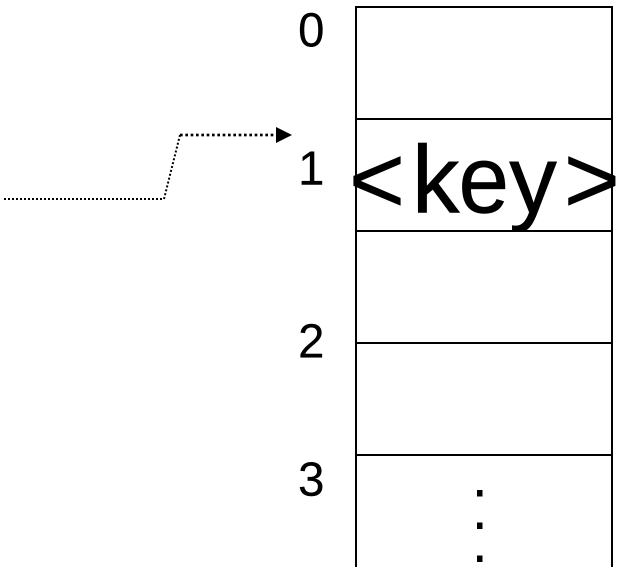
- Conventional Indexes
- B-trees

→ Hashing Schemes

- Multidimensional Indexes

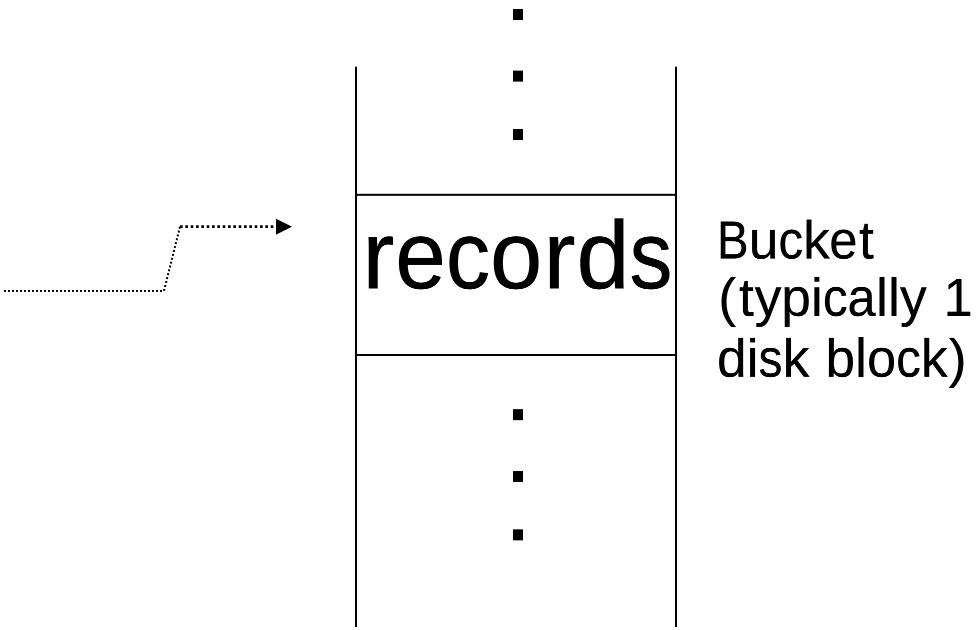
Hashing

key → h(key)



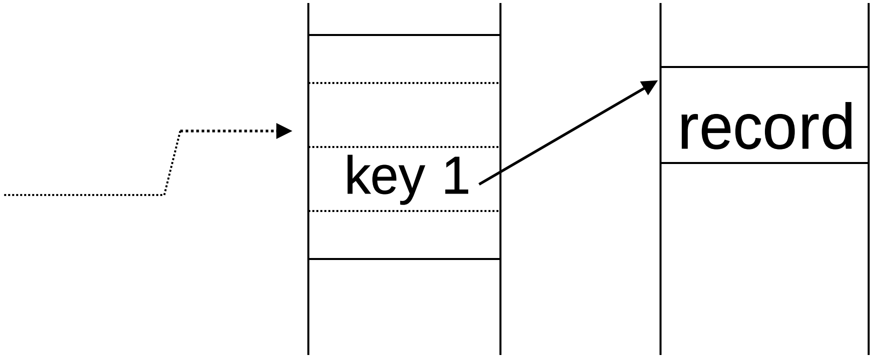
Two alternatives

(1) key → h(key)



Two alternatives

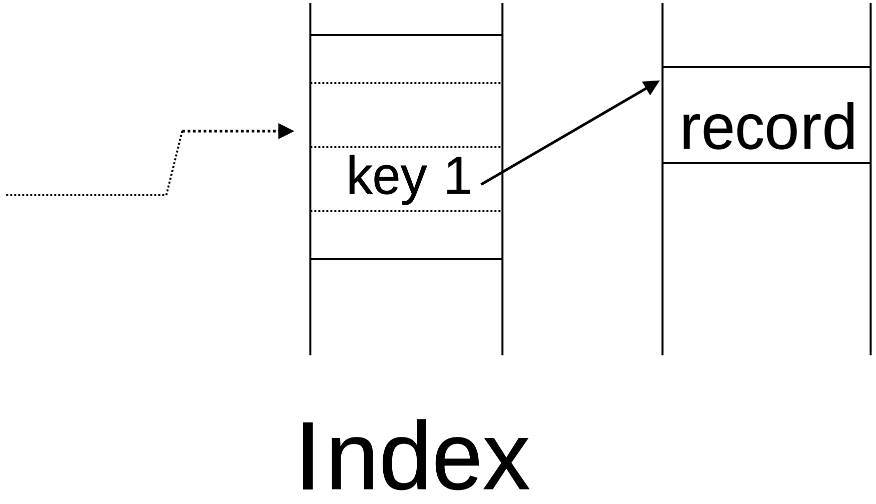
(2) key → h(key)



Index

Two alternatives

(2) key → h(key)



- Alt (2) for “secondary” search key

Example hash function

- Key = ‘x_1 x_2 … x_n’ *n* byte character string
- Have *b* buckets
- h: add $x_1 + x_2 + \ldots\ldots x_n$
  - compute sum modulo *b*

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

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

Good hash function:

☞ Expected number of hash-value/bucket is the same for all buckets

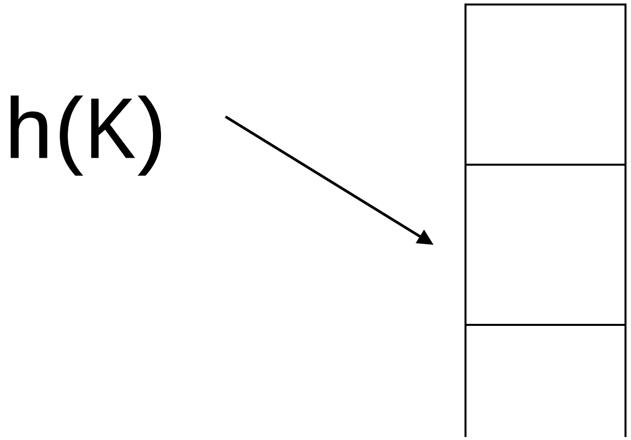
Within a bucket:

- Do we keep keys sorted?

- Yes, if CPU time critical

  & Inserts/Deletes not too frequent

Next: example to illustrate inserts, overflows, deletes



EXAMPLE 2 records/bucket

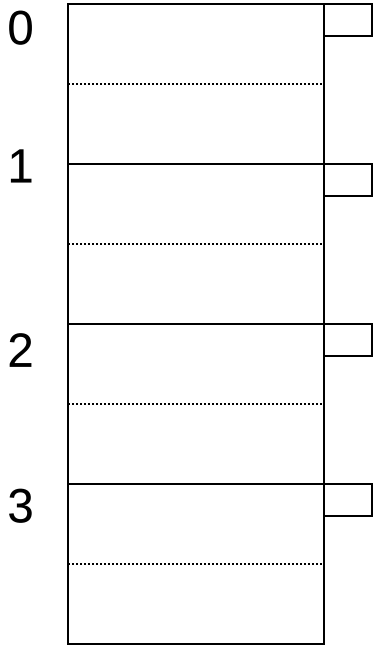
INSERT:

h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0



EXAMPLE 2 records/bucket

INSERT:

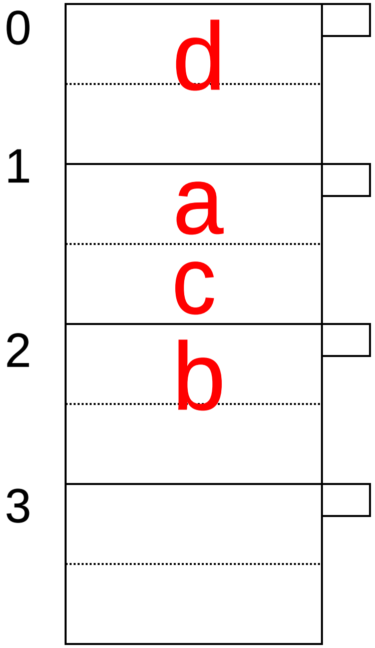
h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0

h(e) = 1



EXAMPLE 2 records/bucket

INSERT:

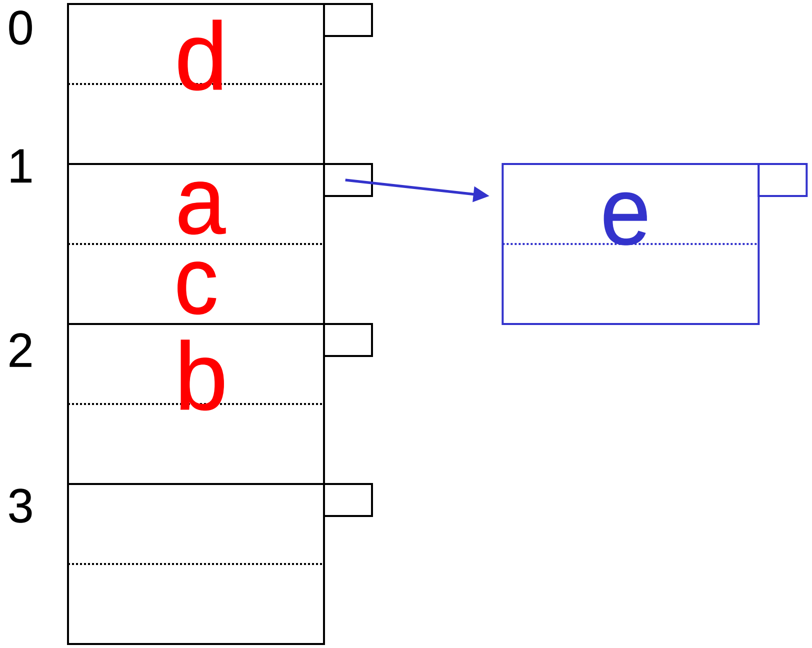
h(a) = 1

h(b) = 2

h(c) = 1

h(d) = 0

h(e) = 1

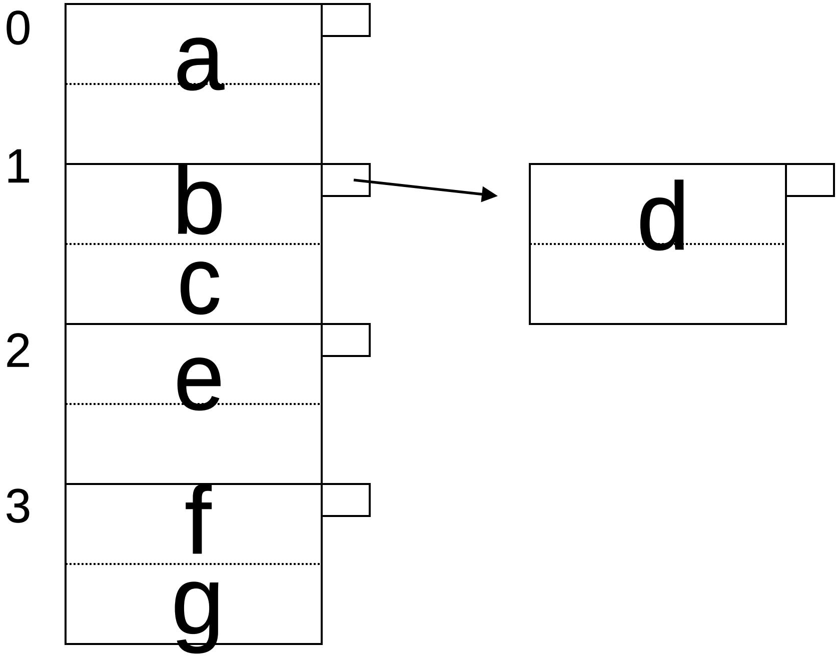


EXAMPLE: deletion

Delete:

e

f



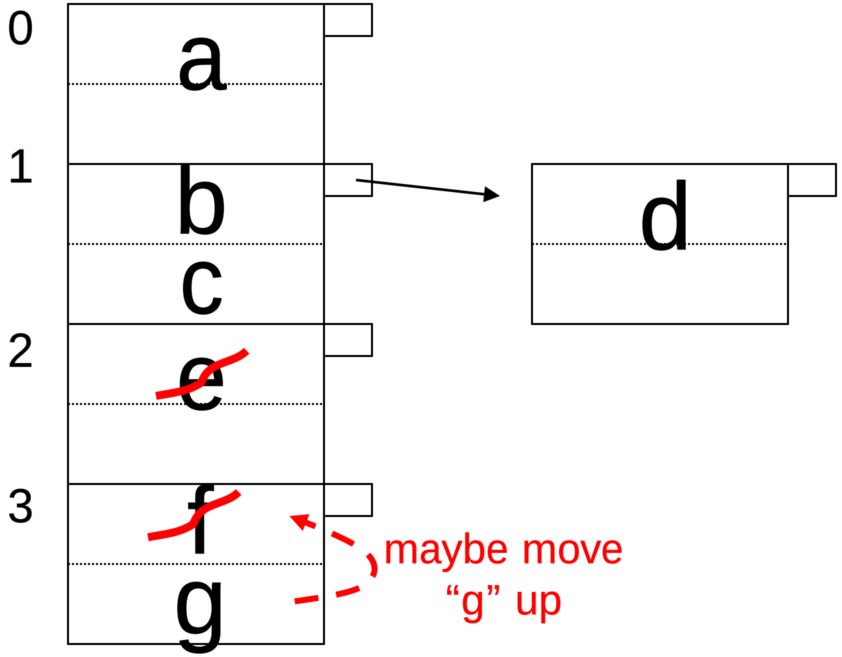
EXAMPLE: deletion

Delete:

e

f

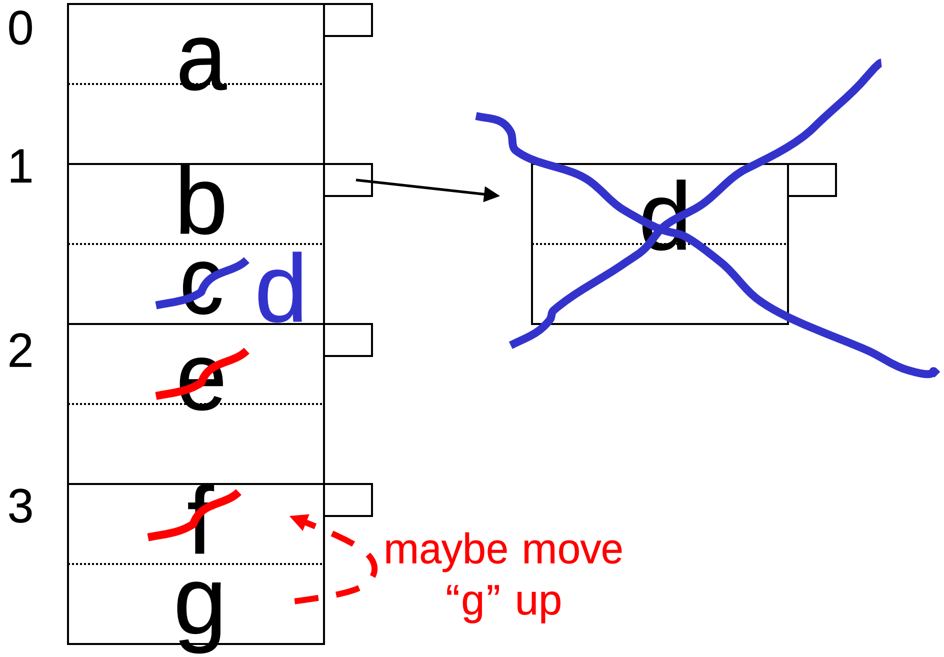
c



EXAMPLE: deletion

Delete:

- e
- f
- c



Rule of thumb:

- Try to keep space utilization between 50% and 80%

$$\text{Utilization} = \frac{\text{\# keys used}}{\text{total \# keys that fit}}$$

Rule of thumb:

- Try to keep space utilization between 50% and 80%

$$\text{Utilization} = \frac{\text{\# keys used}}{\text{total \# keys that fit}}$$

- If < 50%, wasting space
- If > 80%, overflows significant

depends on how good hash function is & on # keys/bucket

How do we cope with growth?

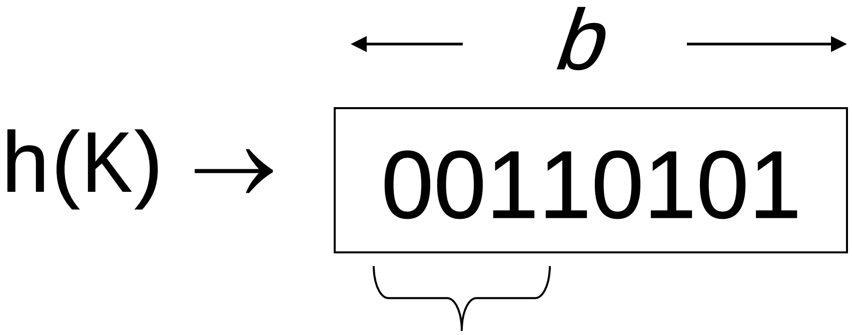
- Overflows and reorganizations
- Dynamic hashing

How do we cope with growth?

- Overflows and reorganizations
- Dynamic hashing
  - Extensible
  - Linear

Extensible hashing: two ideas

(a) Use *i* of *b* bits output by hash function

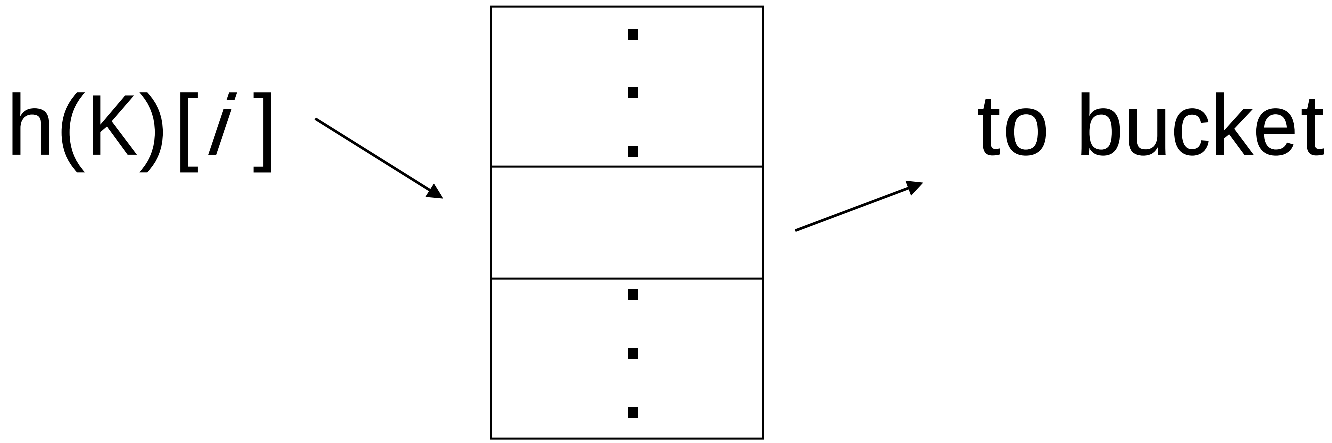


b

h(K) → 00110101

use *i* → grows over time….

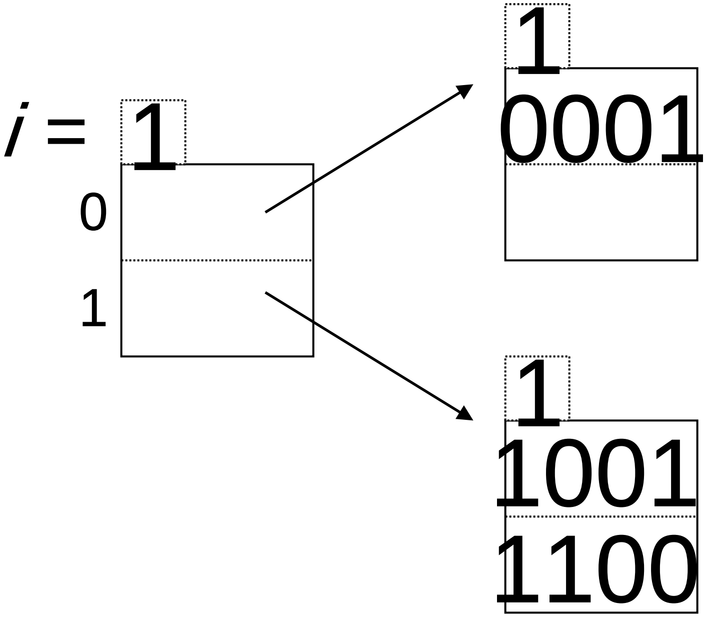
(b) Use directory



h(K)[*i*]

