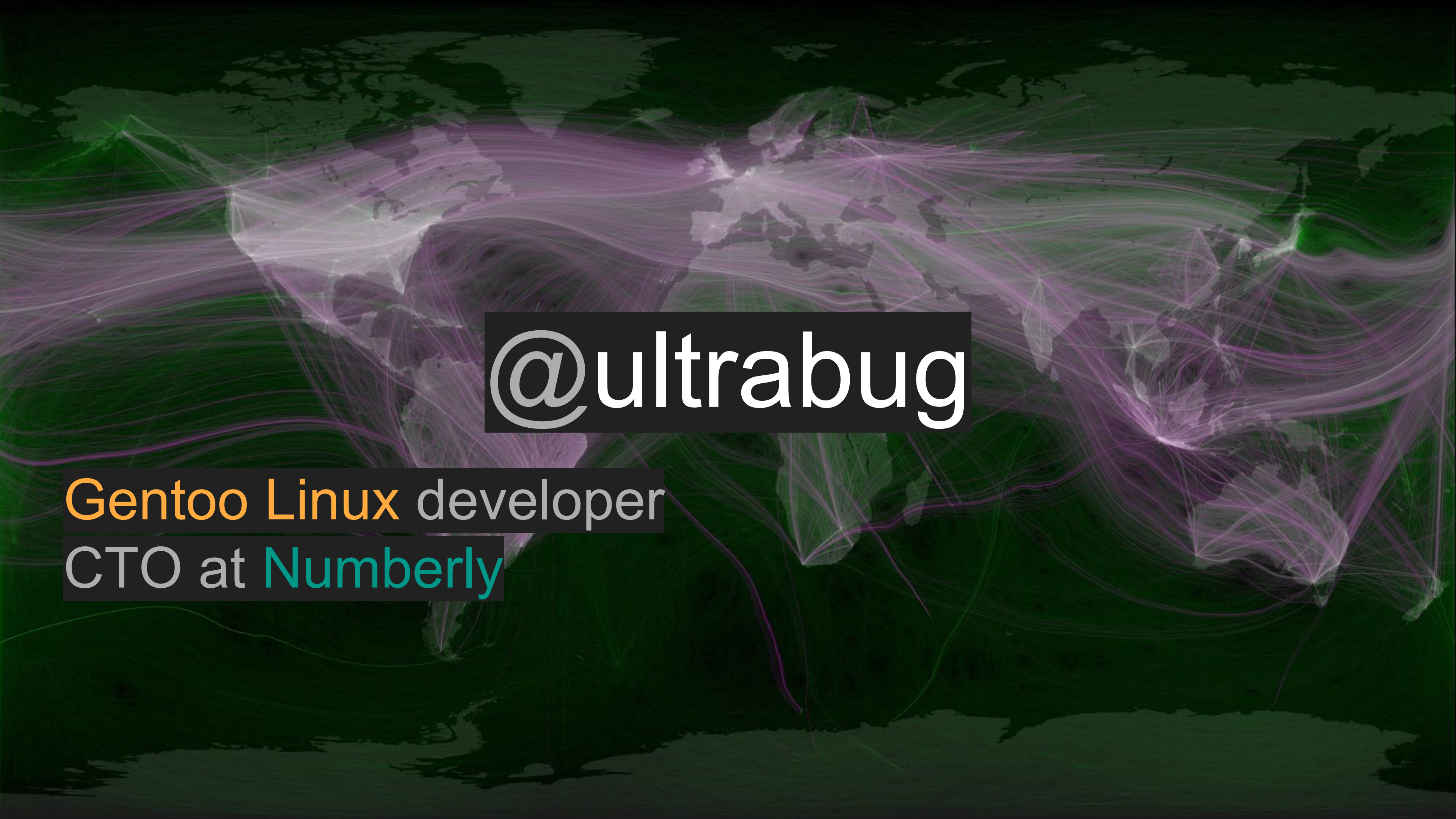
Consistent Hashing

in your python applications

Europython 2017



@ultrabug

Gentoo Linux developer

CTO at Numberly

History & main use cases

Distributed (web) caching (Akamai)

