Dictionary в Python

По мотивам Objects/dictnotes.txt

Cyril @notorca Lashkevich

Как создать словарь

- {}
- dict()
- PyObject* PyDict_New()

Сколько словарей в Hello World?

```
$ python -c "print('Hello world')"
| wc -l
```

Сколько словарей в Hello World?

```
$ python -c "print('Hello world')"
| wc -l
```

1642

Именованные параметры функций

- 1 запись, 1 чтение
- 1-3 элемента
- Часто встречается в обычных программах на Python

Поиск метода в классе

- 1 запись, много чтений
- 8-16 элементов
- При наследовании много неудачных чтений с последующим поиском в базовом классе

Атрибуты и глобальные пременные

- Много записей и чтений
- 4-10 элементов

Builtins

- Частые чтение, почти не бывает записи
- ~150 строковых ключей (3.3)
- По некоторым ключам чтения гораздо чаще чем по другим

Удаление повторов, подсчет элементов

- Одинократное чтение по каждому из ключей
- Произвольное количество элементов
- Многократный доступ по одному ключу подряд

Удаление дубликатов

```
dict.fromkeys(seqn).keys()
```

- Все операции записи при конструировании

Подсчет элементов в последовательности

```
for e in seqn:
    d[e] = d.get(e,0) + 1
```

- 2 последовательных доступа по одинаковому ключу

Создание индекса из словаря списков

```
for pnum, page in enumerate(pages):
    for w in page:
        d.setdefault(w, []).append(pnum)
```

- setdefault совмещает 2 поиска в 1м

Проверка принадлежности

- Словари произвольных размеров
- Создаются 1 раз и затем мало изменяются
- Много вызовов has_key() и __contains__()

Динамические отображения

- Чередующиеся добавления, удаления, чтение и перезапись элементов

Реализация (2.7)

- Последовательная область памяти с доступом по индксу

```
typedef struct {
    Py_hash_t me_hash;
    PyObject *me_key;
    PyObject *me_value;
} PyDictKeyEntry;
```

Пустой dict с размером по умолчанию (8 элементов)

```
>>> d = {}
```

Idx		Hash	Key	Value
000				
001				
010				
011				
100				
101				
110				
111				

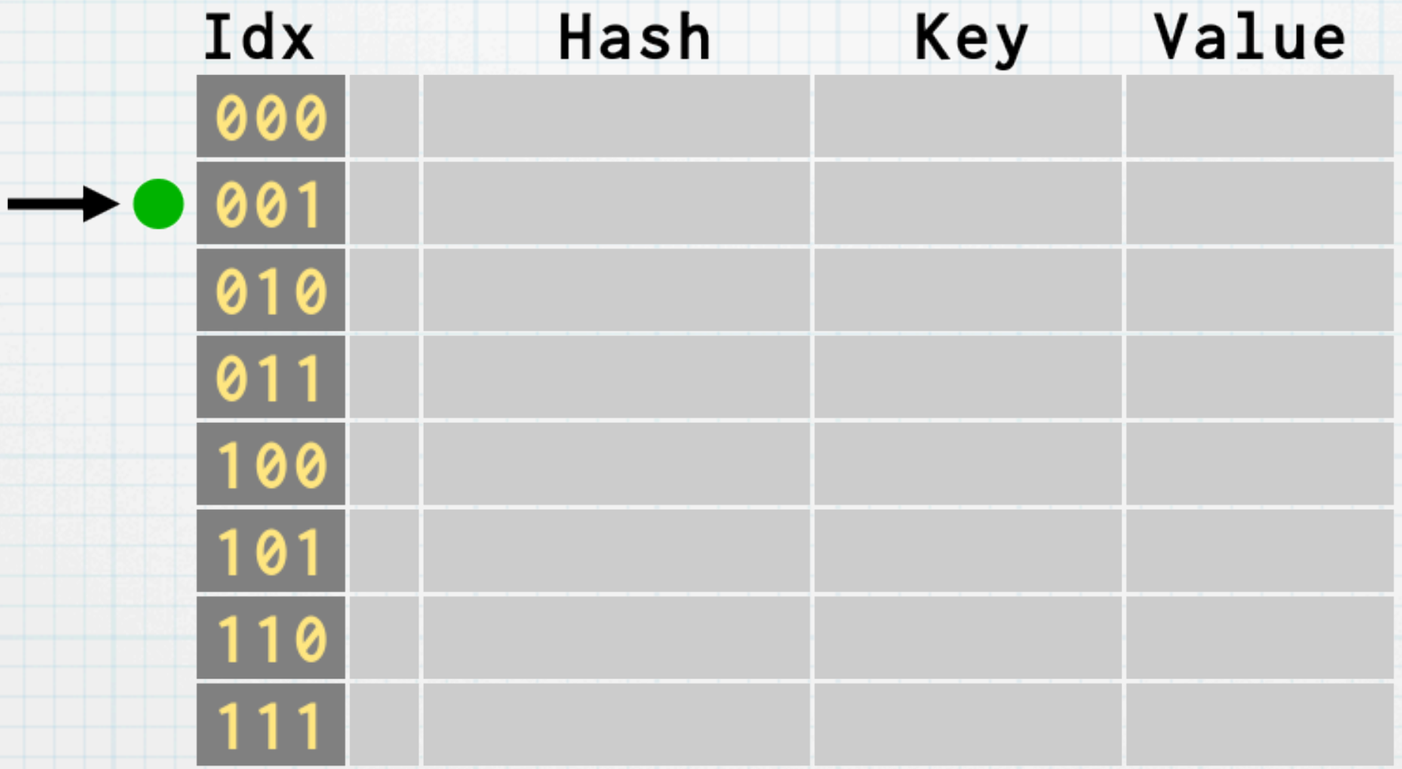
Хеширование ключа

- Ключ преобразуется в индекс с помощъю функции hash()
- hash() возвращает 32/64bit значение
- Для индекса берется n младших бит

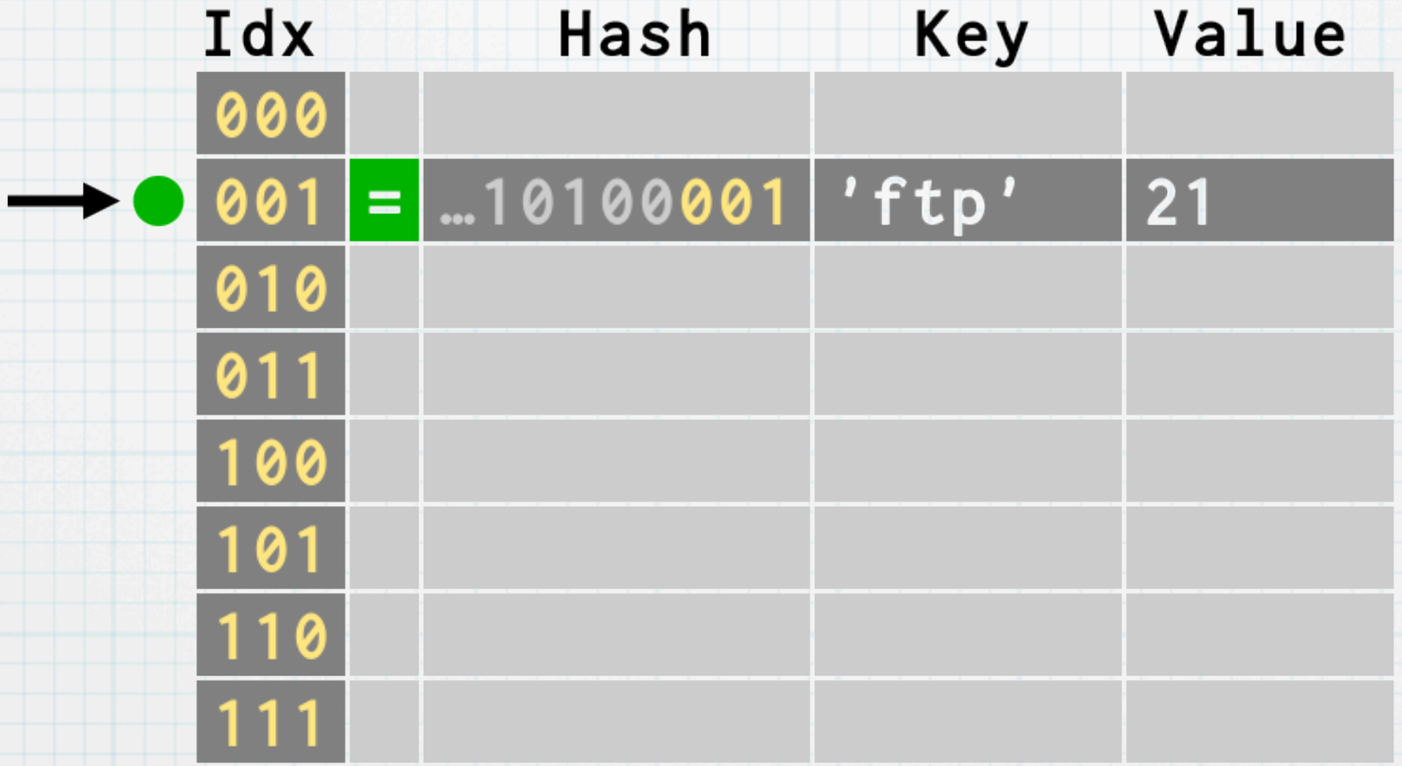
Свойства хеша

- Для равных значений хеши всегда равны
- Даже если представление значений разное: 9, 9.0, complex(9,0)
- Похожие значения дают сильно отличающиеся хеши

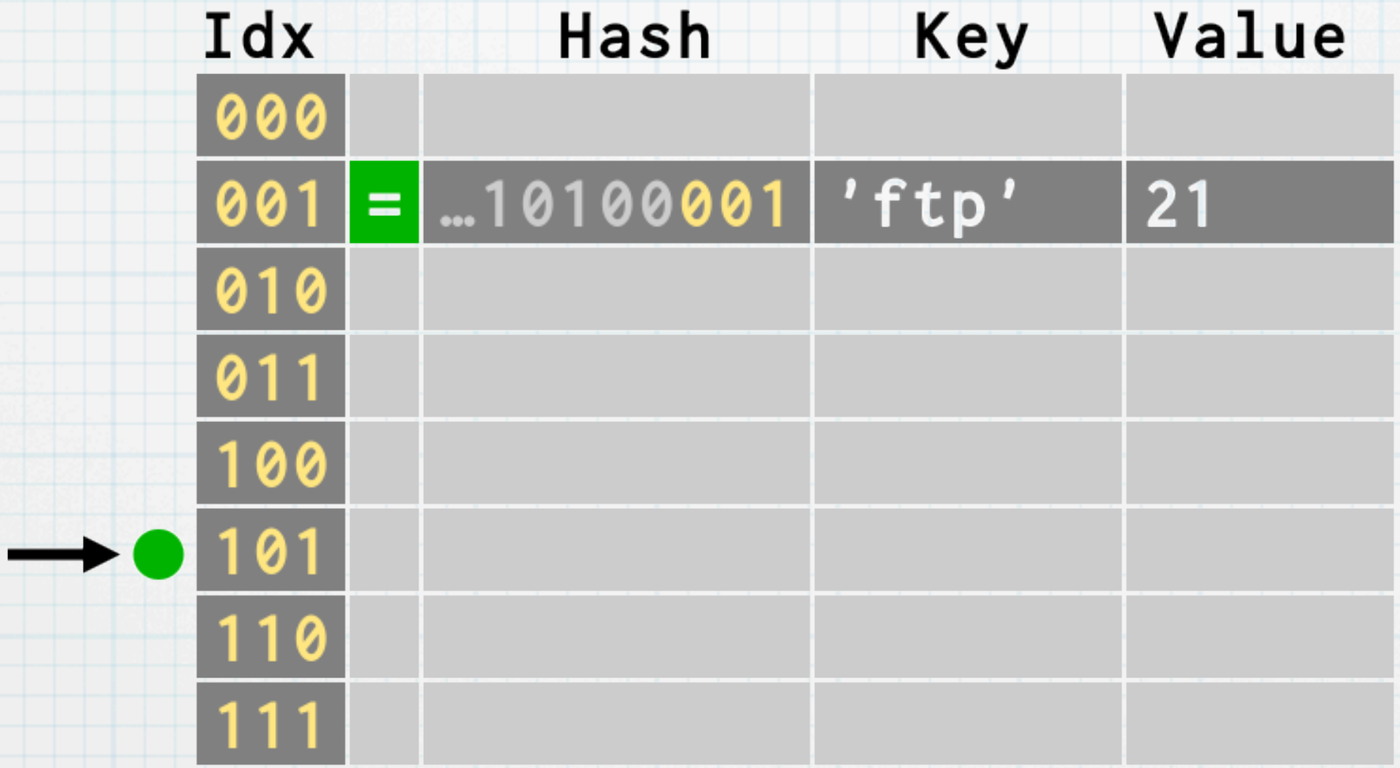
```
>>> d['ftp'] = 21
>>> bits(hash('ftp'))[-8:]
10100001
```