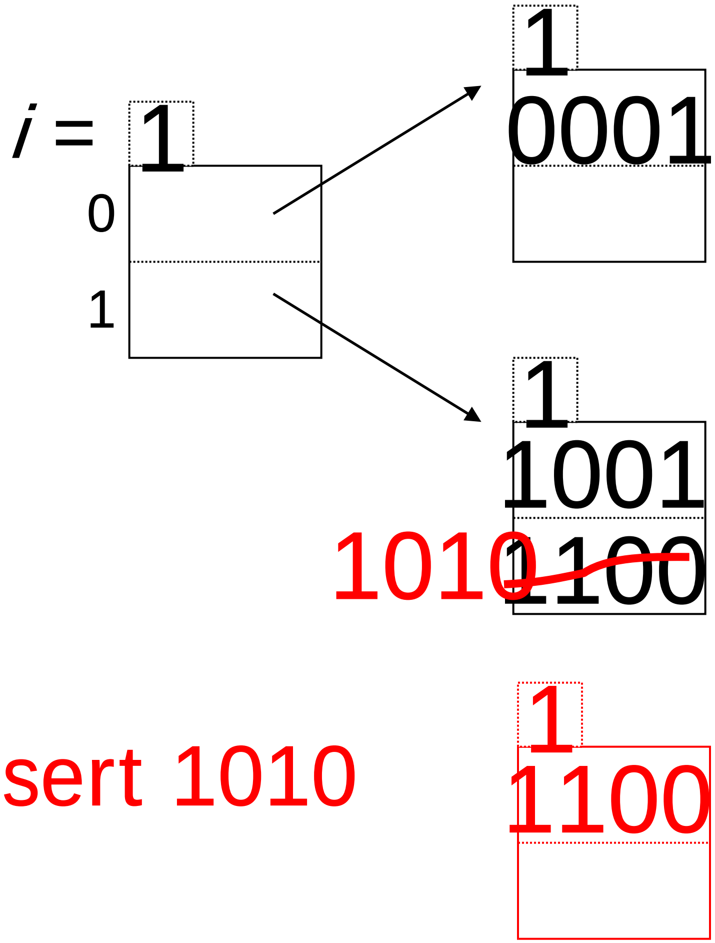
to bucket

Example: h(k) is 4 bits; 2 keys/bucket



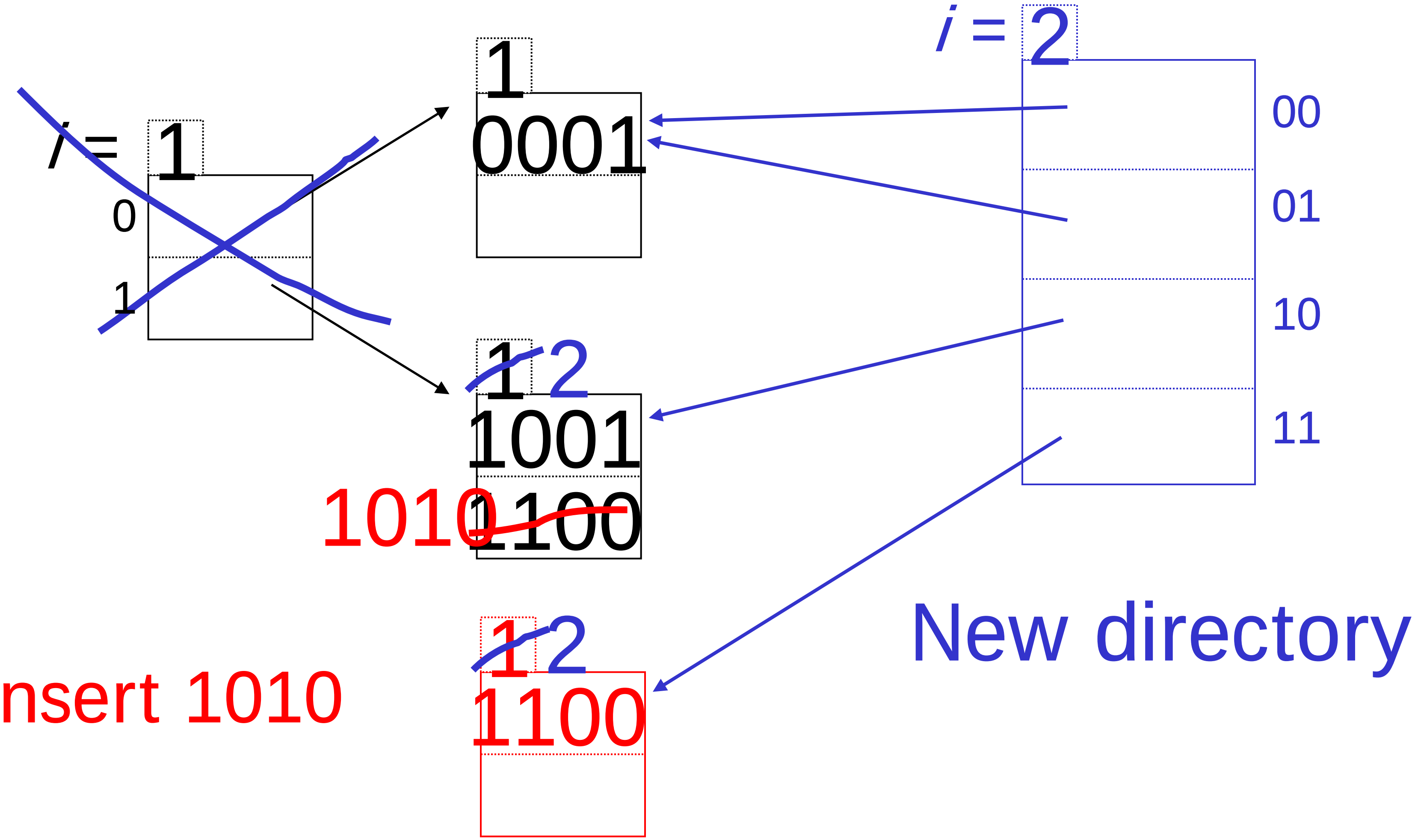
Insert 1010

Example: h(k) is 4 bits; 2 keys/bucket



Insert 1010

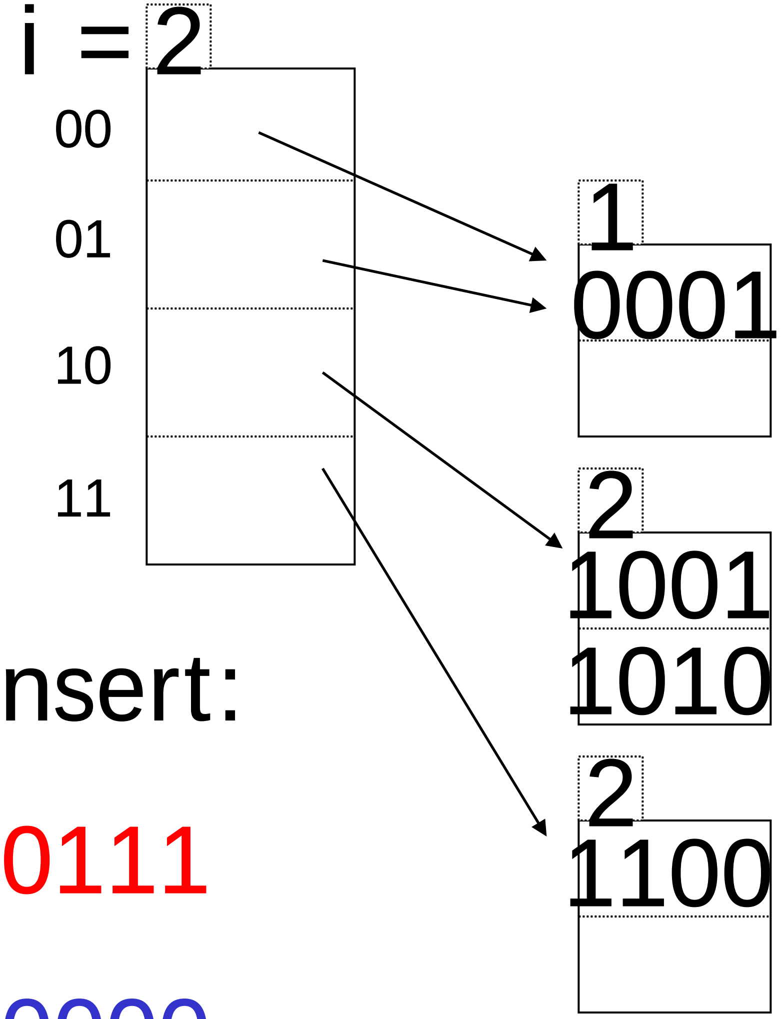
Example: h(k) is 4 bits; 2 keys/bucket



Insert 1010

New directory

Example continued

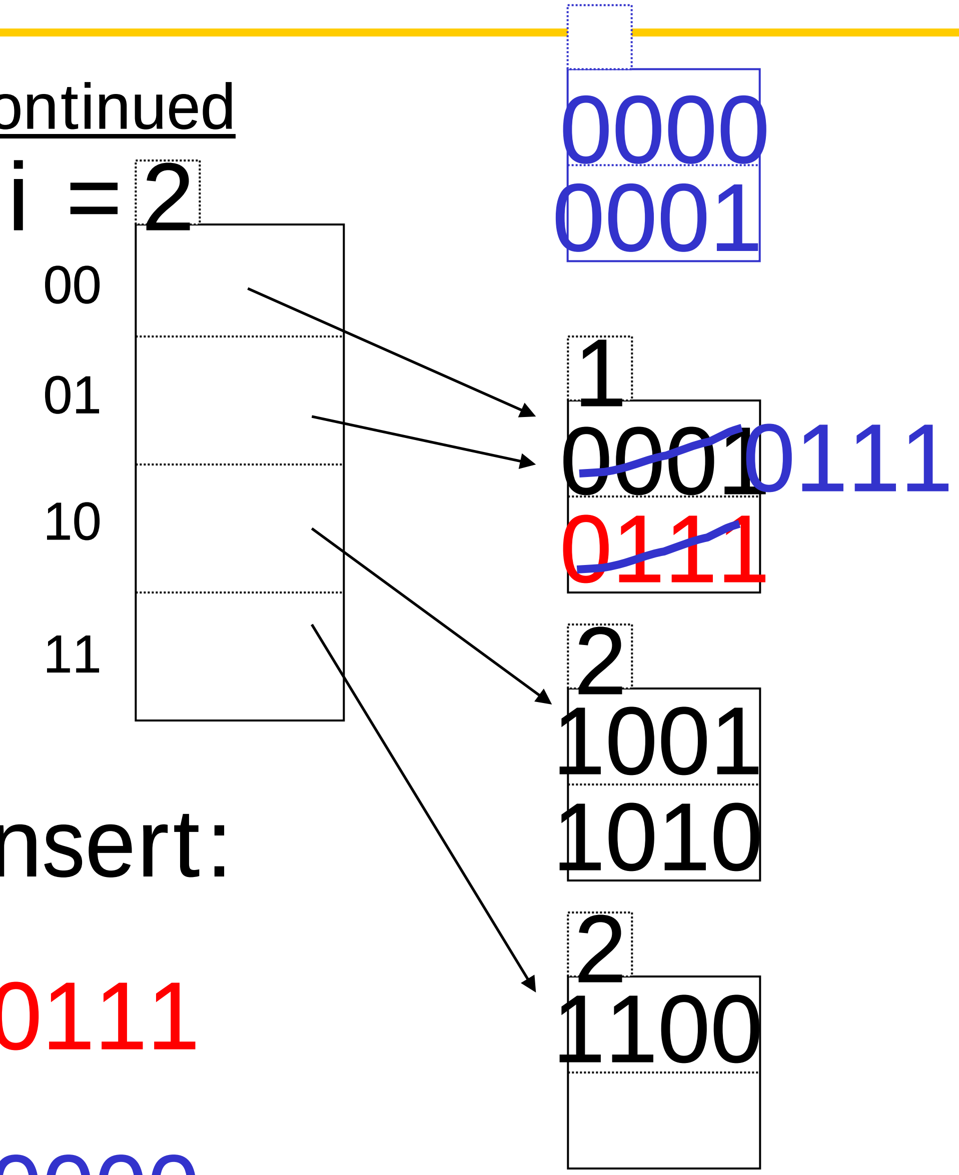


Insert:

0111

0000

Example continued

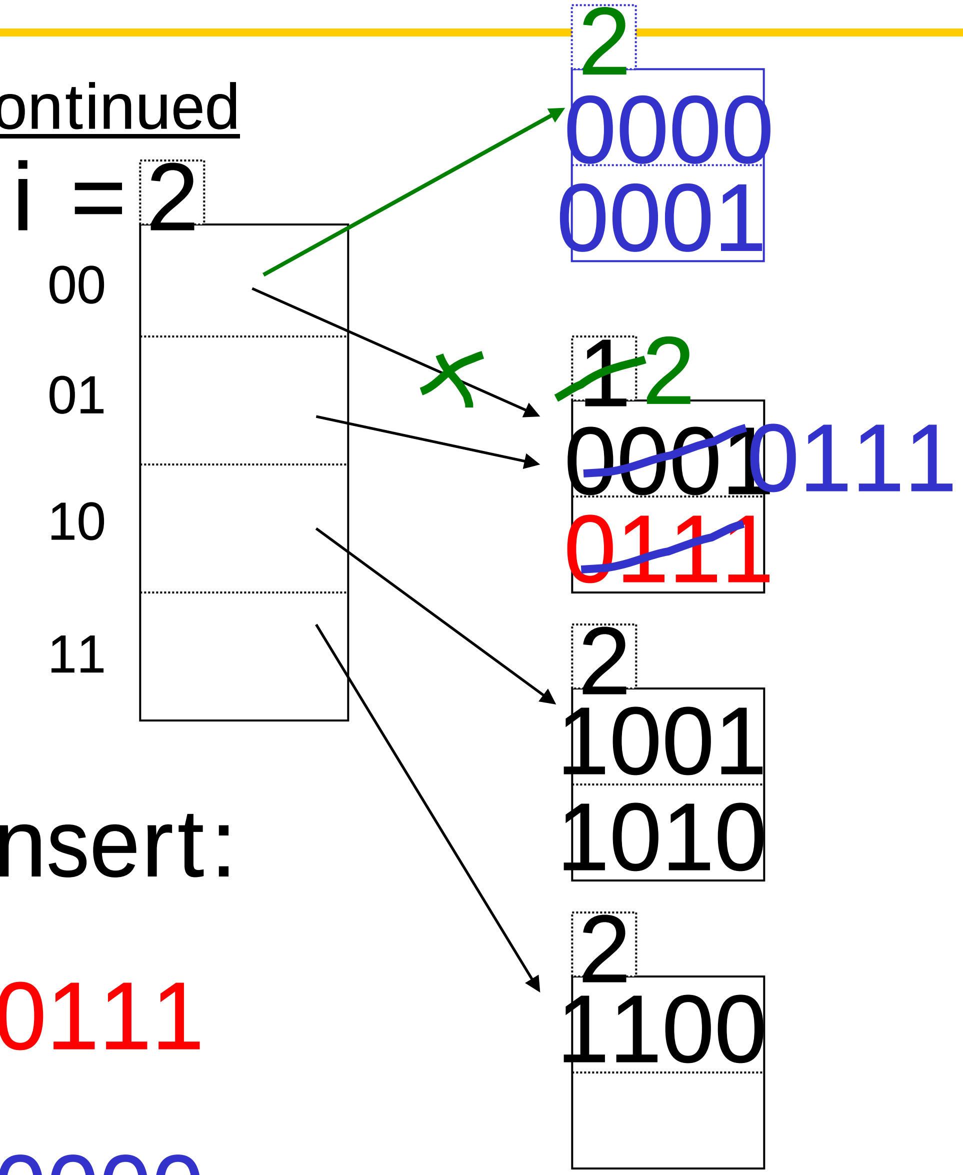


Insert:

0111

0000

Example continued

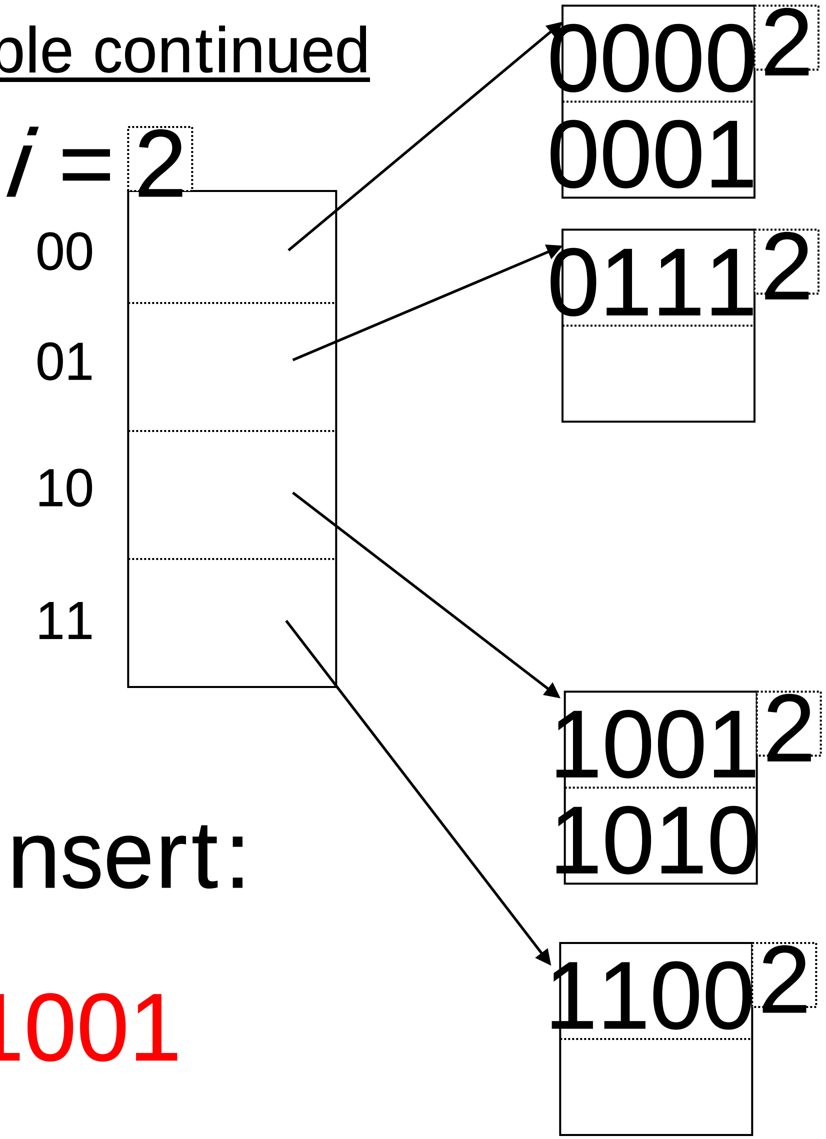


Insert:

0111

0000

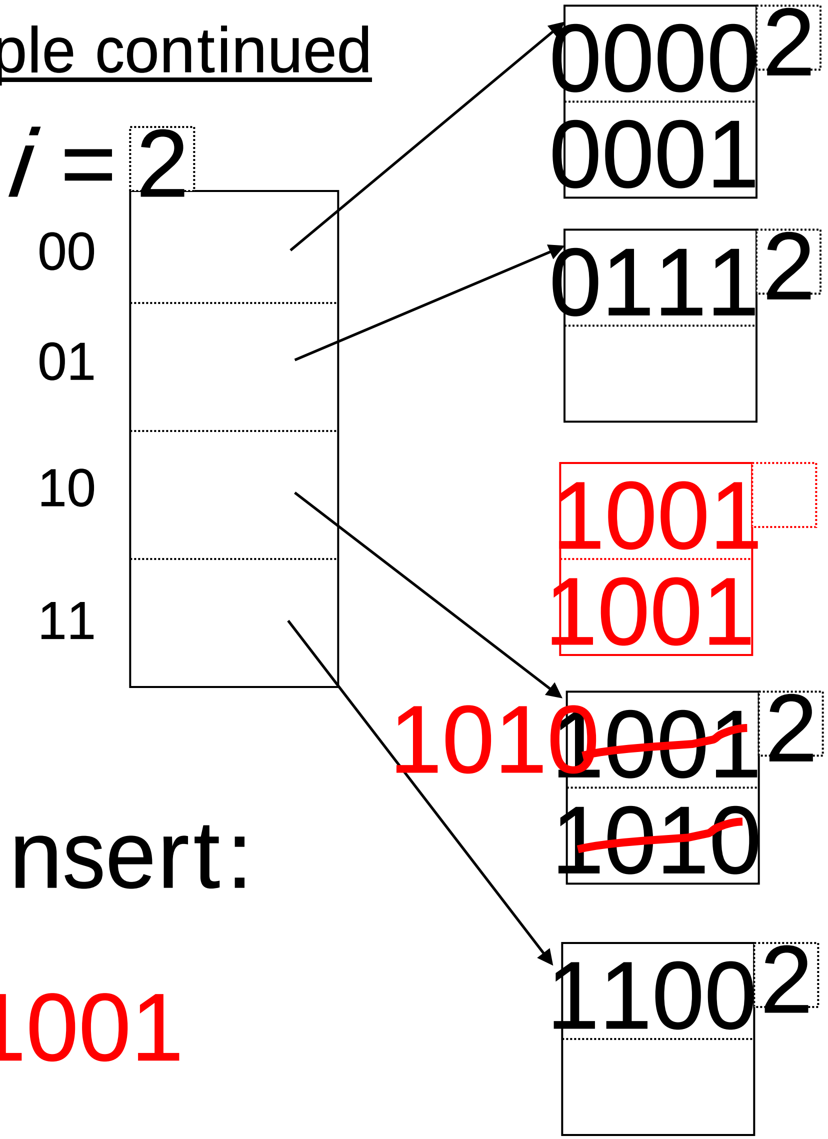
Example continued



Insert:

1001

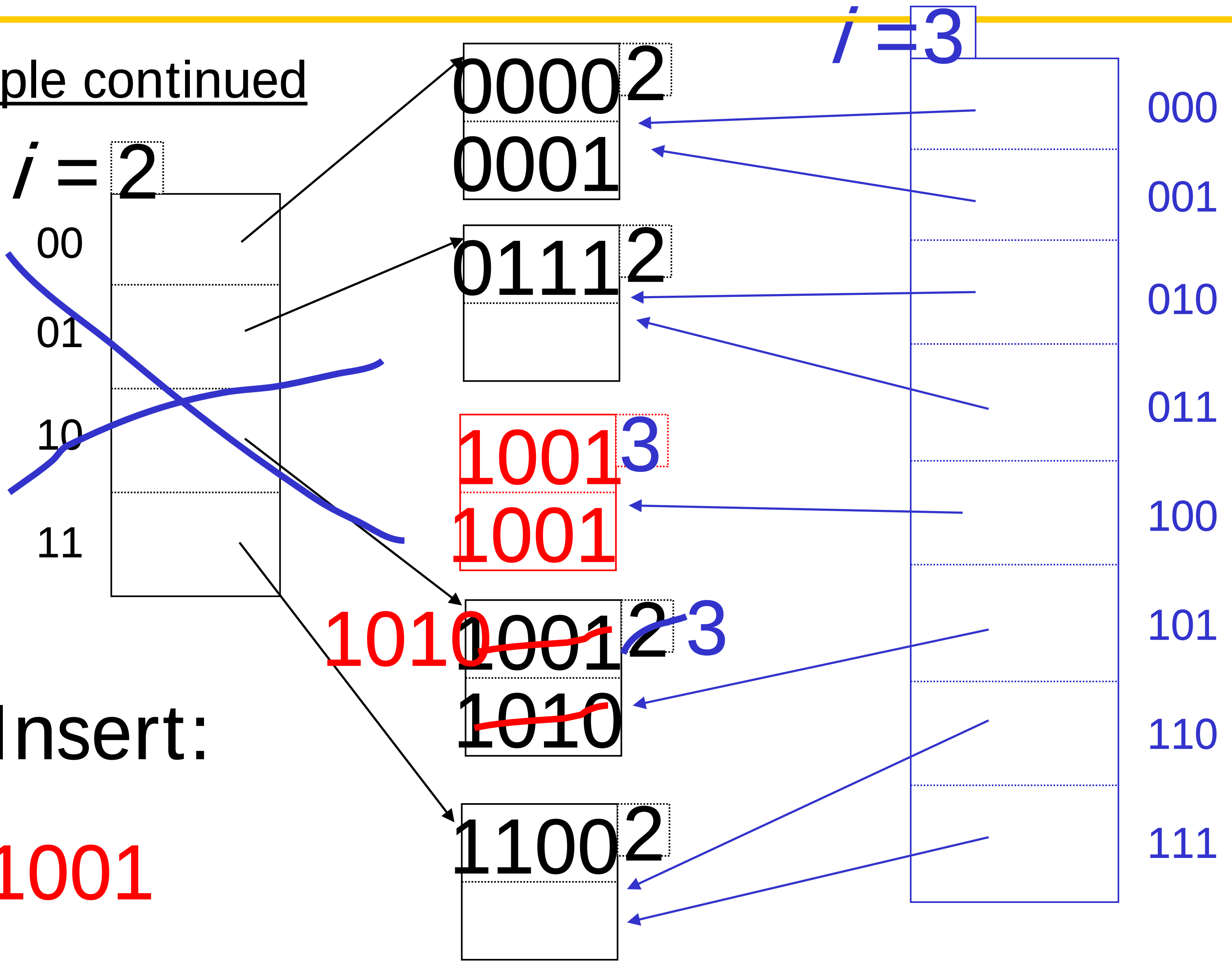
Example continued



Insert:

1001

Example continued



Insert:

1001

Extensible hashing: deletion

- No merging of blocks
- Merge blocks
  and cut directory if possible

  (Reverse insert procedure)

Deletion example:

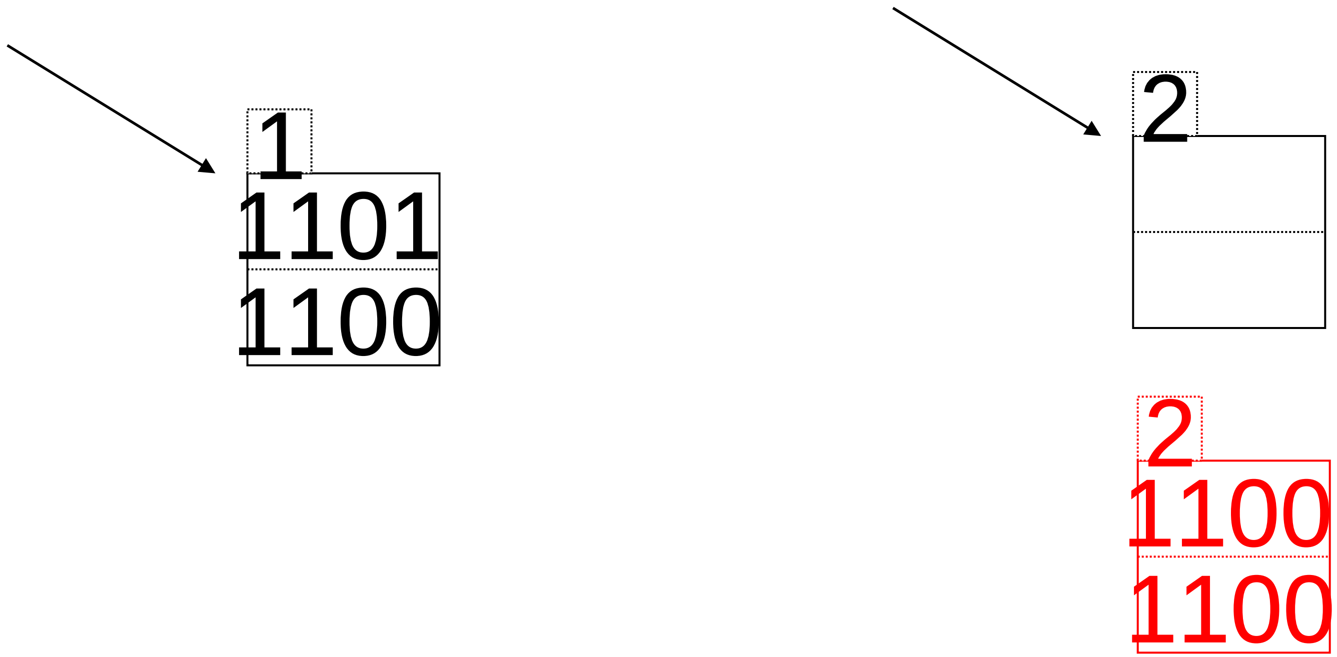
- Run thru insert example in reverse!

Note: Still need overflow chains

- Example: many records with duplicate keys

insert 1100

if we split:



1

1101

1100

2

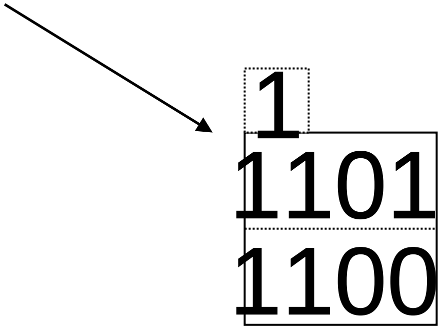
2

1100

1100

Solution: overflow chains

insert 1100

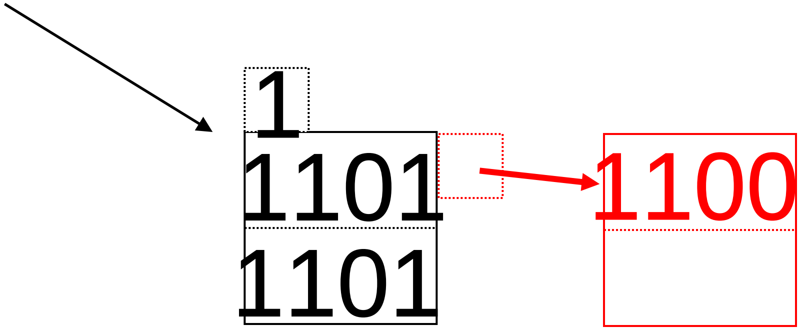


1

1101

1100

add overflow block:



1

1101

1101

1100

Summary Extensible hashing

⊕ Can handle growing files

- with less wasted space
- with no full reorganizations

Summary Extensible hashing

⊕ Can handle growing files

- with less wasted space
- with no full reorganizations

(-) Indirection

(Not bad if directory in memory)

(-) Directory doubles in size

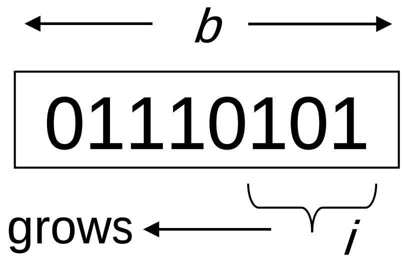
(Now it fits, now it does not)

Linear hashing

- Another dynamic hashing scheme

Two ideas:

(a) Use *i* low order bits of hash

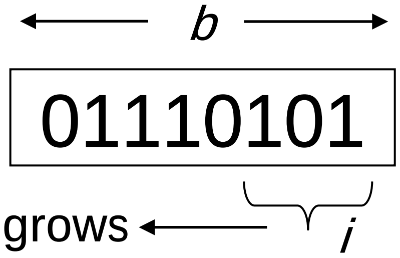


Linear hashing

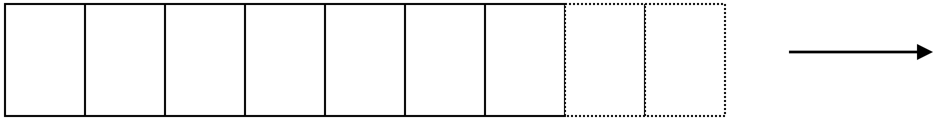
- Another dynamic hashing scheme

Two ideas:

(a) Use *i* low order bits of hash



(b) File grows linearly



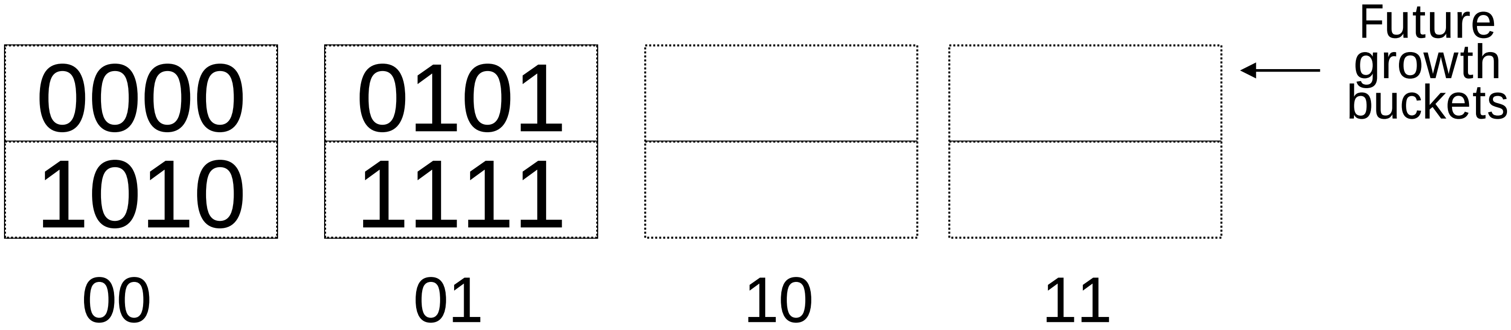
✉ When do we expand file?