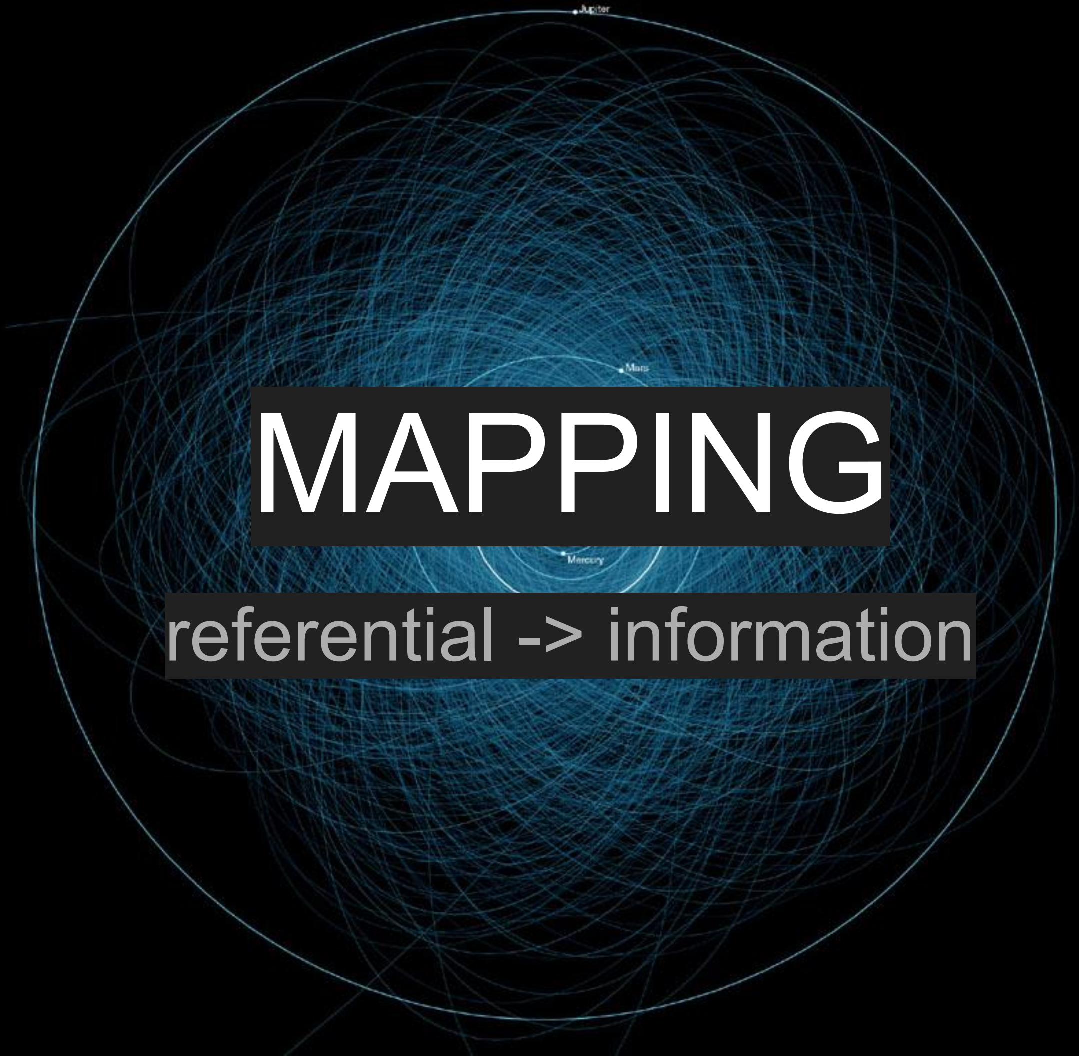
P2P (Chord & BitTorrent)

Distributed databases (data distribution / sharding)