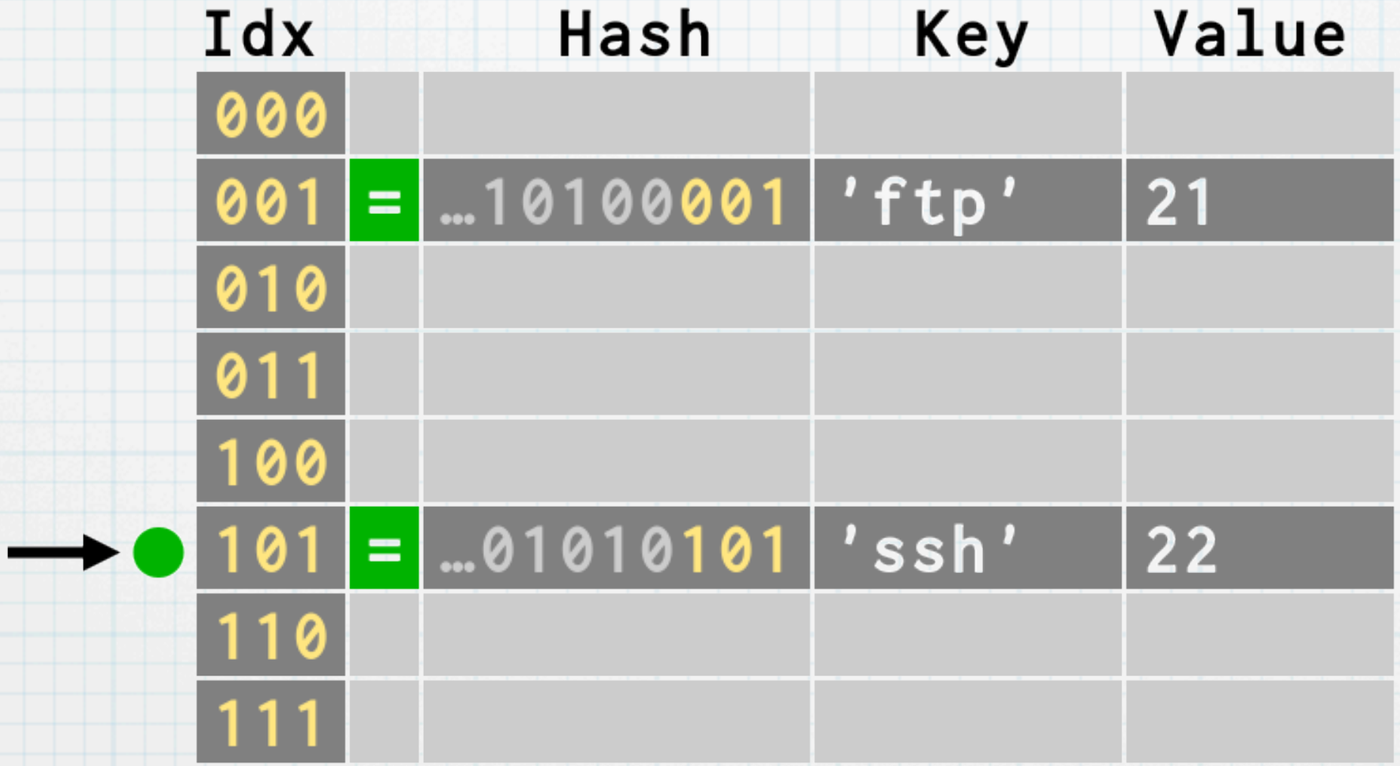
```
>>> d['ftp'] = 21
>>> bits(hash('ftp'))[-8:]
10100001
```



```
>>> d['ssh'] = 22
>>> bits(hash('ssh'))[-3:]
101
```



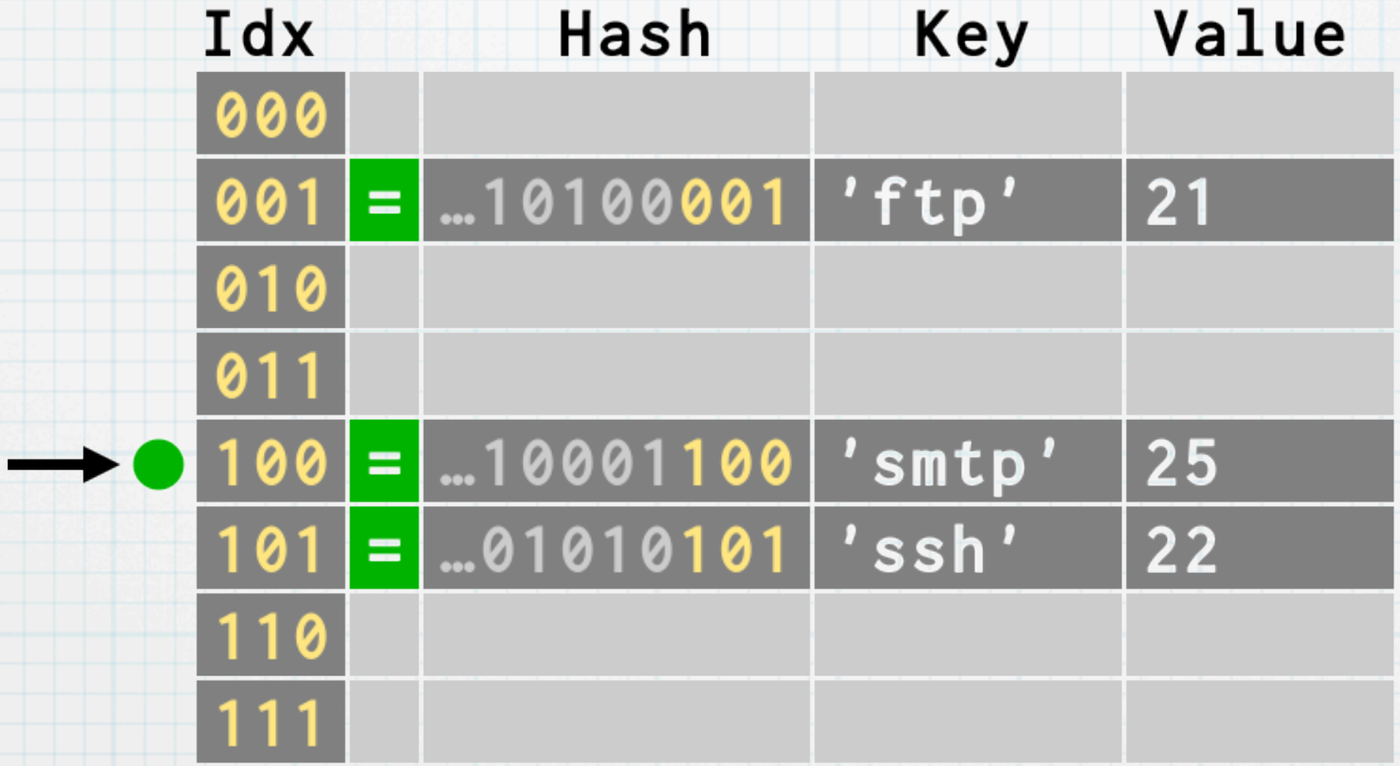
```
>>> d['ssh'] = 22
>>> bits(hash('ssh'))[-3:]
101
```



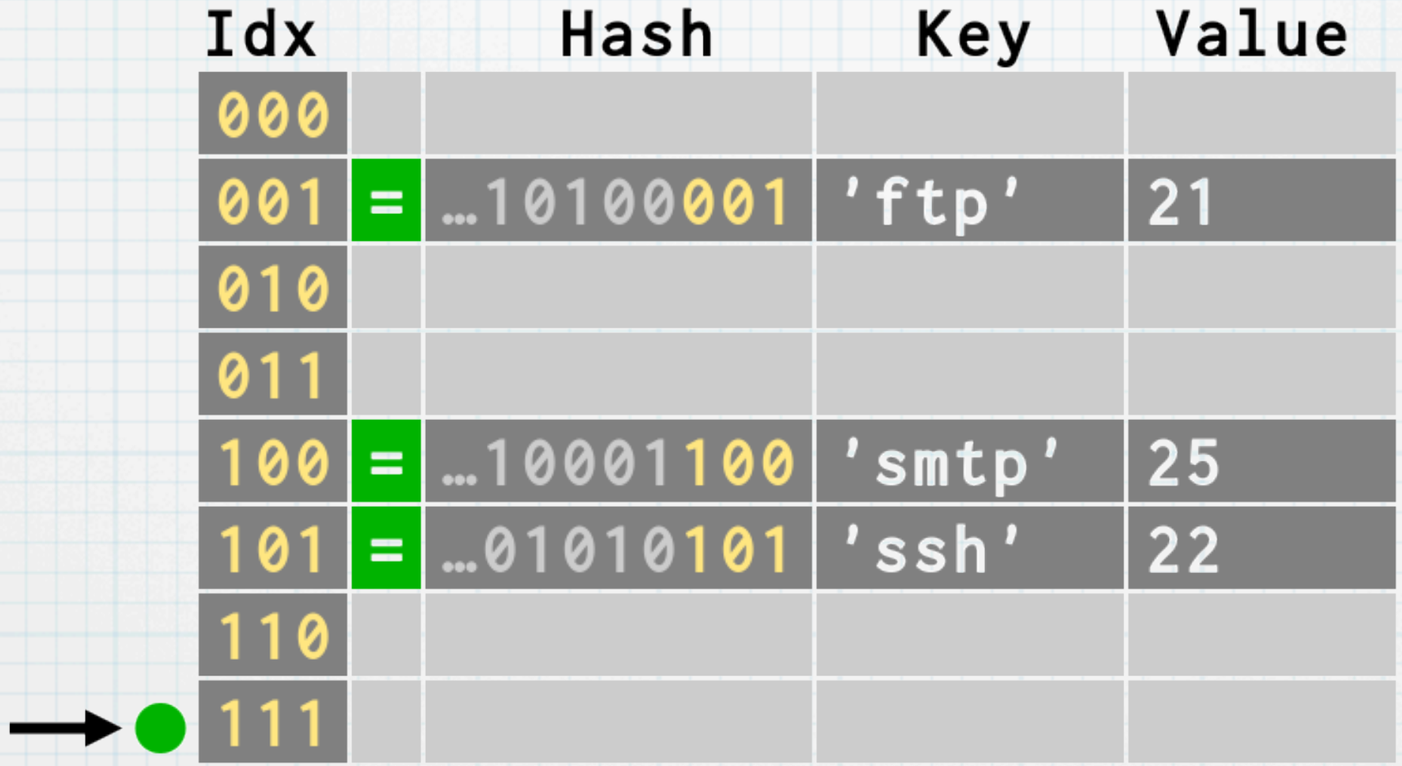
```
>>> d['smtp'] = 25
>>> bits(hash('smtp'))[-3:]
100
```

Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010				
011				
100				
101	=	…01010101	'ssh'	22
110				
111				

```
>>> d['smtp'] = 25
>>> bits(hash('smtp'))[-3:]
100
```



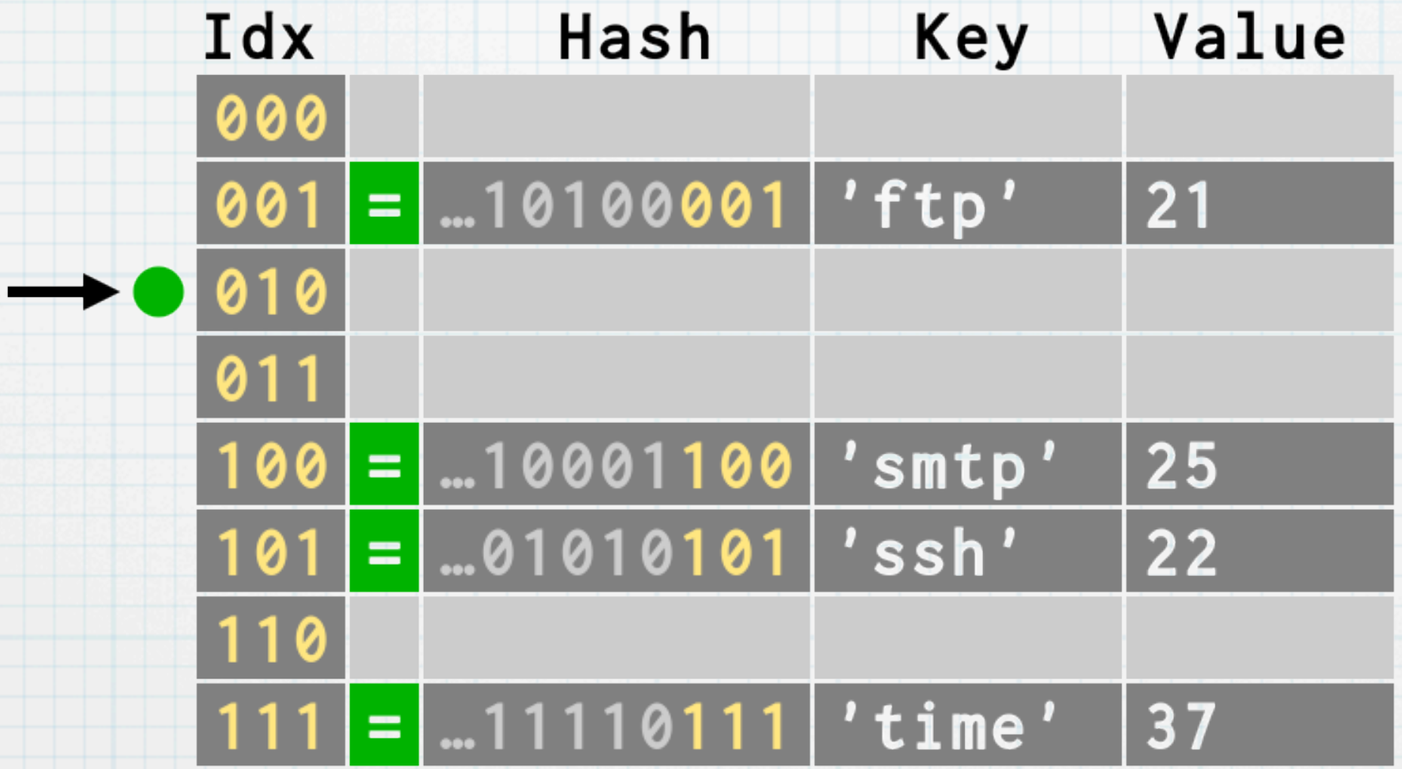
```
>>> d['time'] = 37
>>> bits(hash('time'))[-3:]
111
```