- Keep track of: $\dfrac{\text{\# used slots}}{\text{total \# of slots}} = U$

- After every insertion, check if U > threshold then increase buckets by 1

- If you run out of bits, add 1 more
  - 00 becomes 000

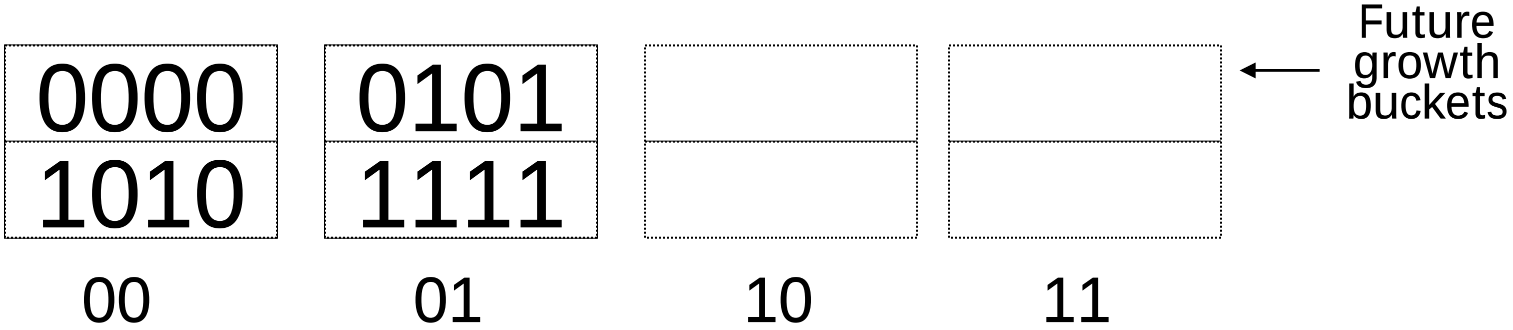
Example b=4 bits, i=2, 2 keys/bucket



00	01	10	11
0000	0101		
1010	1111		

m = 01 (max used block)

Example b=4 bits, i =2, 2 keys/bucket



m = 01 (max used block)

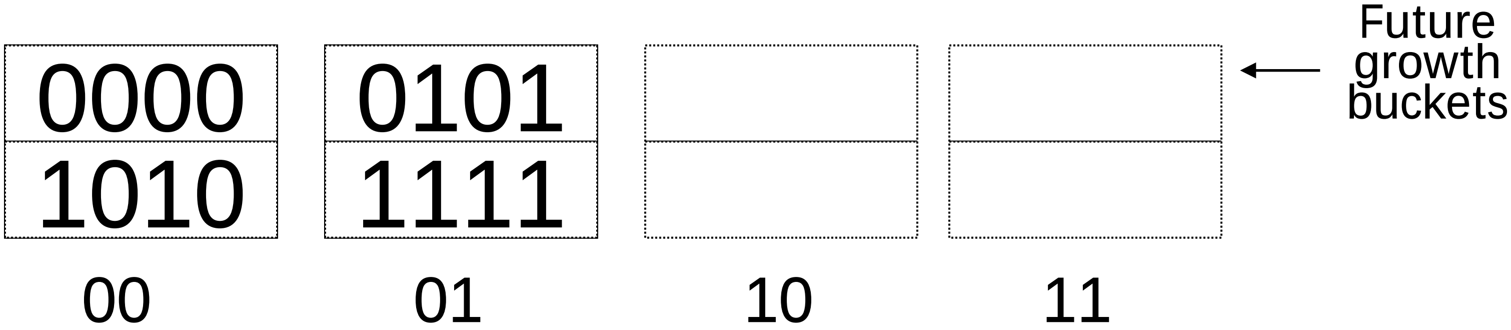
Rule If h(k)[i] ≤ m, then

look at bucket h(k)[i]

else, look at bucket h(k)[i] - 2^{i-1}

Example b=4 bits, i =2, 2 keys/bucket

- insert 0101



0000

1010

0101

1111

00

01

10

11

Future growth buckets

m = 01 (max used block)

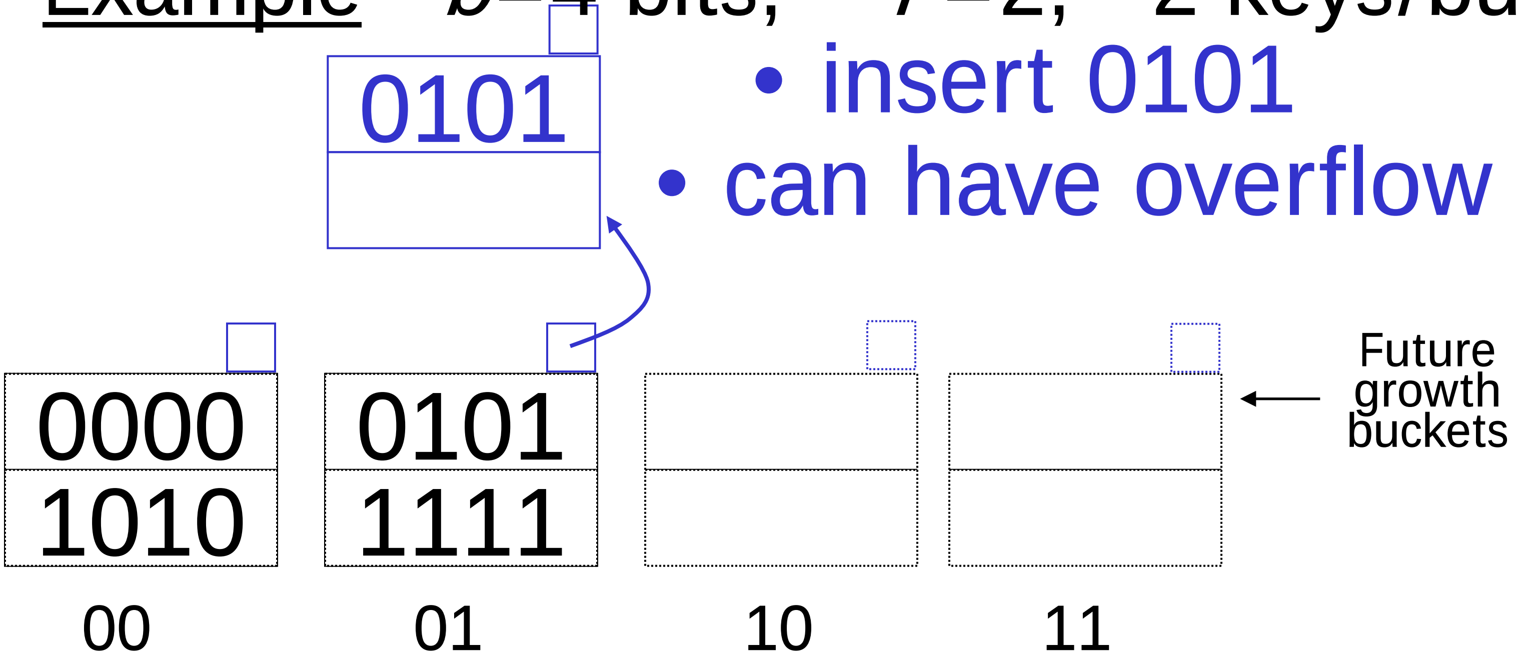
Rule If h(k)[i] $\leq m$, then

look at bucket h(k)[i]

else, look at bucket h(k)[i] - 2^{i-1}

Example b=4 bits, i =2, 2 keys/bucket

- insert 0101
- can have overflow chains!



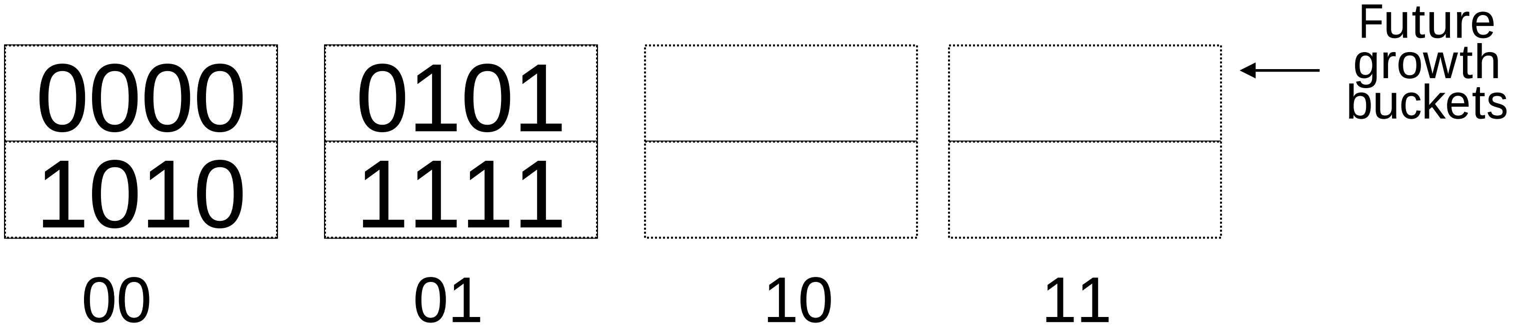
m = 01 (max used block)

Rule If h(k)[i] ≤ m, then

look at bucket h(k)[i]

else, look at bucket h(k)[i] - 2^{i-1}

Example $b=4$ bits, $i=2$, 2 keys/bucket

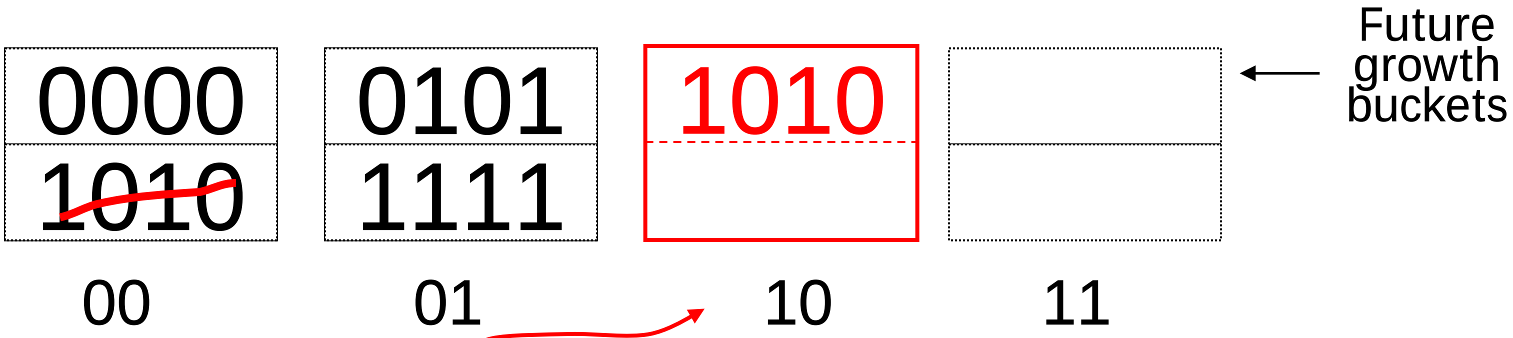


m = 01 (max used block)