- Amazon DynamoDB
- Cassandra / ScyllaDB
- Riak
- CockroachDB

MAPPING

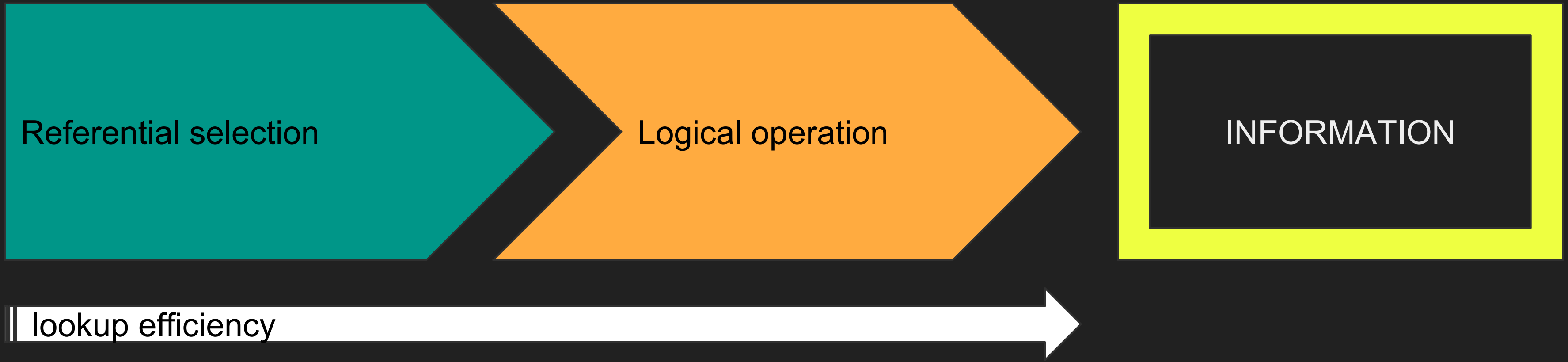
referential -> information



Phonebook

name -> phone number

Map logic

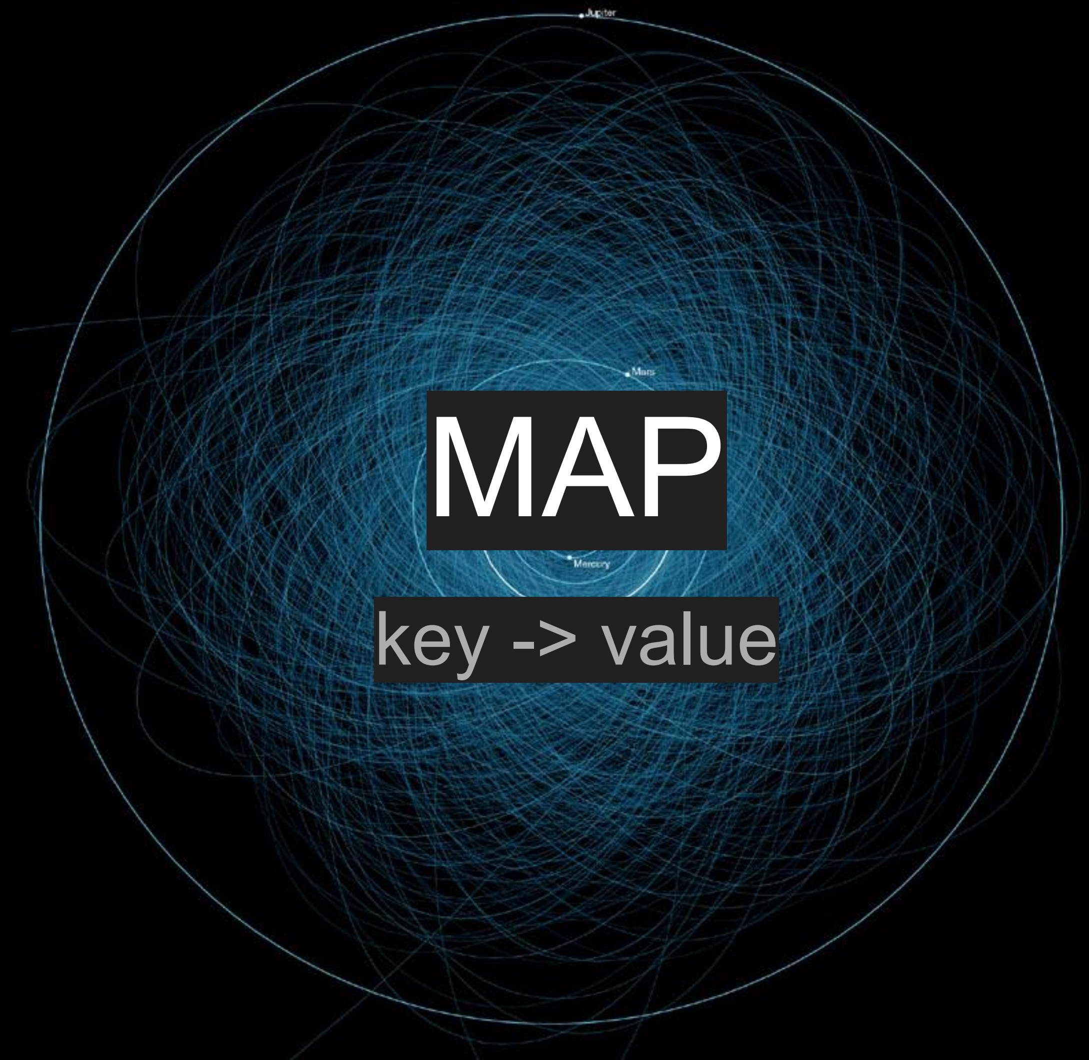


Referential selection

Logical operation

INFORMATION

lookup efficiency



MAP