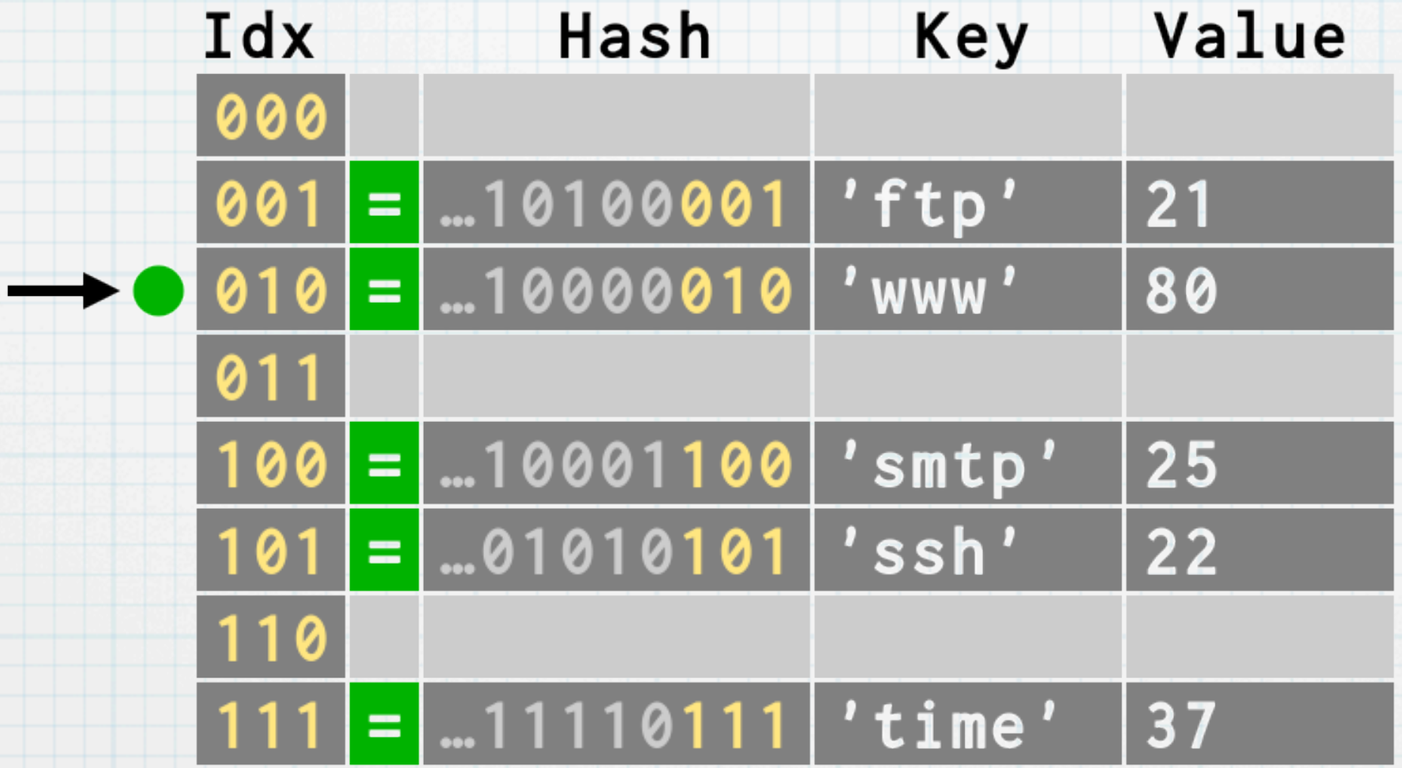
```
>>> d['time'] = 37
>>> bits(hash('time'))[-3:]
111
```

Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010				
011				
100	=	…10001100	'smtp'	25
101	=	…01010101	'ssh'	22
110				
111	=	…11110111	'time'	37

```
>>> d['www'] = 80
>>> bits(hash('www'))[-3:]
010
```



```
>>> d['www'] = 80
>>> bits(hash('www'))[-3:]
010
```



Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010	=	…10000010	'www'	80
011				
100	=	…10001100	'smtp'	25
101	=	…01010101	'ssh'	22
110				
111	=	…11110111	'time'	37

```
d = {'ftp': 21, 'ssh': 22,
     'smtp': 25, 'time': 37,
     'www': 80}
```

Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010	=	…10000010	'www'	80
011				
100	=	…10001100	'smtp'	25
101	=	…01010101	'ssh'	22
110				
111	=	…11110111	'time'	37

Поиск в словаре

- Вычислить хеш от ключа
- Обрезать старшие биты
- Взять значение из слота по индексу

```
>>> d['smtp']
25
>>> bits(hash('smtp'))[-3:]
100
```

Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010	=	…10000010	'www'	80
011				
100	=	…10001100	'smtp'	25
101	=	…01010101	'ssh'	22
110				
111	=	…11110111	'time'	37

Перебор всех элементов

- Словари возвращают свои ключи или значения в порядке отличном от порядка добавления

```
>>> print d
{'ftp': 21, 'www': 80, 'smtp':
25, 'ssh': 22, 'time': 37}
```

Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010	=	…10000010	'www'	80
011				
100	=	…10001100	'smtp'	25
101	=	…01010101	'ssh'	22
110				
111	=	…11110111	'time'	37

```
>>> d.keys()
['ftp', 'www', 'smtp', 'ssh',
'time']
```

Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010	=	…10000010	'www'	80
011				
100	=	…10001100	'smtp'	25
101	=	…01010101	'ssh'	22
110				
111	=	…11110111	'time'	37

```
>>> d.values()
[21, 80, 25, 22, 37]
```

Idx		Hash	Key	Value
000				
001	=	…10100001	'ftp'	21
010	=	…10000010	'www'	80
011				
100	=	…10001100	'smtp'	25
101	=	…01010101	'ssh'	22
110				
111	=	…11110111	'time'	37

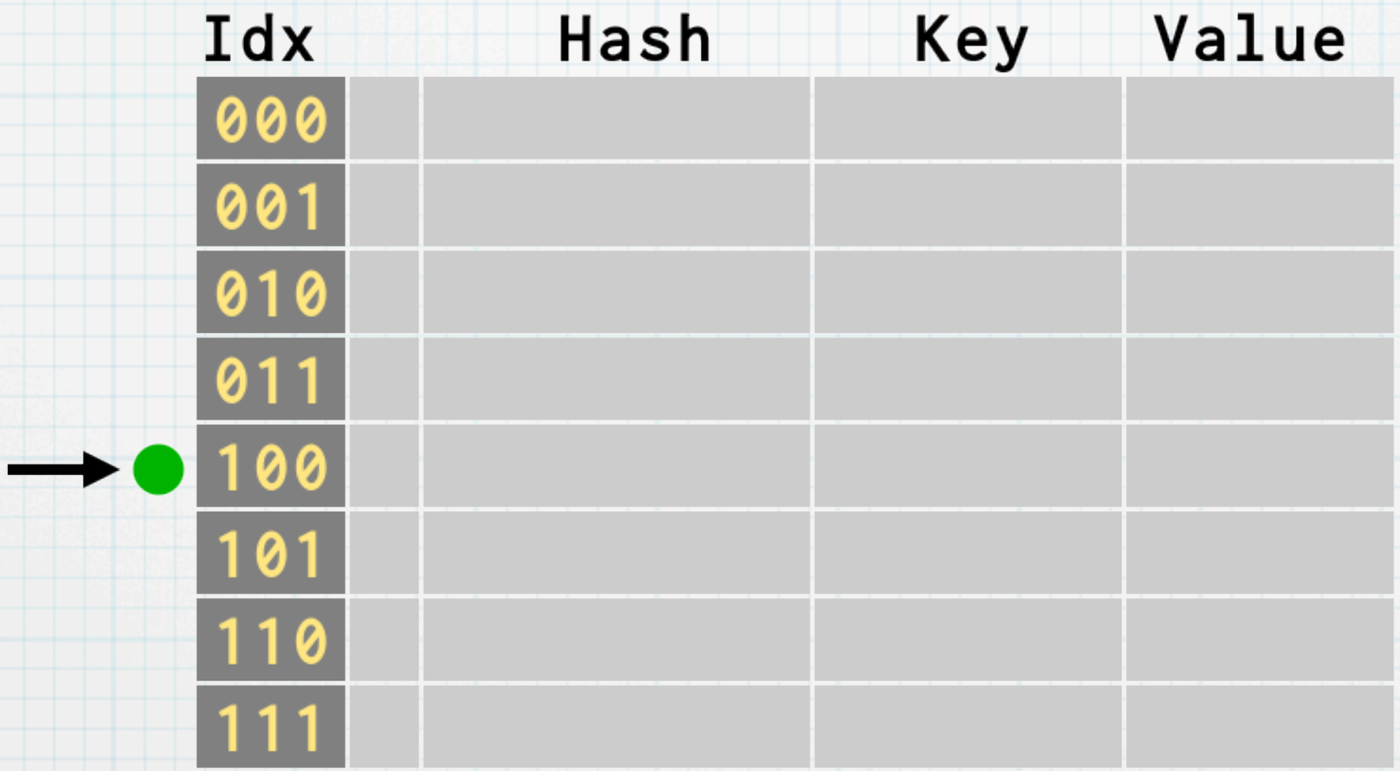
Коллизии

- Разные ключи пытаются доступиться по одинаковому индексу
- Находим первое свободное место

```
>>> d = {}
```

Idx		Hash	Key	Value
000				
001				
010				
011				
100				
101				
110				
111				

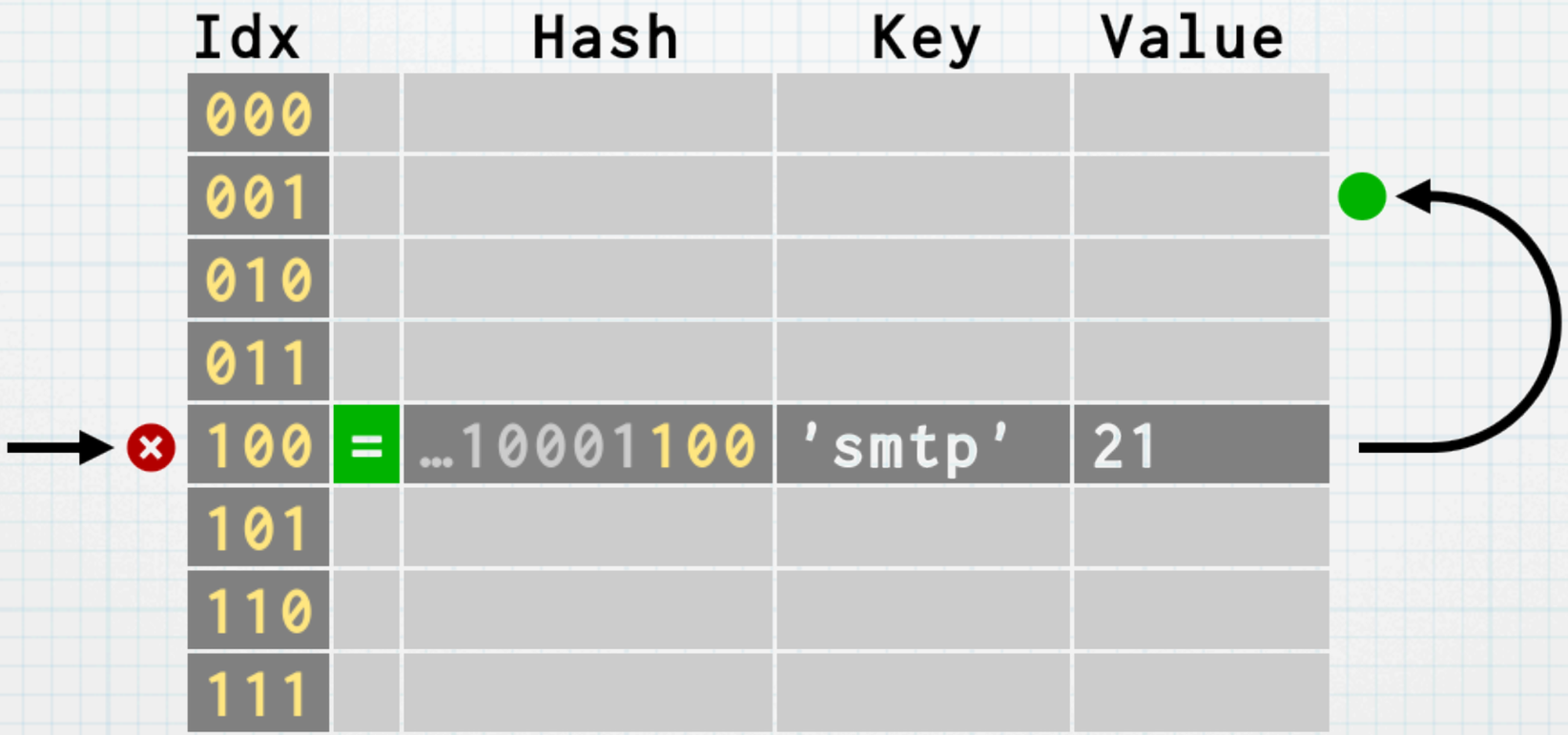
```
>>> d['smtp'] = 21
```



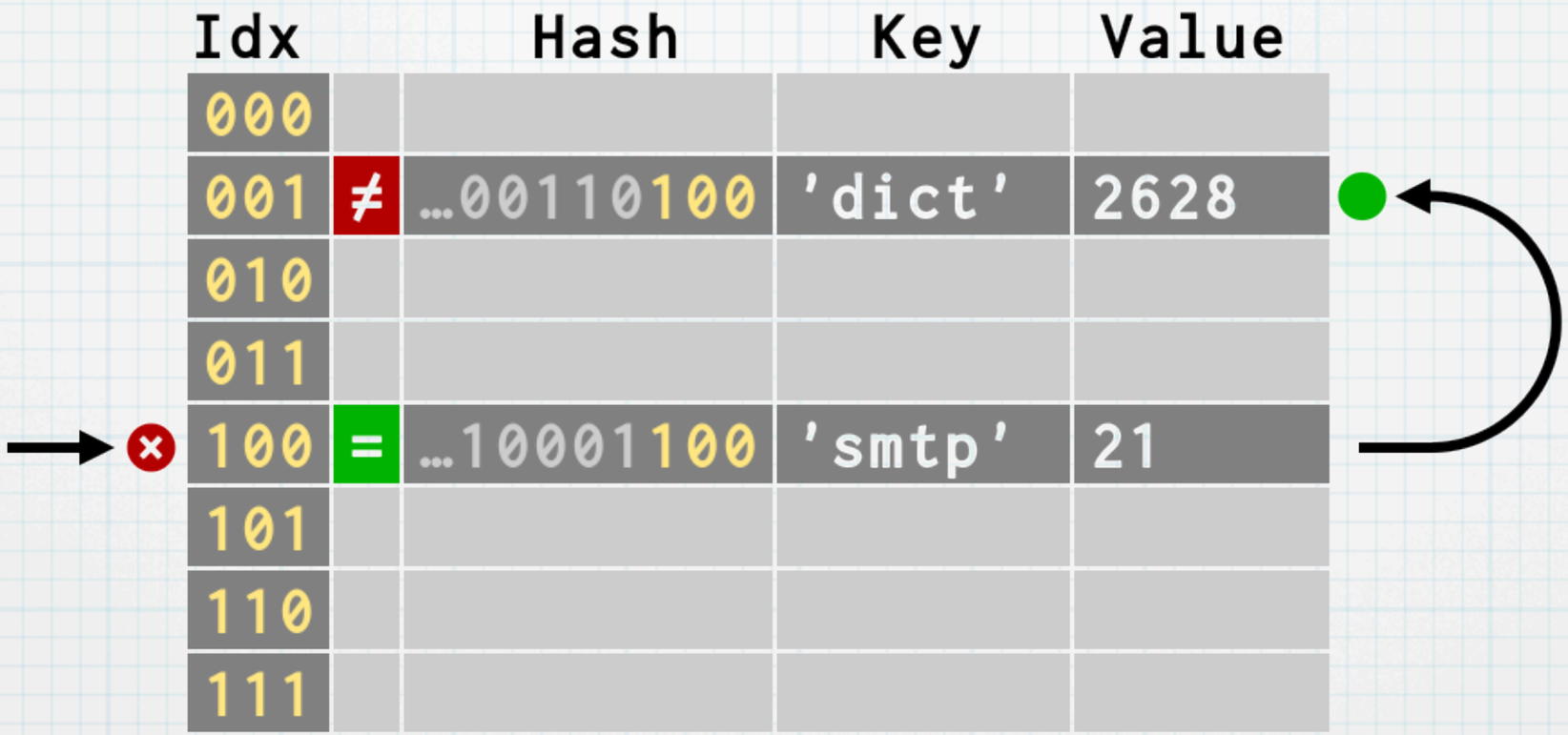
```
>>> d['smtp'] = 21
```

	Idx		Hash	Key	Value
	000				
	001				
	010				
	011				
→	100	=	…10001100	'smtp'	21
	101				
	110				
	111				