- After every insertion, check if U > threshold then increase buckets by 1

$$U = \frac{\underline{\#\text{ used slots}}}{\text{total }\#\text{ of slots}}$$

Example b=4 bits, i =2, 2 keys/bucket



0000

~~1010~~

0101

1111

1010

Future growth buckets

00

01

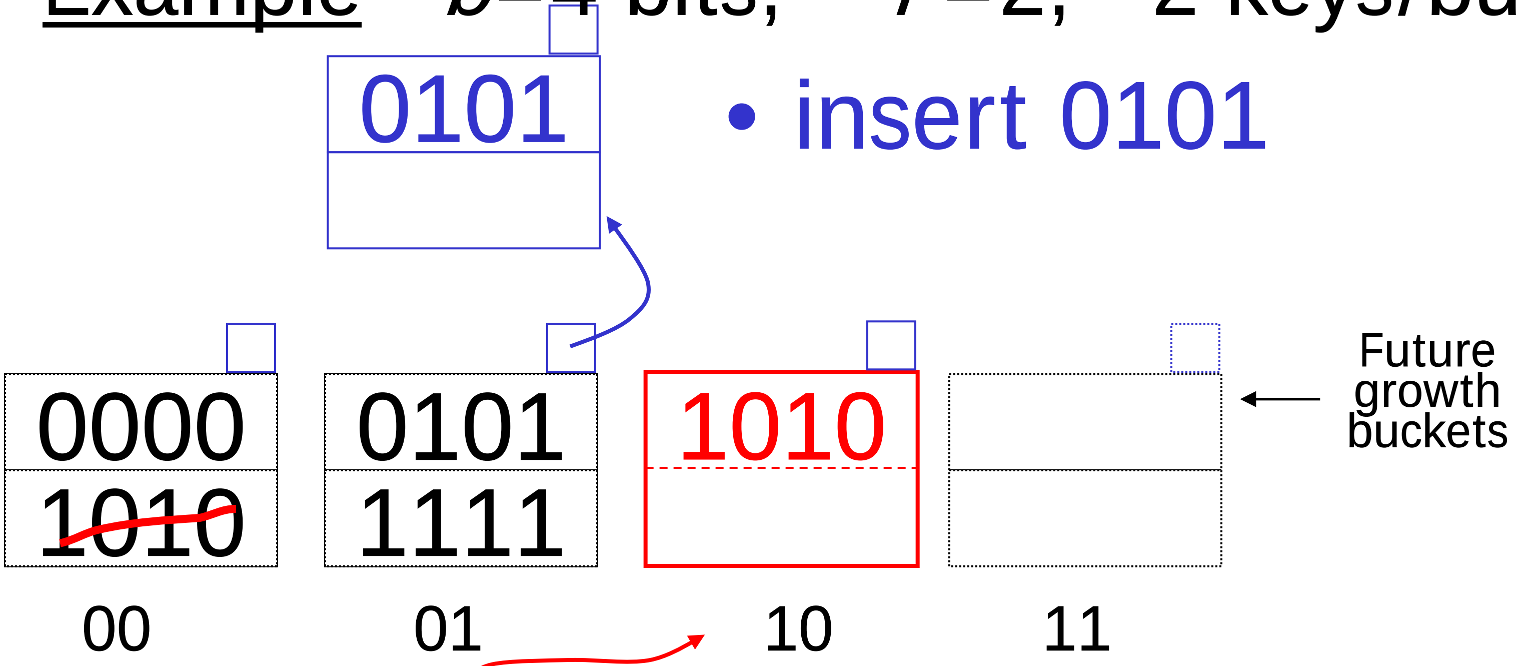
10

11

m = ~~01~~ 10 (max used block)

<u>Example</u> b=4 bits, i =2, 2 keys/bucket

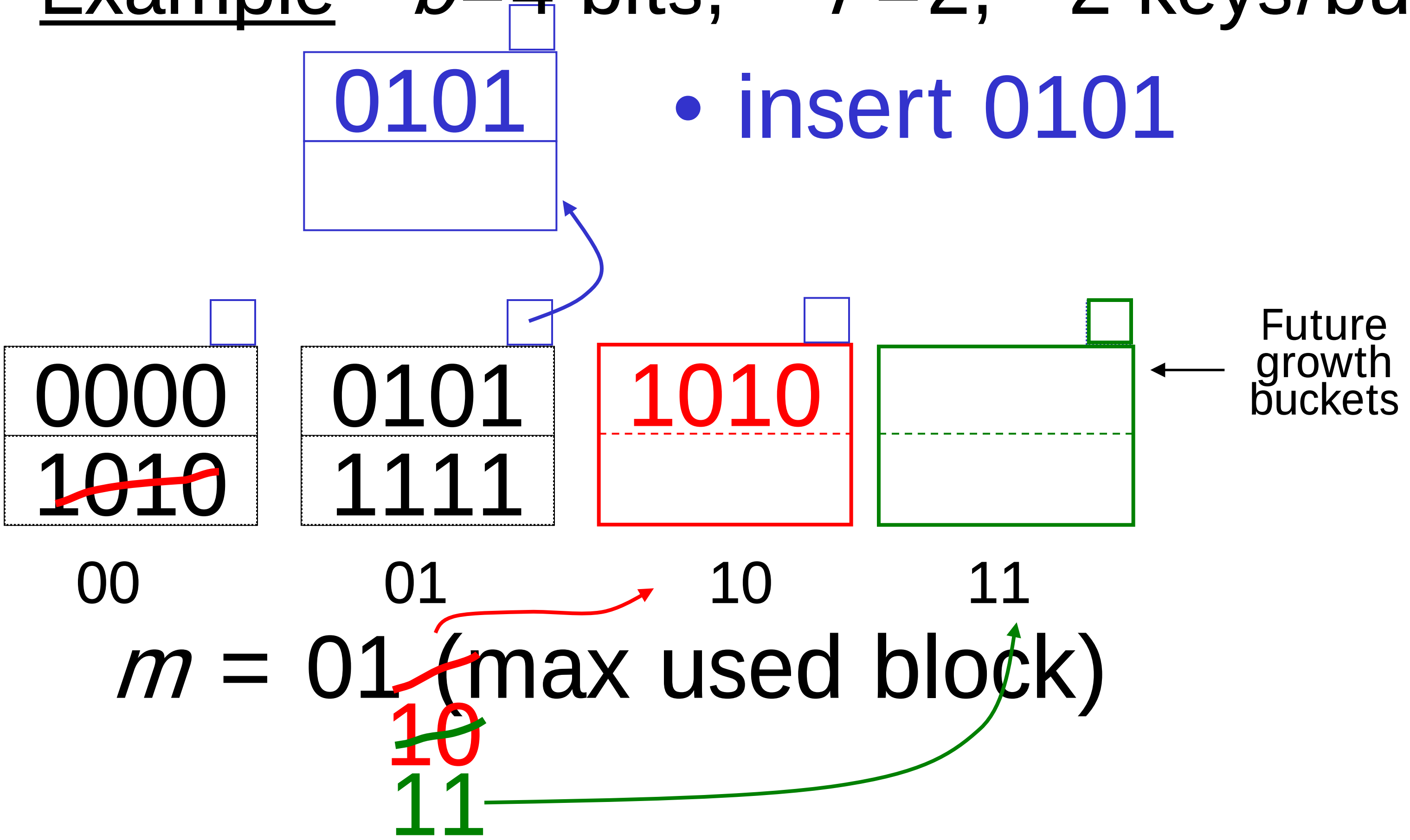
- insert 0101



m = ~~01~~ 10 (max used block)

Example b=4 bits, i =2, 2 keys/bucket

- insert 0101



0101

00	01	10	11
0000	0101	1010	
~~1010~~	1111		

Future growth buckets

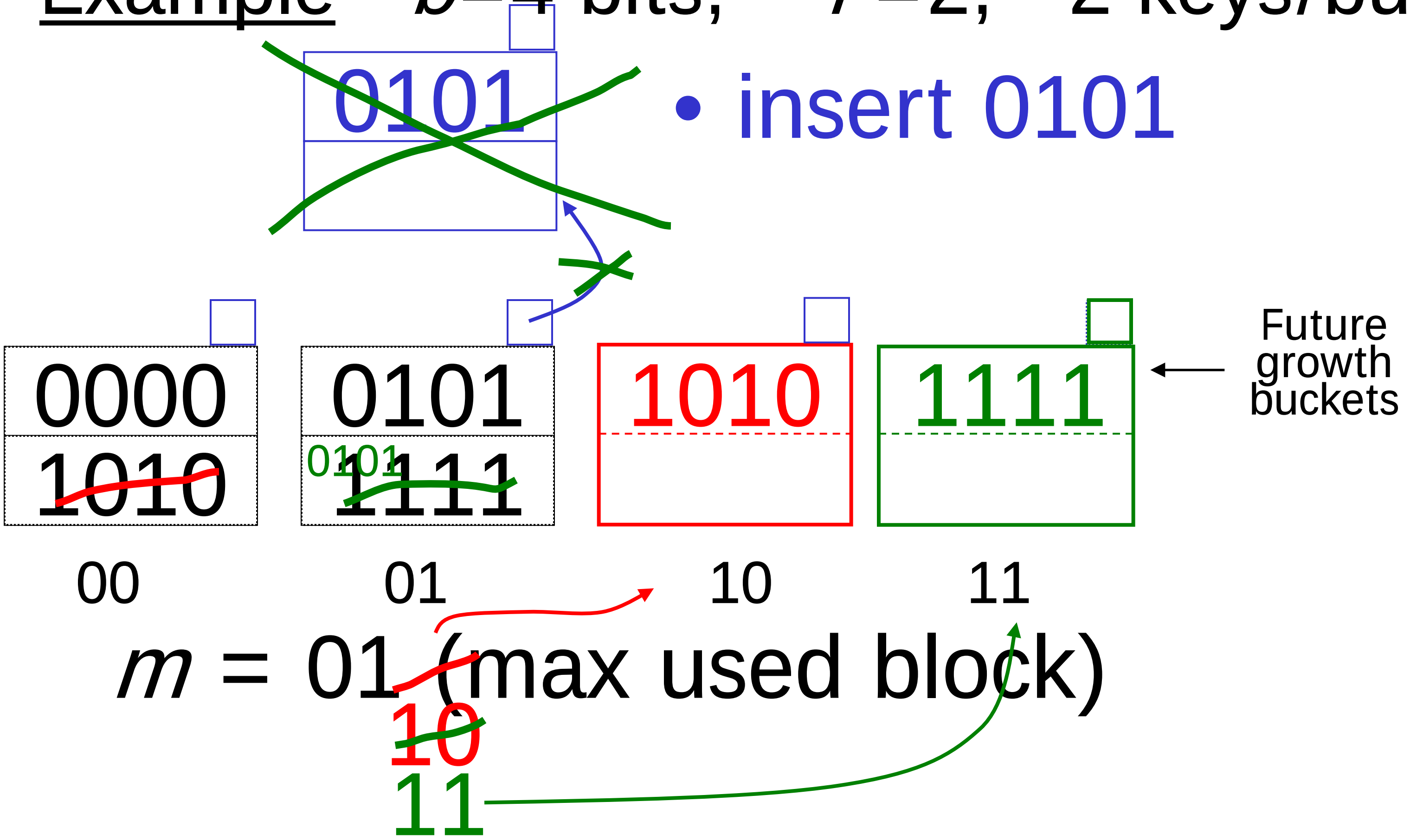
m = ~~01~~ (max used block)