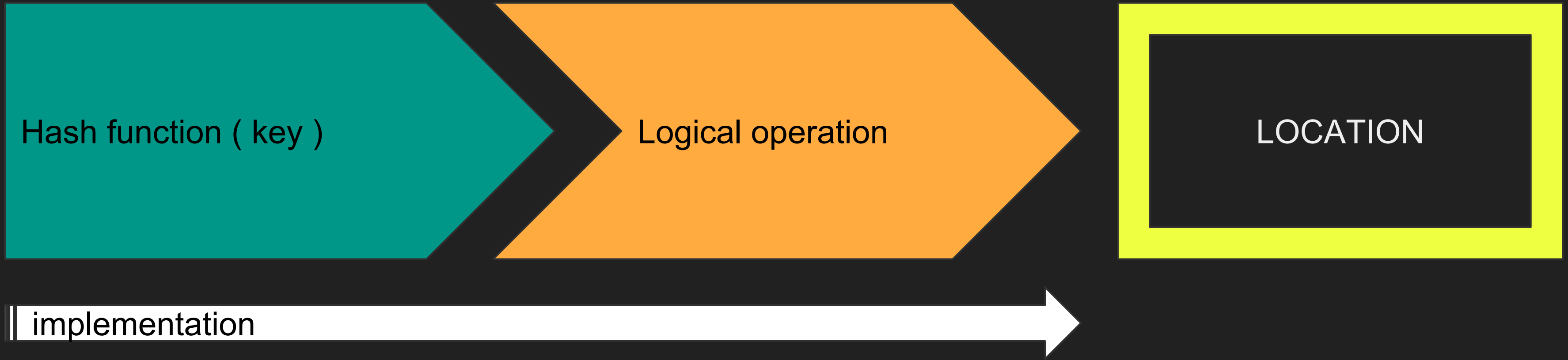
key -> value

Python dict()

{key: value}

Python dict() is a Hash Table

Hash Table logic



Hash function (key)

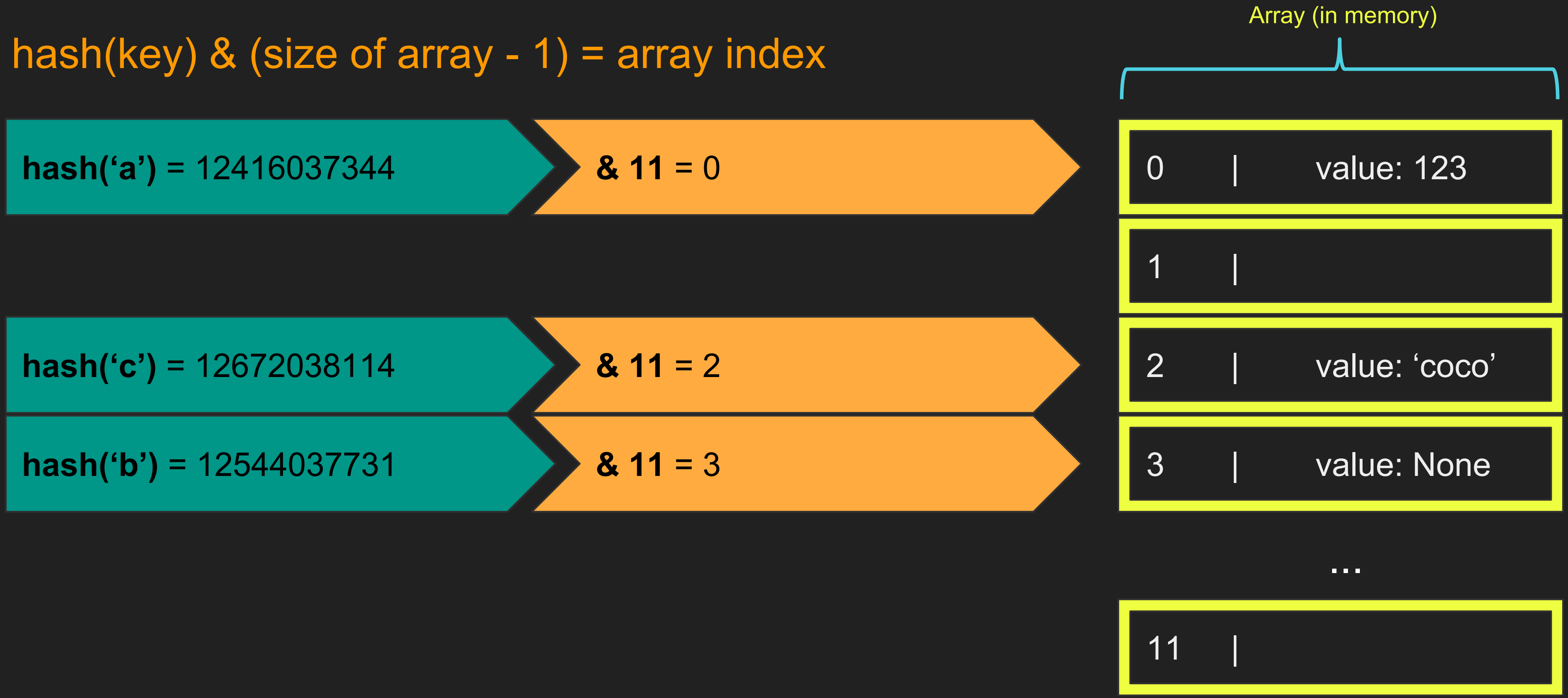
Logical operation

LOCATION

implementation

Python dict() implementation

hash(key) & (size of array - 1) = array index



Array (in memory)