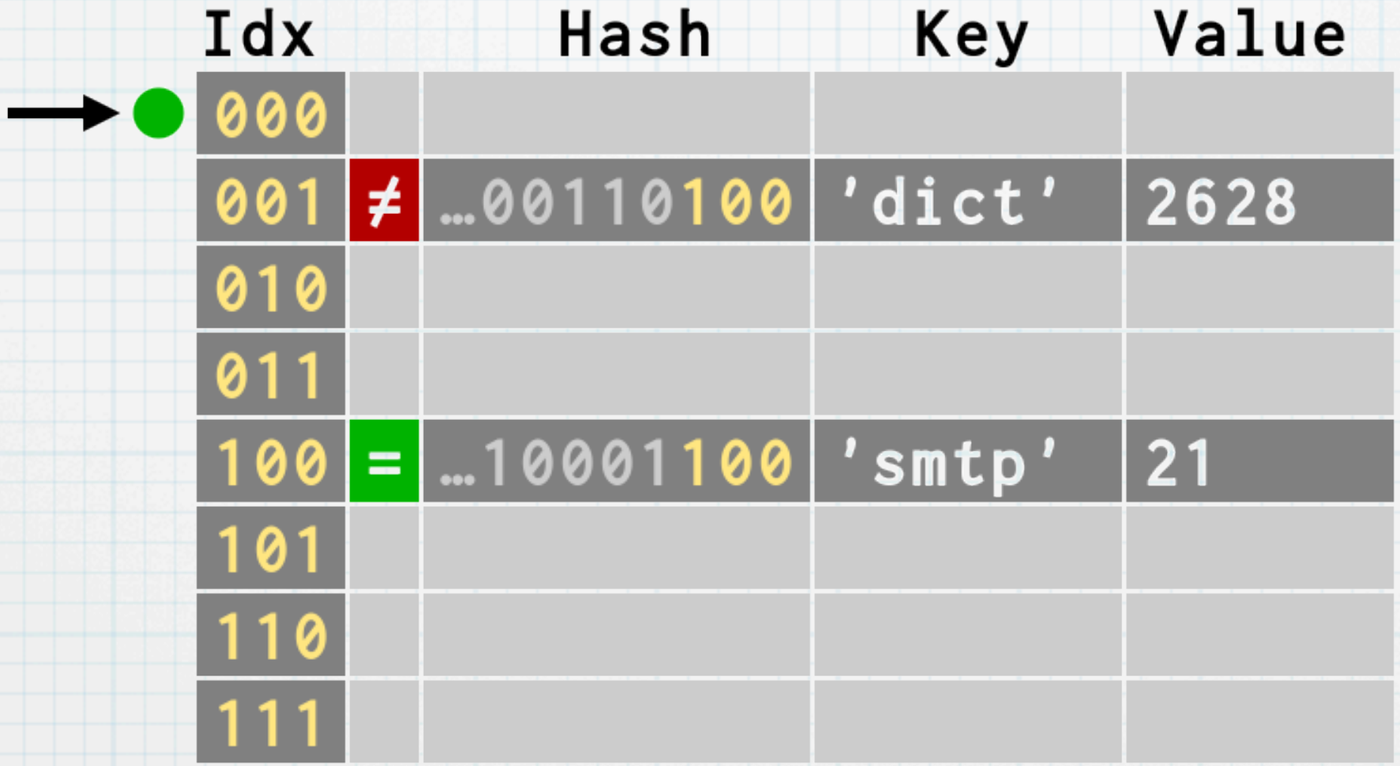
```
>>> d['dict'] = 2628
```


Idx
Hash
Key
Value
000
001
010
011
100
=
…10001100
'smtp'
21
101
110
111

```
>>> d['dict'] = 2628
```


Idx
Hash
Key
Value
000
001
≠
…00110100
'dict'
2628
010
011
100
=
…10001100
'smtp'
21
101
110
111

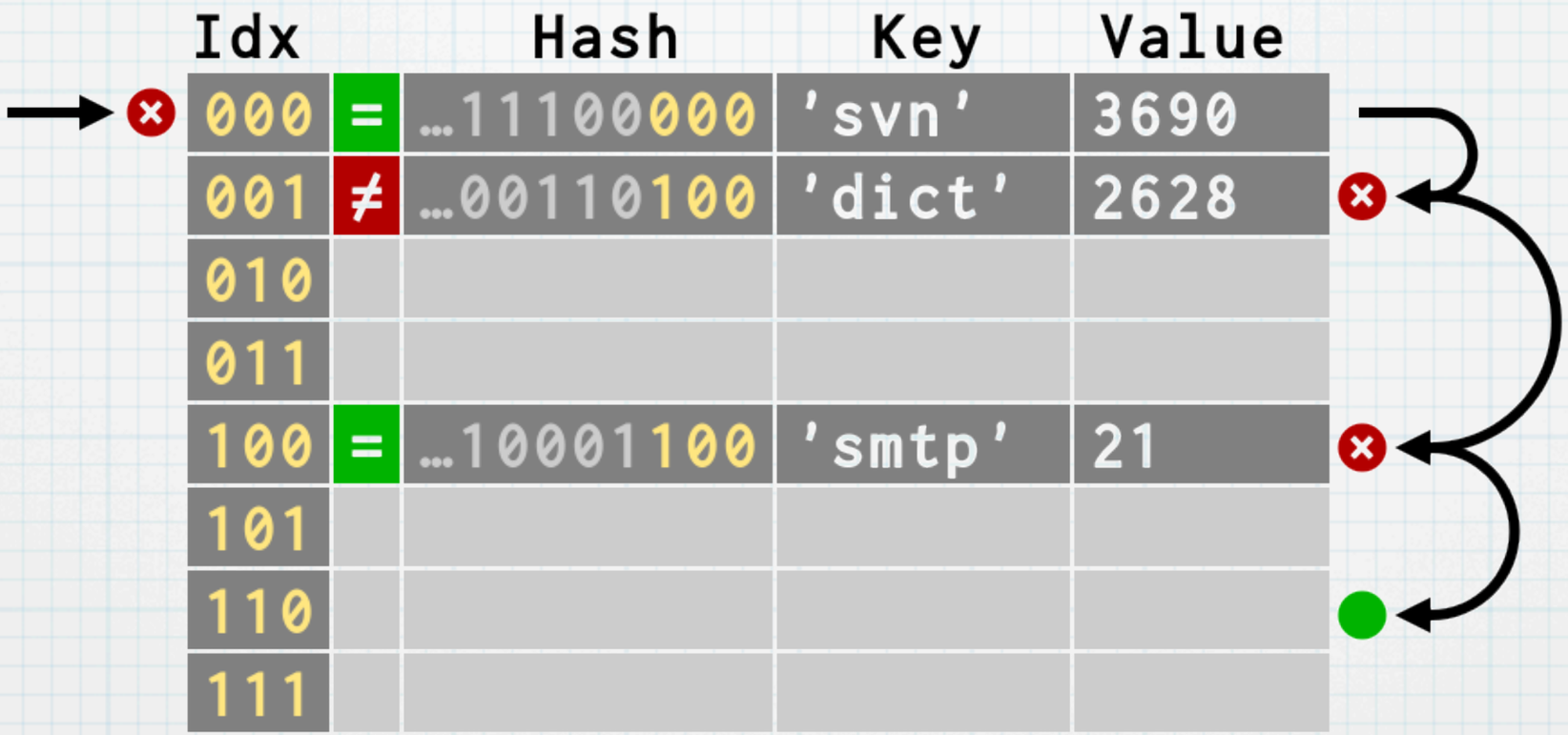
```
>>> d['svn'] = 3690
```


Idx
Hash
Key
Value
000
001
≠
…00110100
'dict'
2628
010
011
100
=
…10001100
'smtp'
21
101
110
111

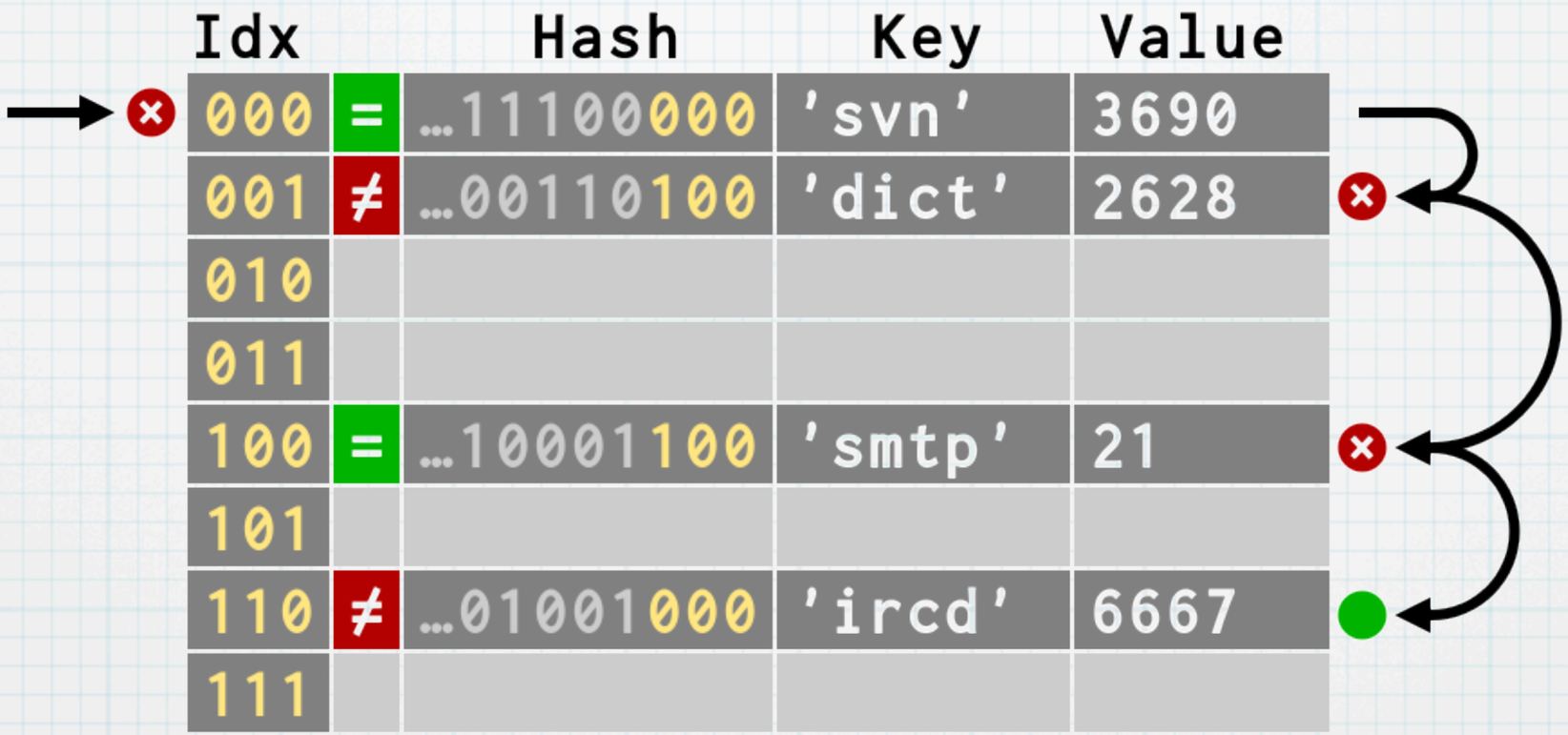
```
>>> d['svn'] = 3690
```

	Idx		Hash	Key	Value
→	000	=	…11100000	'svn'	3690
	001	≠	…00110100	'dict'	2628
	010				
	011				
	100	=	…10001100	'smtp'	21
	101				
	110				
	111				