~~10~~

11

Example *b*=4 bits, *i* =2, 2 keys/bucket

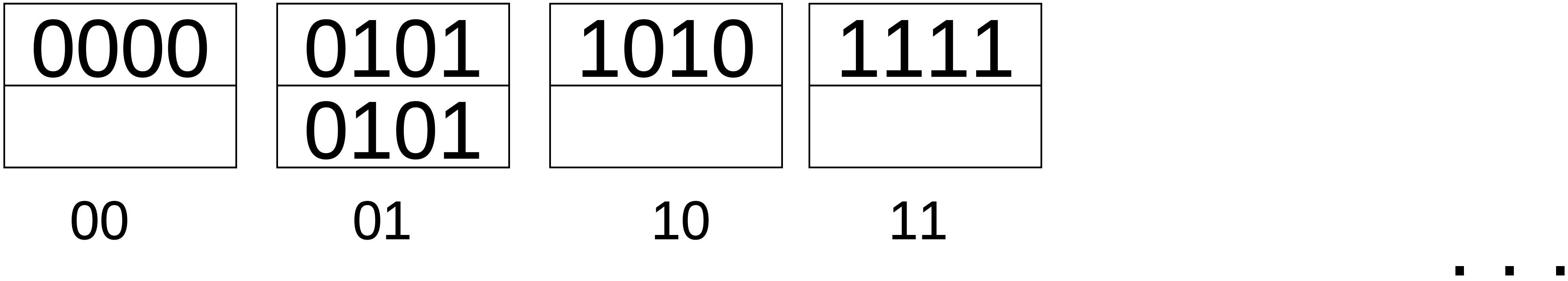
- insert 0101



m = ~~01~~ (max used block)
~~10~~
11

Example Continued: How to grow beyond this?

i = 2



0000

0101
0101

1010

1111

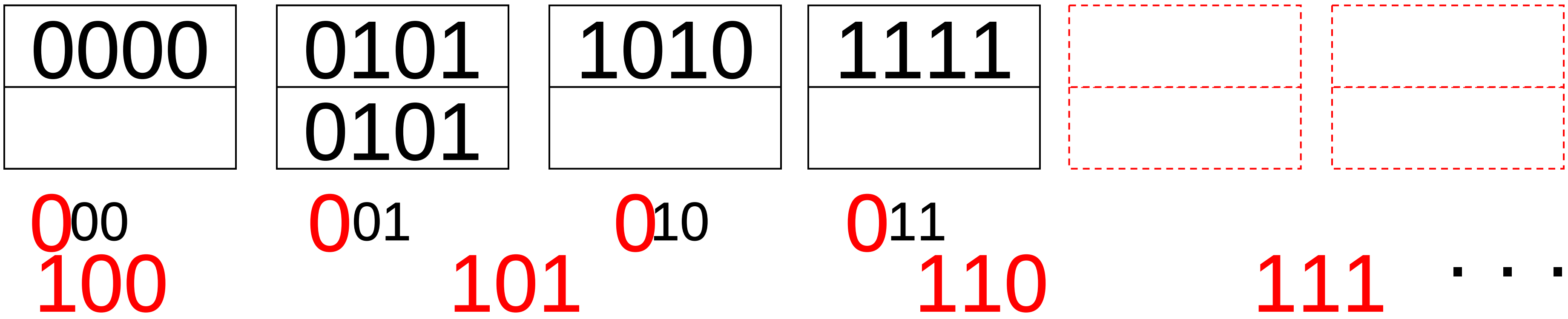
00 01 10 11

. . .

m = 11 (max used block)

Example Continued: How to grow beyond this?

$i = \cancel{2}3$



0000

0101
0101

1010

1111

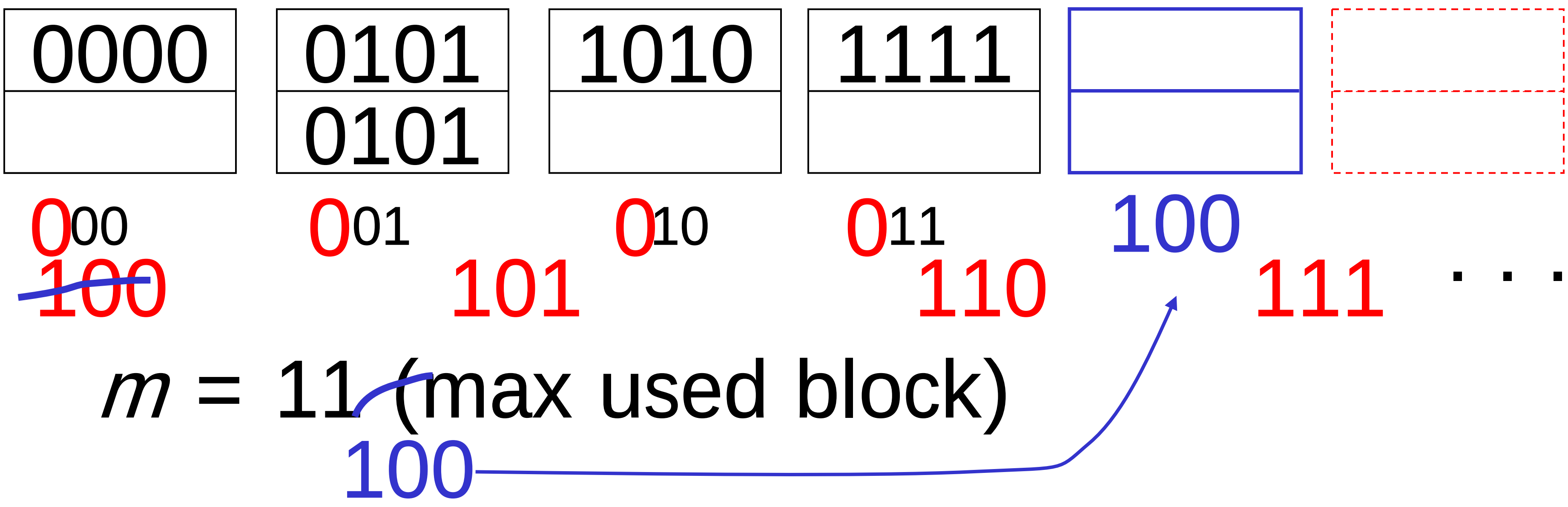
000 001 010 011

100 101 110 111 · · ·

m = 11 (max used block)

Example Continued: How to grow beyond this?

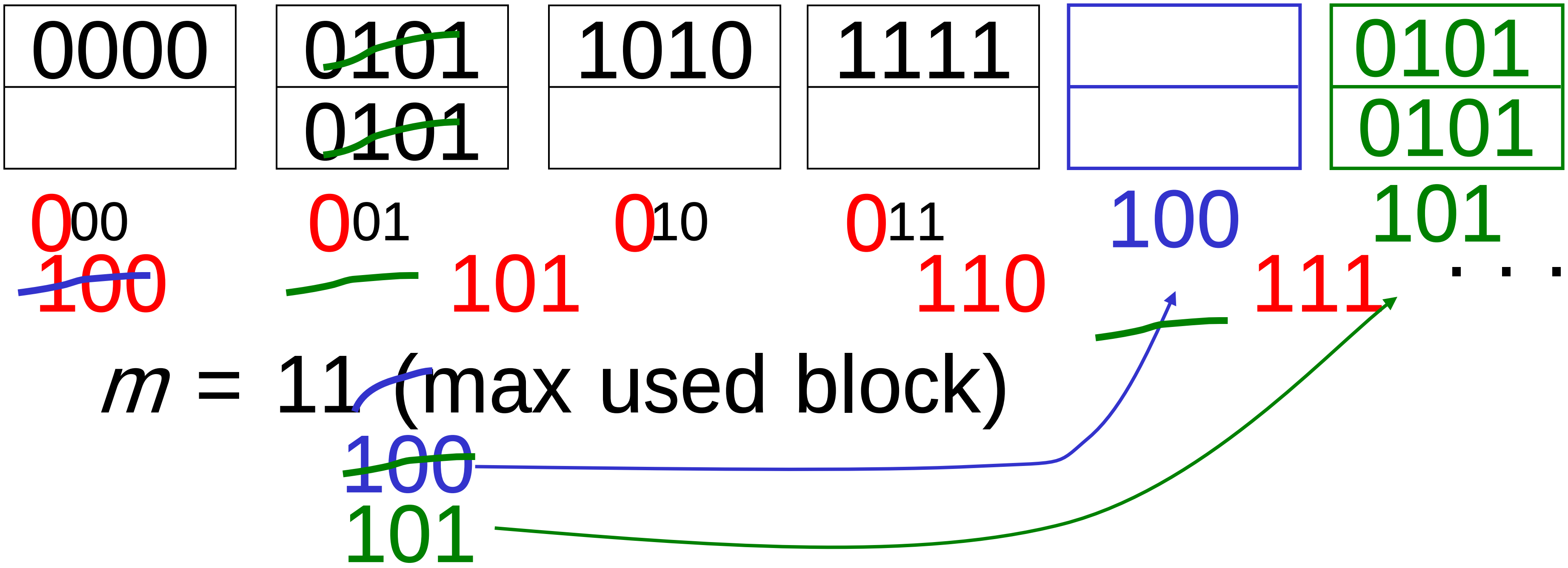
i = ~~2~~ 3



m = ~~11~~ 100 (max used block)

Example Continued: How to grow beyond this?

$i =$ ~~2~~ 3



0000

~~0101~~
~~0101~~

1010

1111

0101
0101

000
~~100~~

001
101

010

011
110

100

101
111

. . .

m = ~~11~~ (max used block)
~~100~~
101

Summary Linear Hashing

⊕ Can handle growing files

- with less wasted space
- with no full reorganizations

⊕ No indirection like extensible hashing

⊖ Can still have overflow chains