hash(‘a’) = 12416037344 → **& 11** = 0

hash(‘c’) = 12672038114 → **& 11** = 2

hash(‘b’) = 12544037731 → **& 11** = 3

0 | value: 123

1 |

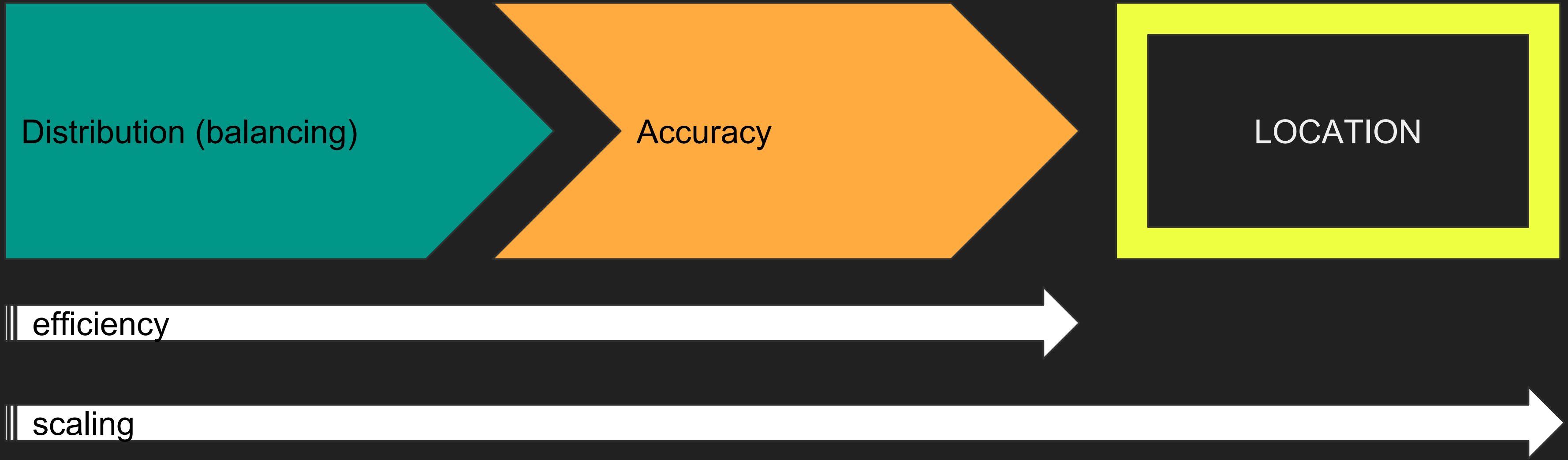
2 | value: ‘coco’

3 | value: None

...

11 |

Key factors to consider



Distribution (balancing)