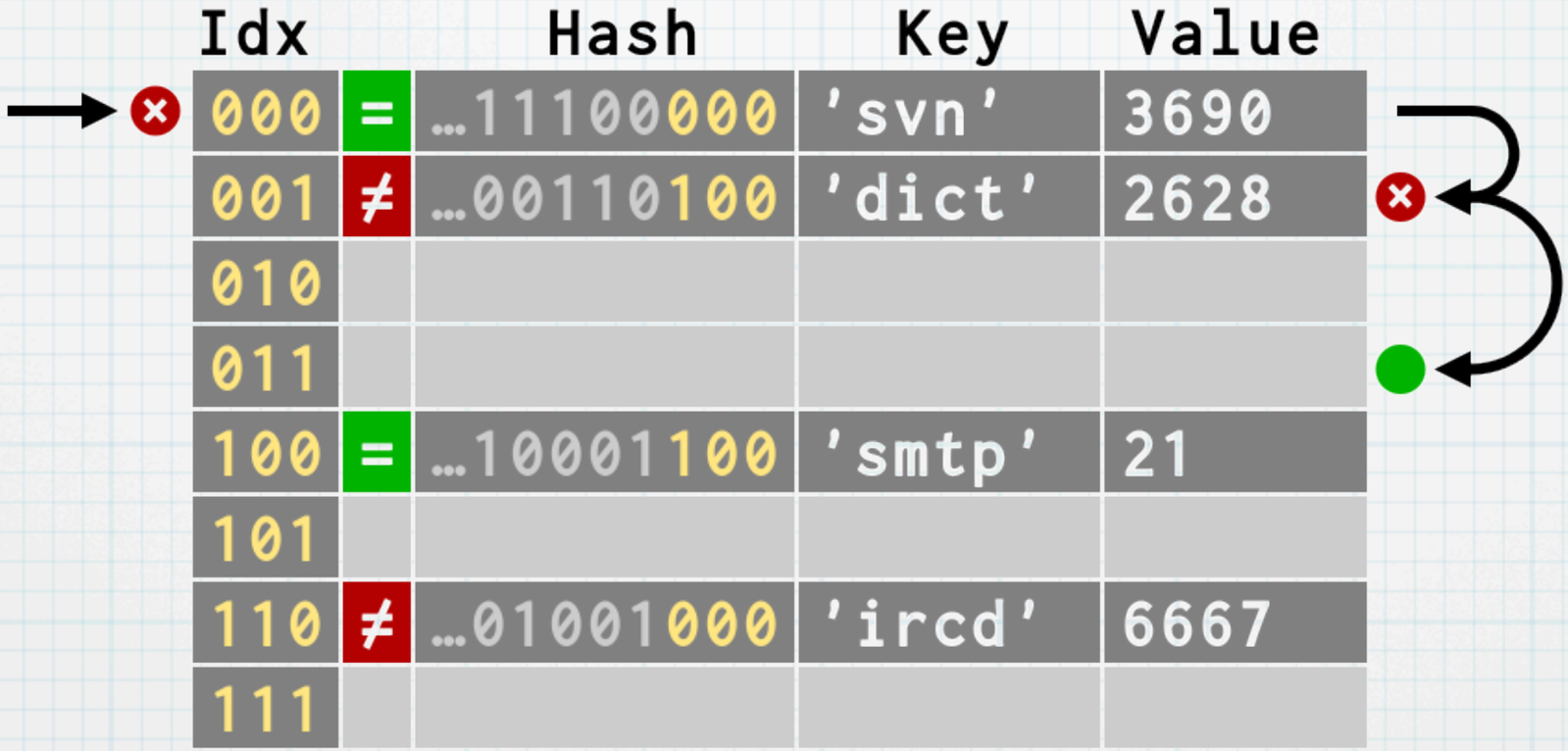
```
>>> d['ircd'] = 6667
```


Idx
Hash
Key
Value
000
=
…11100000
'svn'
3690
001
≠
…00110100
'dict'
2628
010
011
100
=
…10001100
'smtp'
21
101
110
111

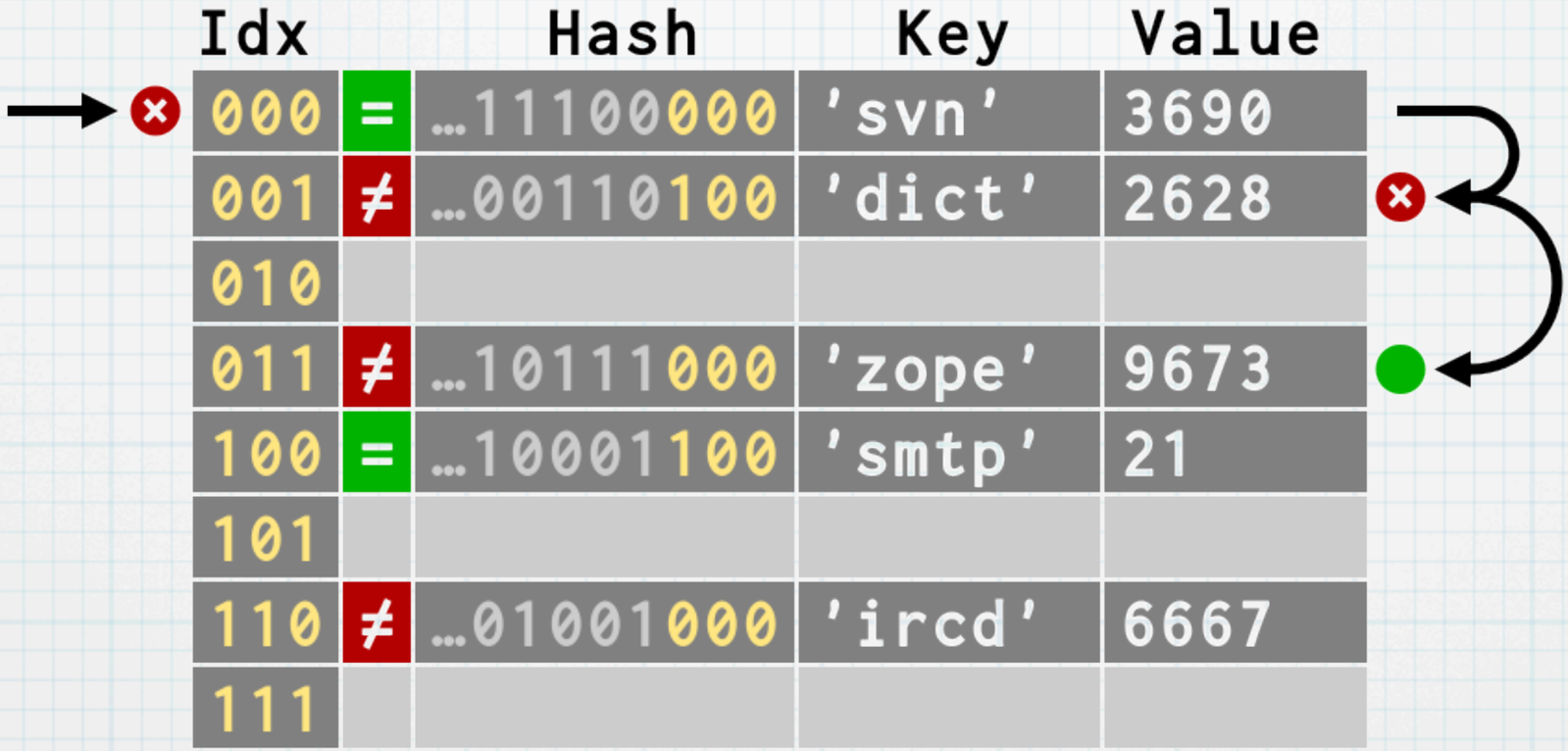
```
>>> d['ircd'] = 6667
```


Idx
Hash
Key
Value
000 = …11100000 'svn' 3690
001 ≠ …00110100 'dict' 2628
010
011
100 = …10001100 'smtp' 21
101
110 ≠ …01001000 'ircd' 6667
111

```
>>> d['zope'] = 9673
```


Idx
Hash
Key
Value
000 = …11100000 'svn' 3690
001 ≠ …00110100 'dict' 2628
010
011
100 = …10001100 'smtp' 21
101
110 ≠ …01001000 'ircd' 6667
111

```
>>> d['zope'] = 9673
# 2 из 5ти элементов на своих
ожидаемых местах
```



Коллизии и очередность

- Поскольку из за коллизий элементы могут находится не по своим "естественным" индексам порядок элементов зависит от порядка добавления

Поиск первой свободной ячейки

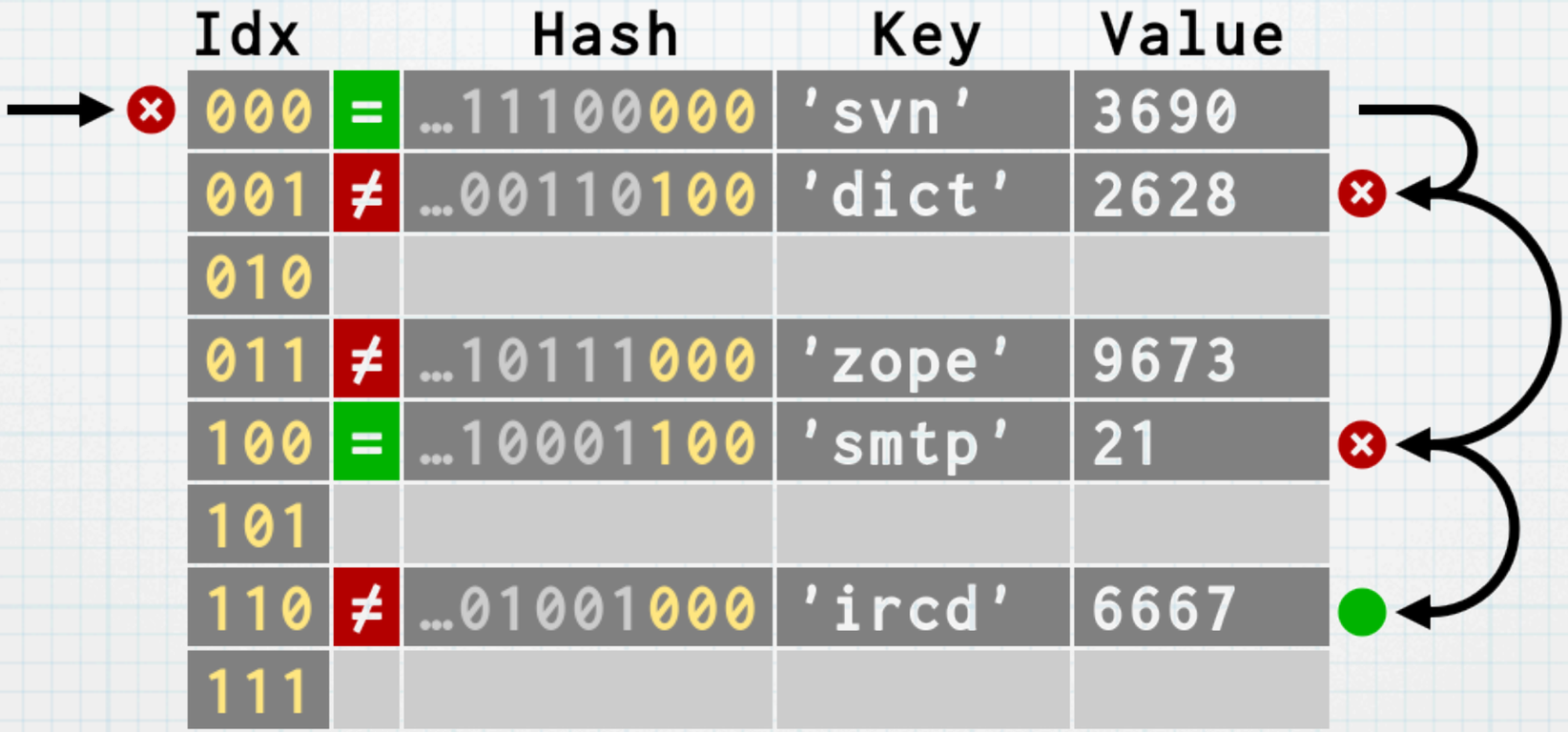
- Последовательный поиск плох для int ключей

```
pertrurb = hash
while (<слот занят>) {
   slot = (5*slot) + 1 + perturb;
   perturb >>= 5;
}
```

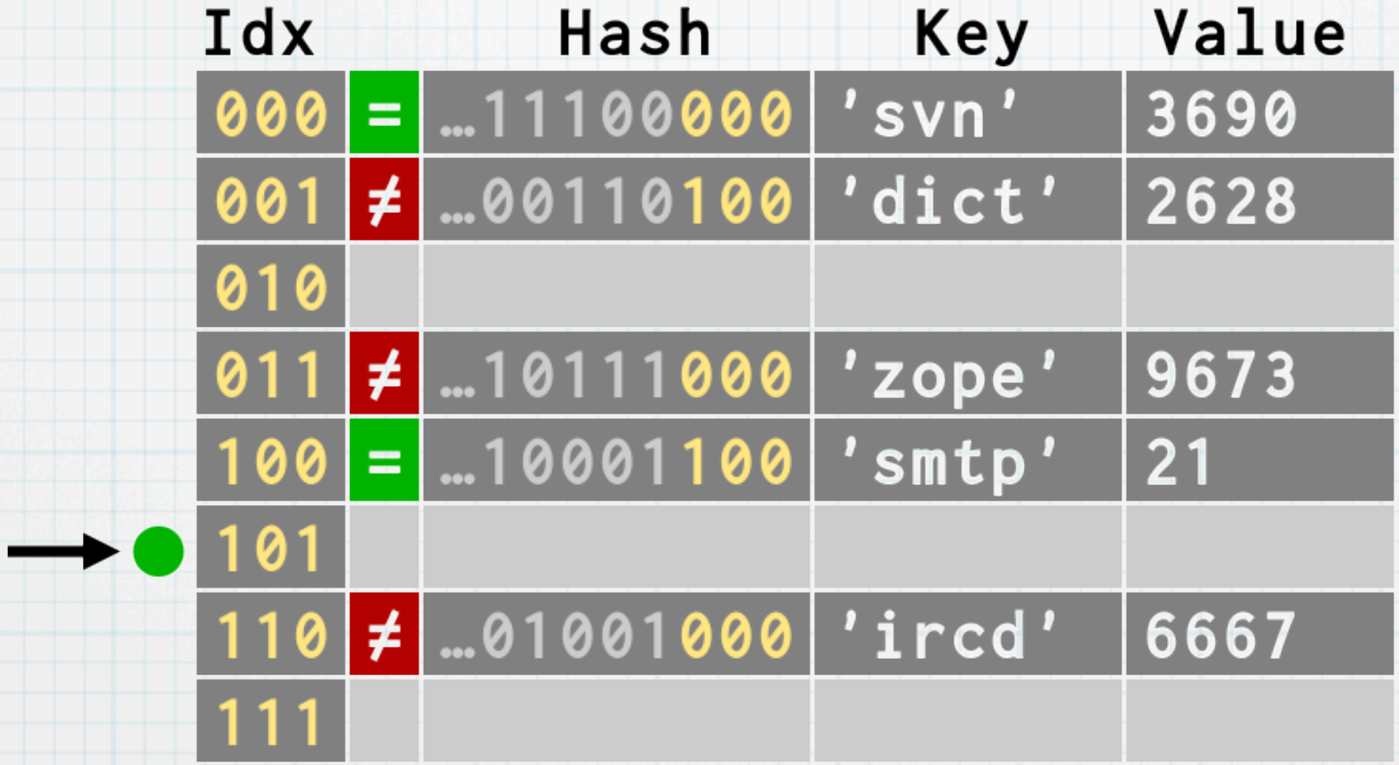
```
>>> d['svn']
3690
```

	Idx		Hash	Key	Value
→	000	=	…11100000	'svn'	3690
	001	≠	…00110100	'dict'	2628
	010				
	011	≠	…10111000	'zope'	9673
	100	=	…10001100	'smtp'	21
	101				
	110	≠	…01001000	'ircd'	6667
	111				

```
>>> d['ircd']
6667
```