Accuracy

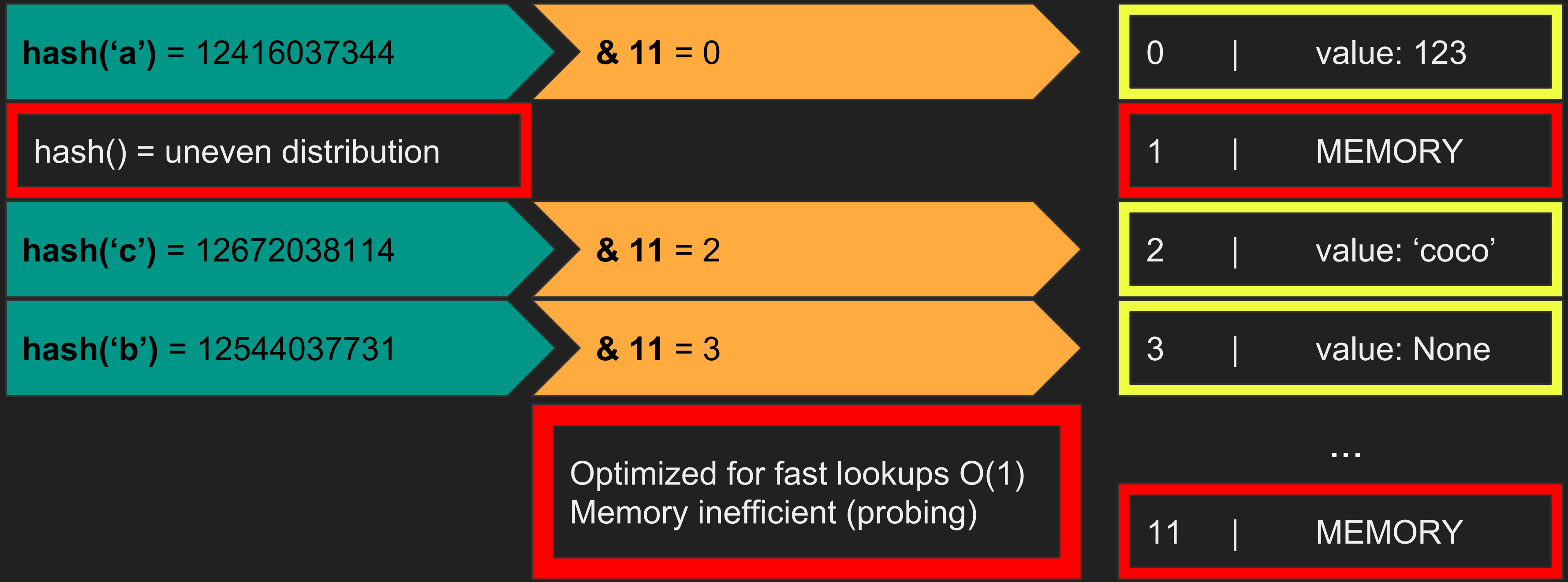
LOCATION

efficiency

scaling

Python dict efficiency & scaling

hash(key) & (size of array - 1) = array index



hash(‘a’) = 12416037344 → **& 11** = 0

hash() = uneven distribution

hash(‘c’) = 12672038114 → **& 11** = 2

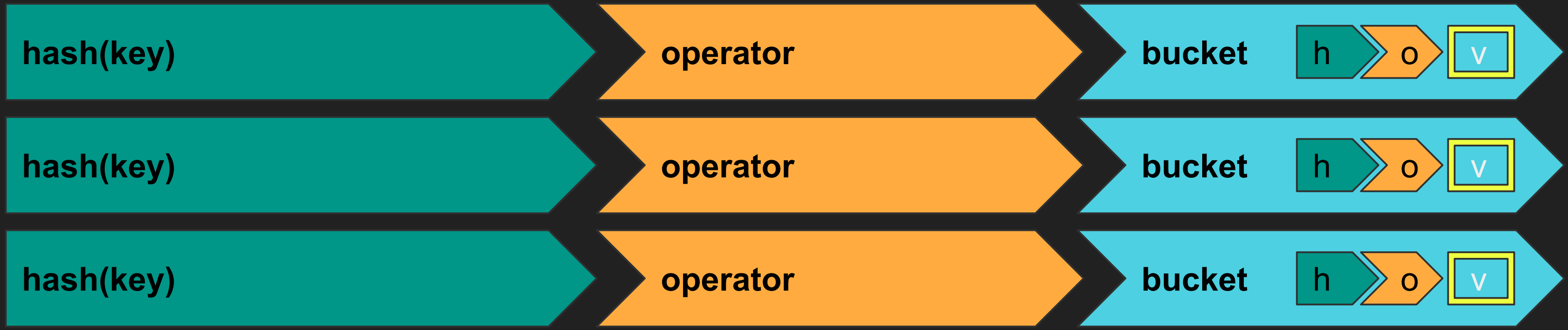
hash(‘b’) = 12544037731 → **& 11** = 3

Optimized for fast lookups O(1)
Memory inefficient (probing)

0 | value: 123
1 | MEMORY
2 | value: ‘coco’
3 | value: None
...
11 | MEMORY

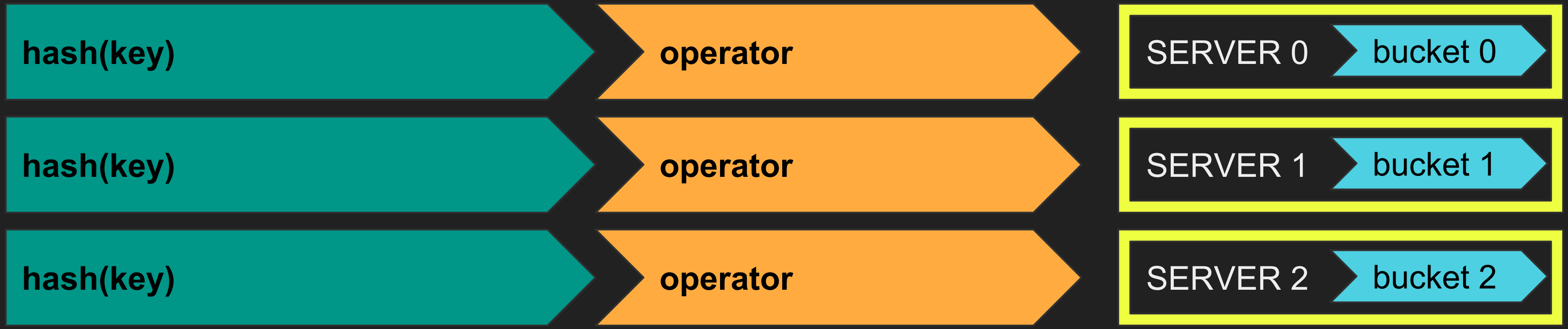
Distributed Hash Tables
(DHT)

Split your key space into buckets



the hash function will impact the size of each bucket

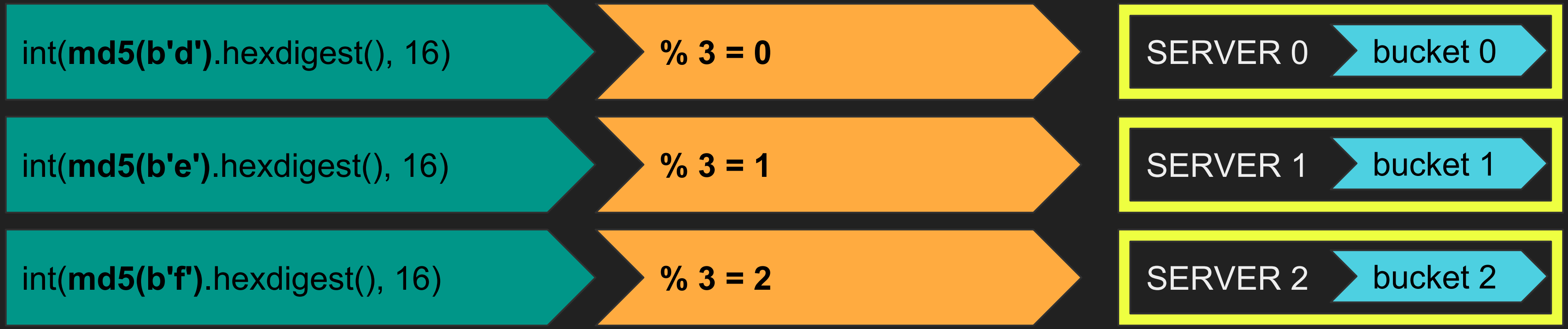
Distribute your buckets to servers



what’s the best operator function to find the server hosting the bucket for my key ?

Naive DHT implementation

md5(key) % (number of buckets) = server



int(**md5(b'd')**.hexdigest(), 16) **% 3 = 0** SERVER 0 bucket 0

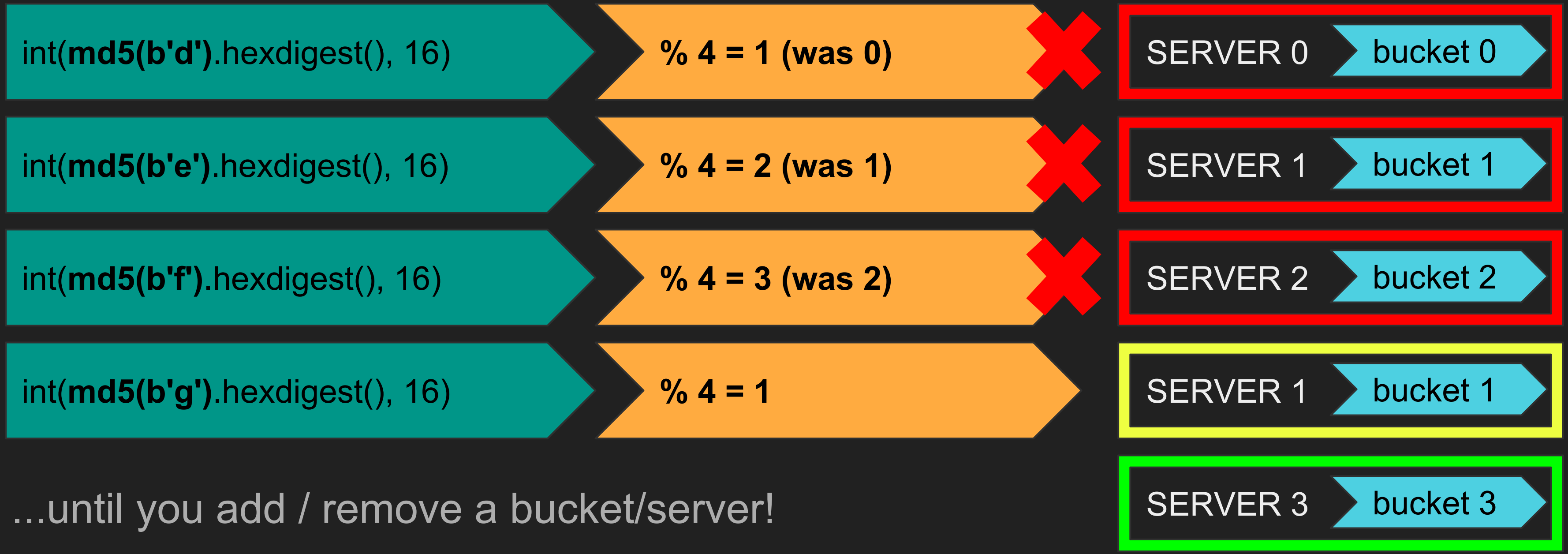
int(**md5(b'e')**.hexdigest(), 16) **% 3 = 1** SERVER 1 bucket 1

int(**md5(b'f')**.hexdigest(), 16) **% 3 = 2** SERVER 2 bucket 2

simple & looking good...