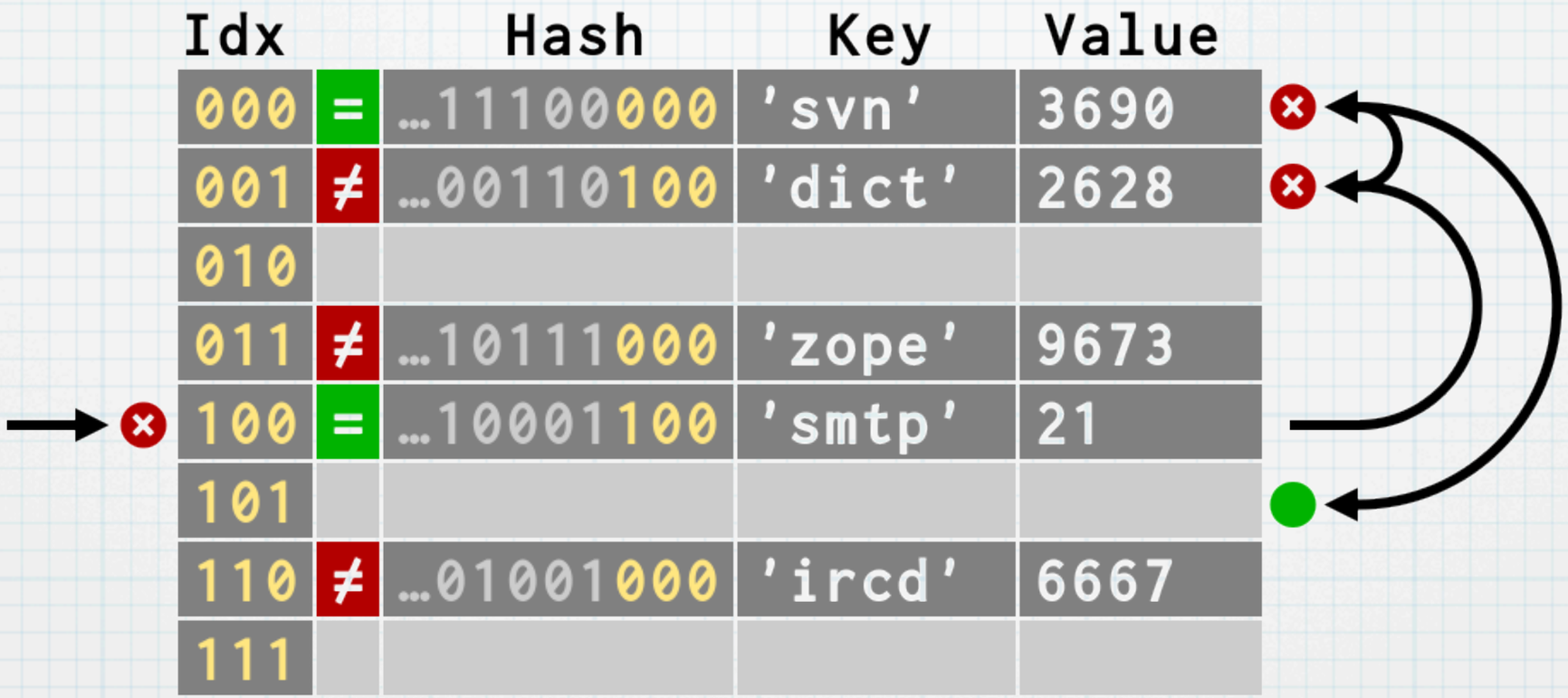

Idx
Hash
Key
Value
000 = …11100000 'svn' 3690
001 ≠ …00110100 'dict' 2628
010
011 ≠ …10111000 'zope' 9673
100 = …10001100 'smtp' 21
101
110 ≠ …01001000 'ircd' 6667
111

```
>>> d['nsca']
KeyError: 'nsca'
```



Idx		Hash	Key	Value
000	=	…11100000	'svn'	3690
001	≠	…00110100	'dict'	2628
010				
011	≠	…10111000	'zope'	9673
100	=	…10001100	'smtp'	21
101				
110	≠	…01001000	'ircd'	6667
111				

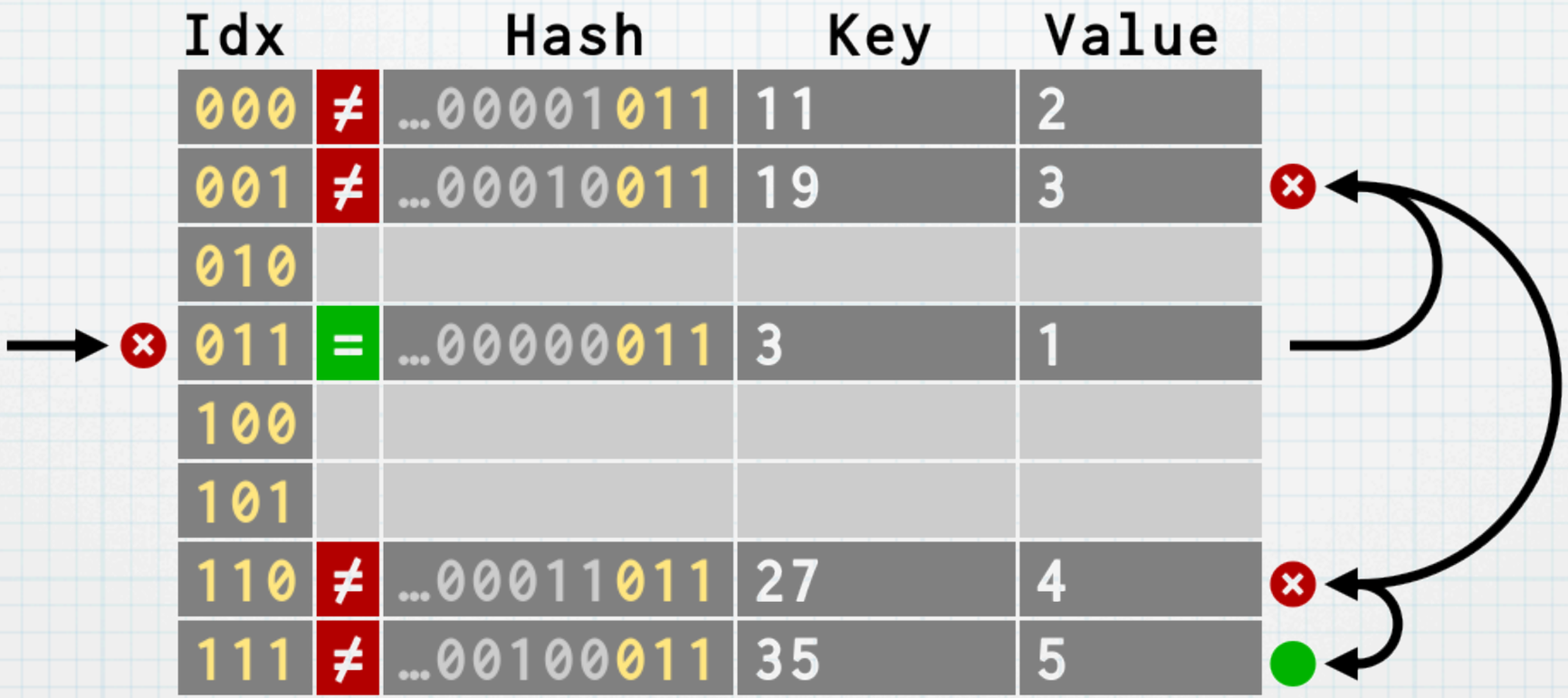
```
>>> d['netstat']
KeyError: 'netstat'
```



Не все поиски одинаковы

- Некоторые находят результат сразу
- Некоторым нужны несколько итераций

```
threes = {3: 1, 3+8: 2, 3+16: 3,
3+24: 4, 3+32: 5}
```



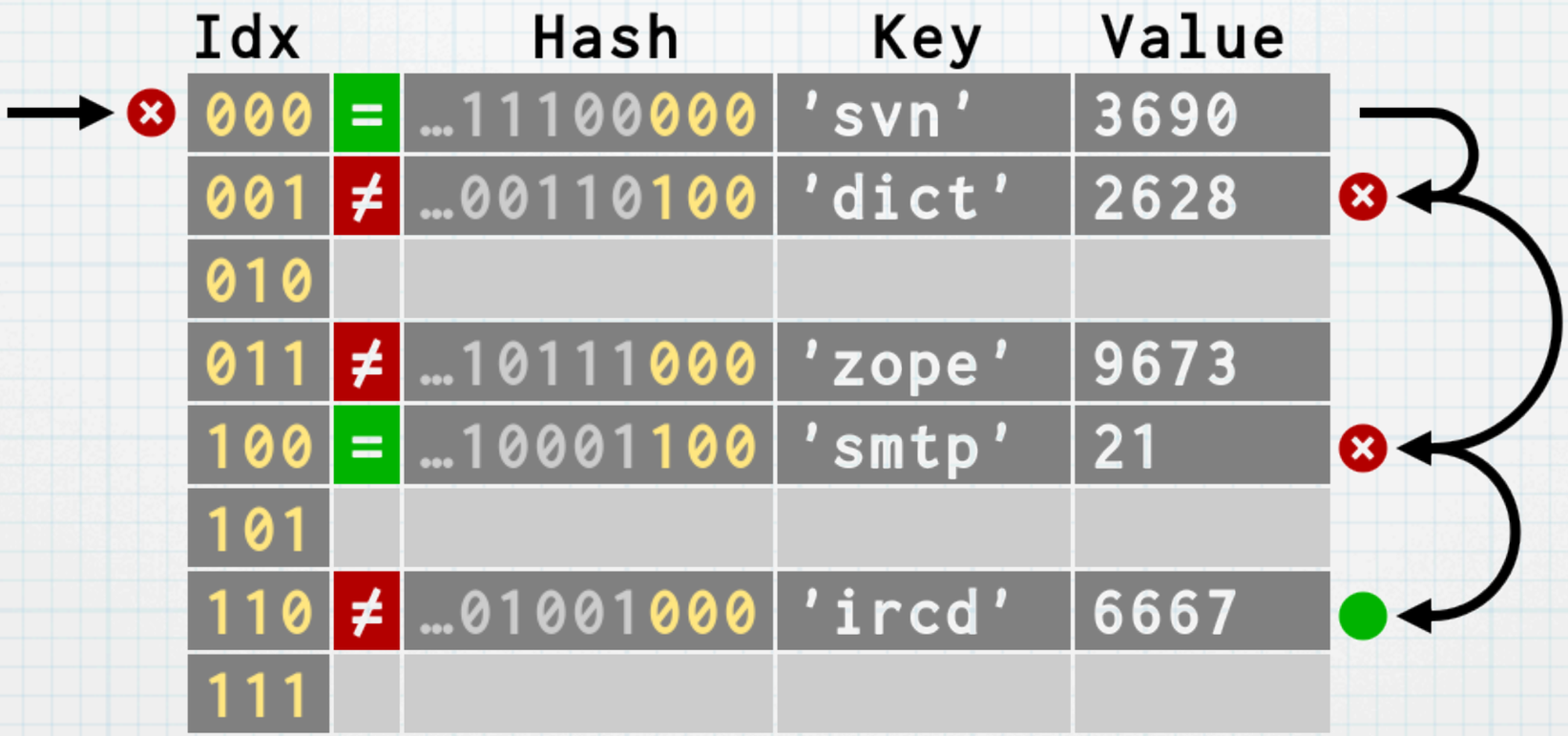
Удаление элементов

- Нелзя просто так взять, и пометить ячейку как пустую
- Необходимо вставить специальный "dummy" элемент

```
del d['smtp']
```

Idx		Hash	Key	Value
000	=	…11100000	'svn'	3690
001	≠	…00110100	'dict'	2628
010				
011	≠	…10111000	'zope'	9673
100	=	…10001100	'smtp'	21
101				
110	≠	…01001000	'ircd'	6667
111				

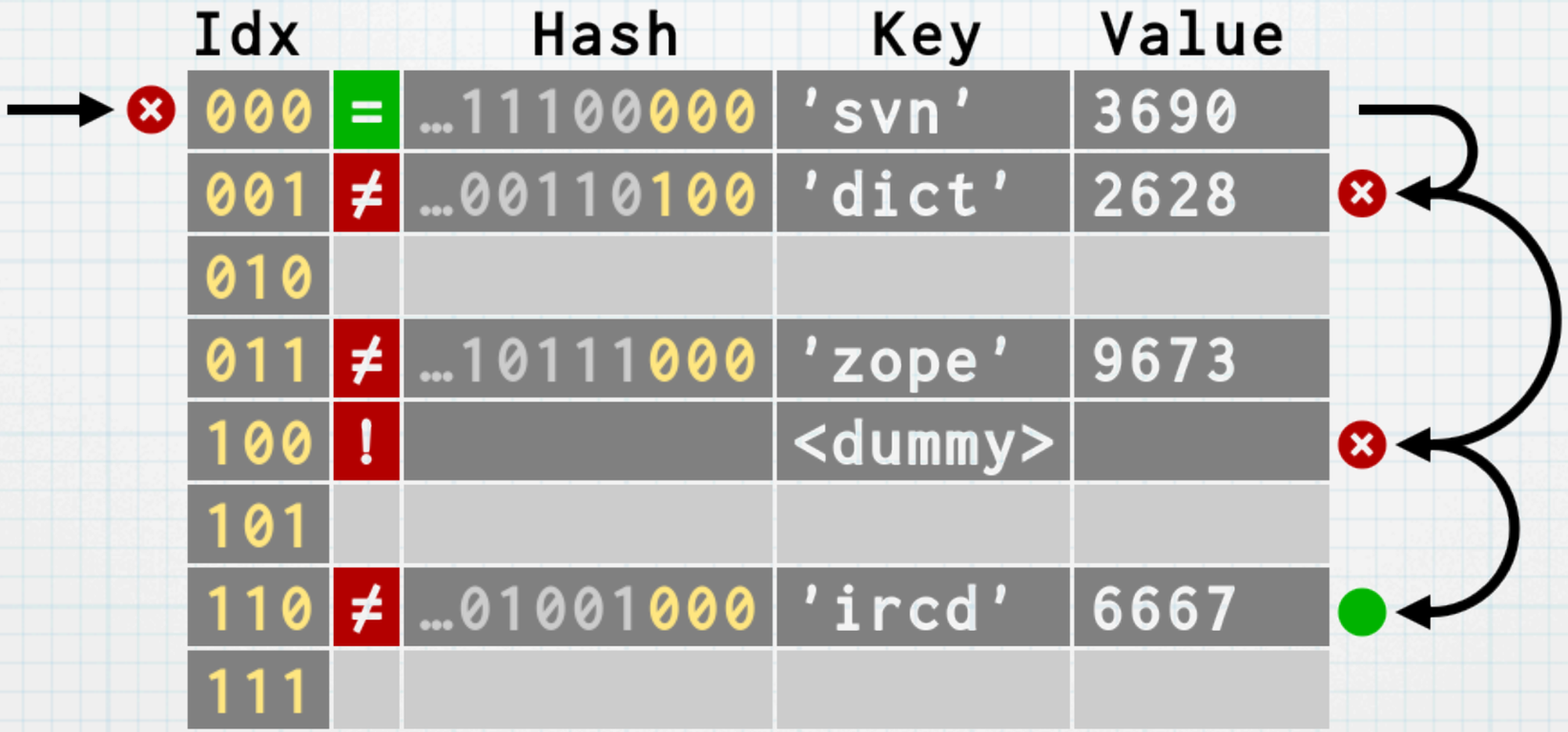
```
del d['smtp']
d['ircd'] ???
```



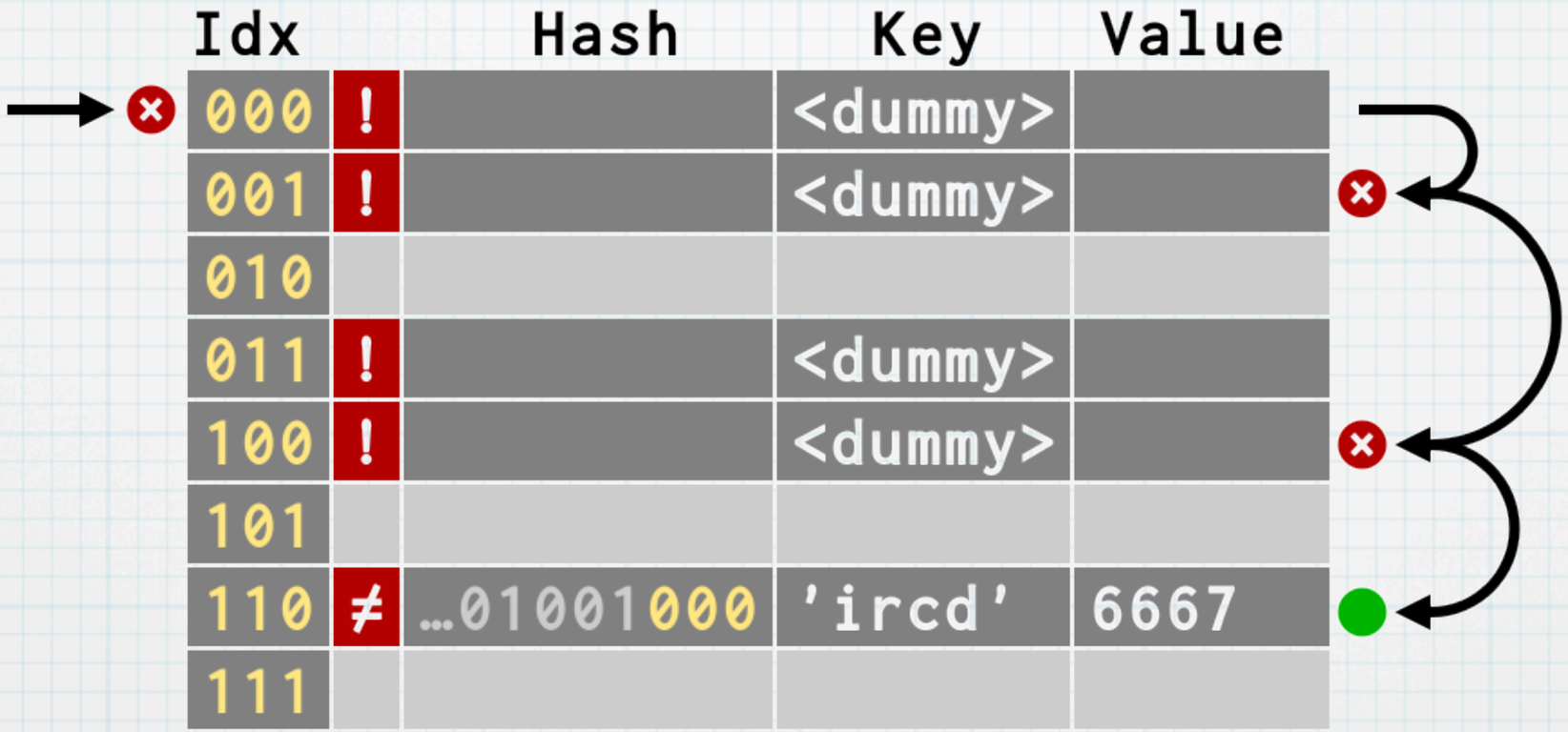
```
del d['smtp']
#Заменяем на "dummy" слот
#Может быть использован снова
```

Idx		Hash	Key	Value
000	=	…11100000	'svn'	3690
001	≠	…00110100	'dict'	2628
010				
011	≠	…10111000	'zope'	9673
100	!		<dummy>	
101				
110	≠	…01001000	'ircd'	6667
111				

```
del d['smtp']
#Заменяем на "dummy" слот
#Может быть использован снова
```



```
>>> del d['svn'], d['dict'],
d['zope']
>>> d['ircd']
```



Увеличение размера таблицы

- Таблица заполнена максимум на 2/3
- 2.7: < 50k size x 4
  > 50k size x 2
- 3.3: size x 2

```
>>> d = {}
```

Idx		Hash	Key	Value
000				
001				
010				
011				
100				
101				
110				
111				

```
d = dict.fromkeys(words[:5])
# 40% коллизий
# Заполнен на ⅔, resize
```

Idx		Hash	Key	Value
000	=	…11100000	'a'	None
001	=	…00100001	'aback'	None
010				
011	=	…00100011	'abaci'	None
100	≠	…00011111	'abase'	None
101				
110				
111	≠	…01110001	'abaft'	None

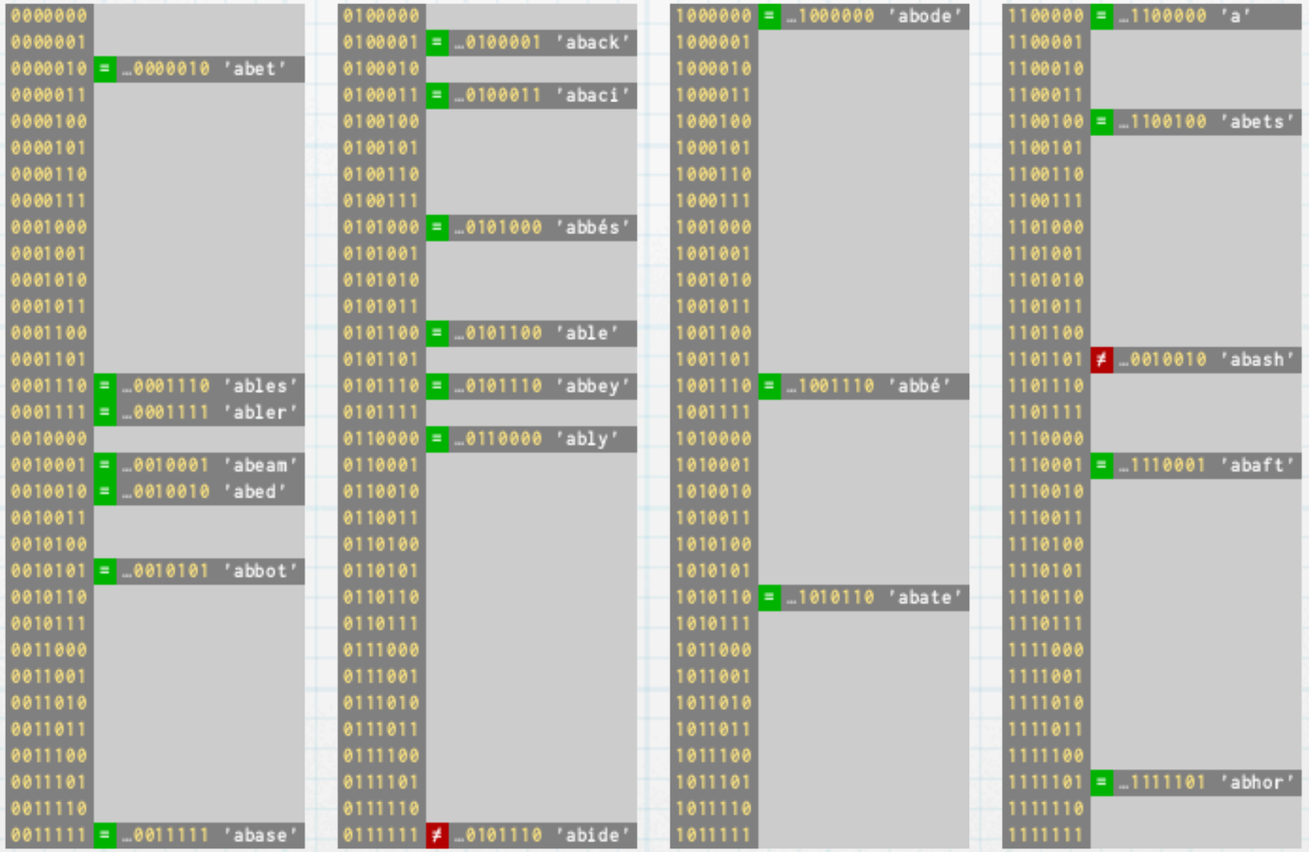
```
d['abash'] = None
# размер ×4 до 32
# 0% коллизий
```

```
00000 = …011000111100000 'a'     None
00001 = …110010100100001 'aback' None
00010
00011 = …110010100100011 'abaci' None
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
01110
01111
10000
10001 = …001101001110001 'abaft' None
10010 = …000100100010010 'abash' None
10011
10100
10101
10110
10111
11000
11001
11010
11011
11100
11101
11110
11111 = …000100100011111 'abase' None
```

```
d = dict.fromkeys(words[:21])
# 29% коллизий
# Заполнен на ⅔
```

```
00000 = …011000111100000 'a'     None
00001 = …110010100100001 'aback' None
00010 = …110011100000010 'abet'  None
00011 = …110010100100011 'abaci' None
00100 = …000001011100100 'abets' None
00101
00110 ≠ …011101110110000 'ably'  None
00111 ≠ …101111110010001 'abeam' None
01000 = …001110010101000 'abbés' None
01001 ≠ …110110010001110 'ables' None
01010
01011
01100 = …011101110101100 'able'  None
01101 ≠ …110011100010010 'abed'  None
01110 = …111001000101110 'abbey' None
01111 = …110110010001111 'abler' None
10000 ≠ …001110011001110 'abbé'  None
10001 = …001101001110001 'abaft' None
10010 = …000100100010010 'abash' None
10011
10100
10101 = …111111110010101 'abbot' None
10110 = …100001011010110 'abate' None
10111
11000
11001
11010
11011
11100
11101 = …001111111111101 'abhor' None
11110 ≠ …000101010101110 'abide' None
11111 = …000100100011111 'abase' None
```

```
d['abode'] = None
# размер ×4 до 128
# 9% коллизий
```



```
d = dict.fromkeys(words[:85])
# 33% коллизий
# Заполнен на ⅔
```

```
0000000
0000001
0000010 = …0000010 'abet'
0000011 ≠ …0101011 'acne'
0000100 = …0000100 'adobe'
0000101 ≠ …1010110 'acorn'
0000110 ≠ …1010101 'adult'
0000111
0001000
0001001 ≠ …0101100 'adz'
0001010 = …0001010 'acted'
0001011
0001100 ≠ …0100001 'ad's'
0001101
0001110 = …0001110 'ables'
0001111 = …0001111 'abler'
0010000
0010001 = …0010001 'abeam'
0010010 = …0010010 'abed'
0010011 = …0010011 'abort'
0010100 = …0010100 'aegis'
0010101 = …0010101 'abbot'
0010110 ≠ …0001110 'adzes'
0010111
0011000 = …0011000 'aces'
0011001
0011010
0011011 = …0011011 'acid'
0011100 = …0011100 'acrid'
0011101 = …0011101 'acing'
0011110
0011111 = …0011111 'abase'
```

```
0100000
0100001 = …0100001 'aback'
0100010
0100011 = …0100011 'abaci'
0100100 = …0100100 'adds'
0100101 ≠ …1000111 'act's'
0100110 ≠ …0101011 'acts'
0100111 ≠ …0101110 'actor'
0101000 = …0101000 'abbés'
0101001
0101010
0101011 ≠ …0100001 'aches'
0101100 = …0101100 'able'
0101101 = …0101101 'ado's'
0101110 = …0101110 'abbey'
0101111 = …0101111 'achoo'
0110000 = …0110000 'ably'
0110001 ≠ …0101111 'acres'
0110010 = …0110010 'add'
0110011
0110100
0110101 = …0110101 'adapt'
0110110 = …0110110 'above'
0110111 = …0110111 'acre'
0111000
0111001 = …0111001 'ado'
0111010
0111011
0111100
0111101 = …0111101 'aeons'
0111110 ≠ …1000011 'adieu'
0111111 ≠ …0101110 'abide'
```

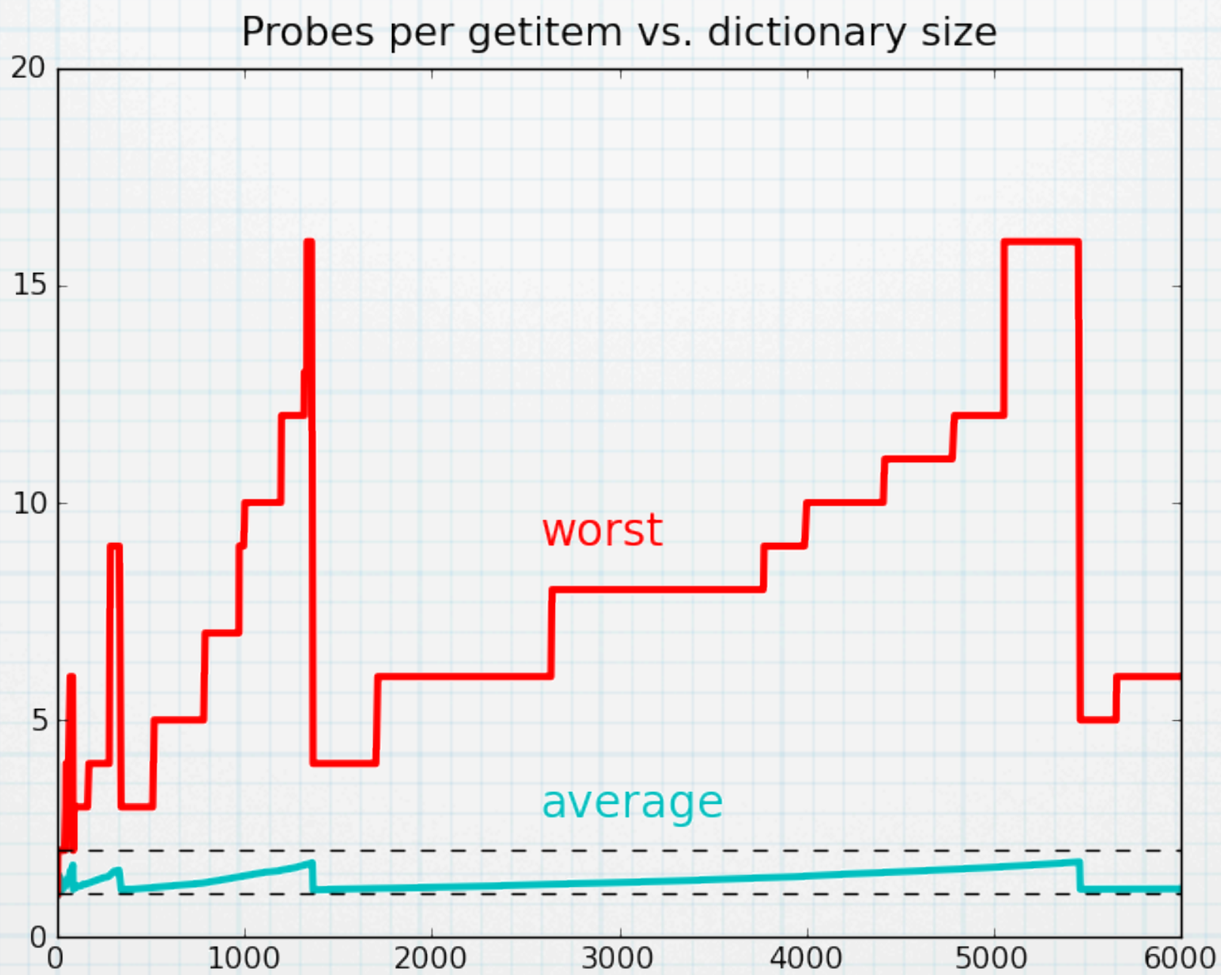
```
1000000 = …1000000 'abode'
1000001 ≠ …0000101 'ad'
1000010
1000011 = …1000011 'added'
1000100
1000101 ≠ …0110110 'ached'
1000110
1000111 ≠ …0100001 'abyss'
1001000 = …1001000 'adz's'
1001001
1001010
1001011 ≠ …1110111 'act'
1001100 = …1001100 'admen'
1001101
1001110 = …1001110 'abbé'
1001111 ≠ …0101100 'adze'
1010000 = …1010000 'acmes'
1010001 = …1010001 'adage'
1010010 = …1010010 'abut'
1010011 = …1010011 'abuse'
1010100
1010101 = …1010101 'adder'
1010110 = …1010110 'abate'
1010111
1011000
1011001 = …1011001 'ache'
1011010 ≠ …0011101 'addle'
1011011 ≠ …0001111 'aced'
1011100
1011101 = …1011101 'aeon'
1011110
1011111 ≠ …0100101 'ads'
```

```
1100000 = …1100000 'a'
1100001 = …1100001 'acute'
1100010 = …1100010 'about'
1100011 = …1100011 'adiós'
1100100 = …1100100 'abets'
1100101 ≠ …1100110 'acme'
1100110 = …1100110 'ace'
1100111 ≠ …1010001 'adept'
1101000 ≠ …1000101 'achy'
1101001
1101010
1101011 = …1101011 'acids'
1101100
1101101 ≠ …0010010 'abash'
1101110
1101111 ≠ …1101011 'adopt'
1110000 = …1110000 'adman'
1110001 = …1110001 'abaft'
1110010 = …1110010 'admit'
1110011 ≠ …0101110 'ace's'
1110100 = …1110100 'abuts'
1110101
1110110
1110111 = …1110111 'abuzz'
1111000
1111001 ≠ …1110100 'adore'
1111010
1111011
1111100
1111101 = …1111101 'abhor'
1111110
1111111 = …1111111 'adorn'
```

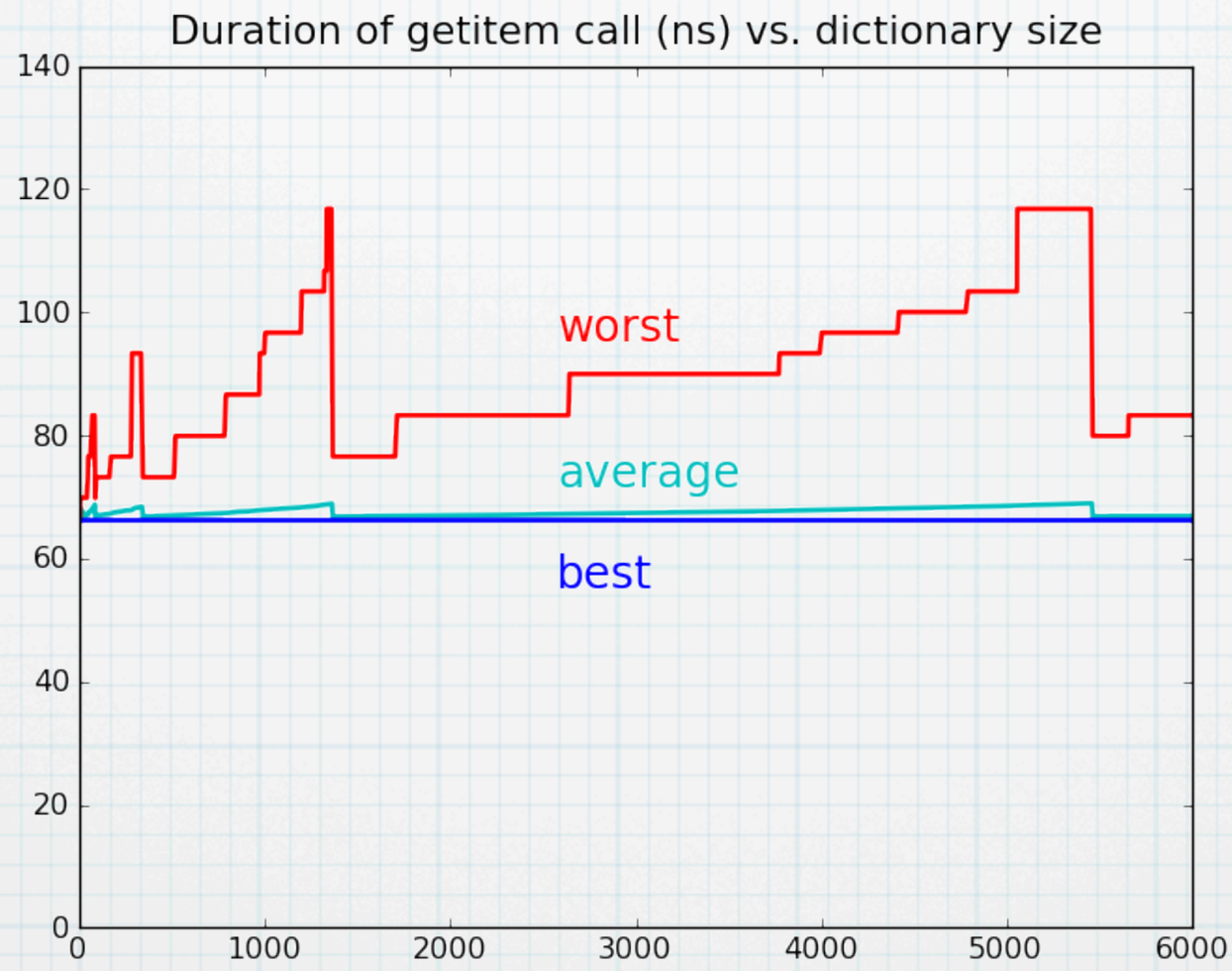
Время доступа к элементам

- Растет по мере заполнения словаря
- Затем резко умешьшается после изменения размера
- Среднее время доступа OK

Поиски vs размер



Время vs размер



Удаление элементов

- Не уменьшает размер таблицы
- Таблица может уменьшиться только при добавлении элементов

Порядок элементов

- Во время изменения размера порядок элементов может полностью поменяться
- Добавление элементов во время итерации запрещено

RuntimeError: dictionary changed size during iteration

Свой __hash__()

- Хорошо перемешать биты
- Равные хеши для равных элементов
- __eq__() должен быть
- Быстро вычисляется

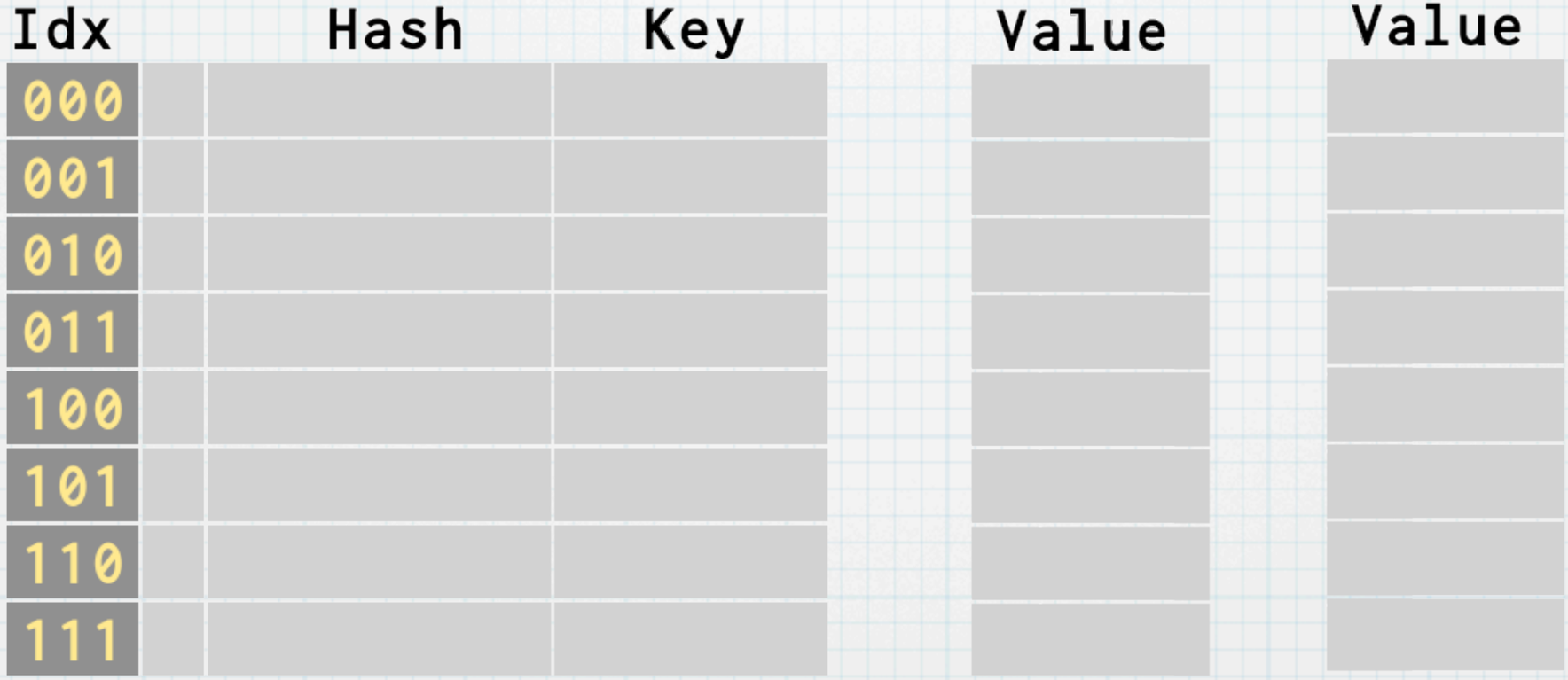
Пример __hash__()

```
class Point(object):
  def __init__(self, x, y):
    self.x, self.y = x, y

  def __eq__(self, p):
    return self.x==p.x and self.y==p.y

  def __hash__(self):
    return hash(self.x) ^ hash(self.y)
```

oCERT #2011-003

- Хэш для str, bytes и datetime смешивается с "солью" уникальной для каждого процесса Python
- pre 3.3: -R option
  3.3: by default

Python 3: Split-table словари. Общая таблица с ключами для разных таблиц со значениями



Спасибо

- http://blip.tv/pycon-us-videos-2009-2010-2011/pycon-2010-the-mighty-dictionary-55-3352147
- Python source code