Naive DHT implementation

md5(key) % (number of buckets) = server



int(**md5(b'd')**.hexdigest(), 16) **% 4 = 1 (was 0)** SERVER 0 bucket 0

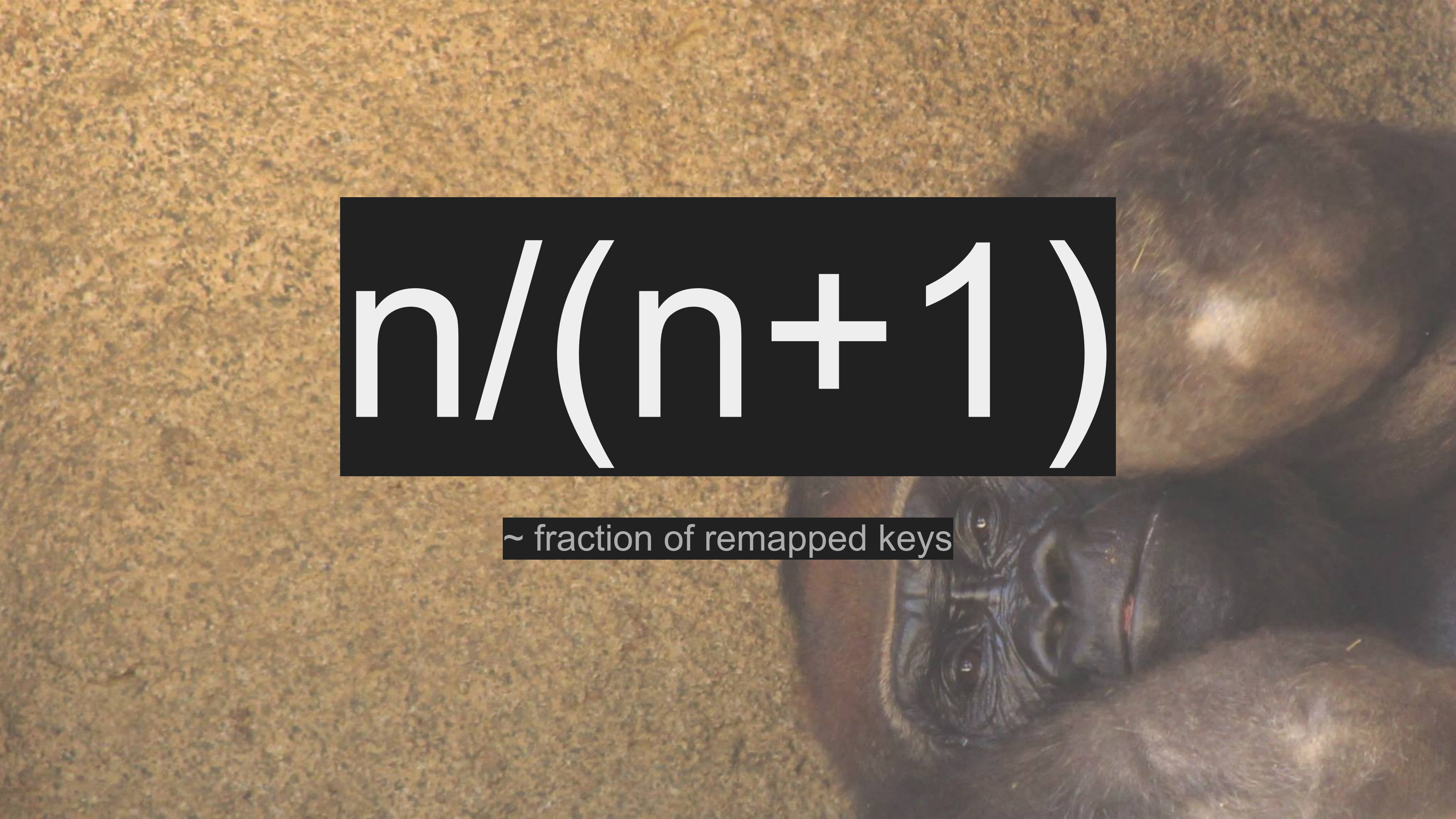
int(**md5(b'e')**.hexdigest(), 16) **% 4 = 2 (was 1)** SERVER 1 bucket 1

int(**md5(b'f')**.hexdigest(), 16) **% 4 = 3 (was 2)** SERVER 2 bucket 2

int(**md5(b'g')**.hexdigest(), 16) **% 4 = 1** SERVER 1 bucket 1

SERVER 3 bucket 3

...until you add / remove a bucket/server!



$n/(n+1)$

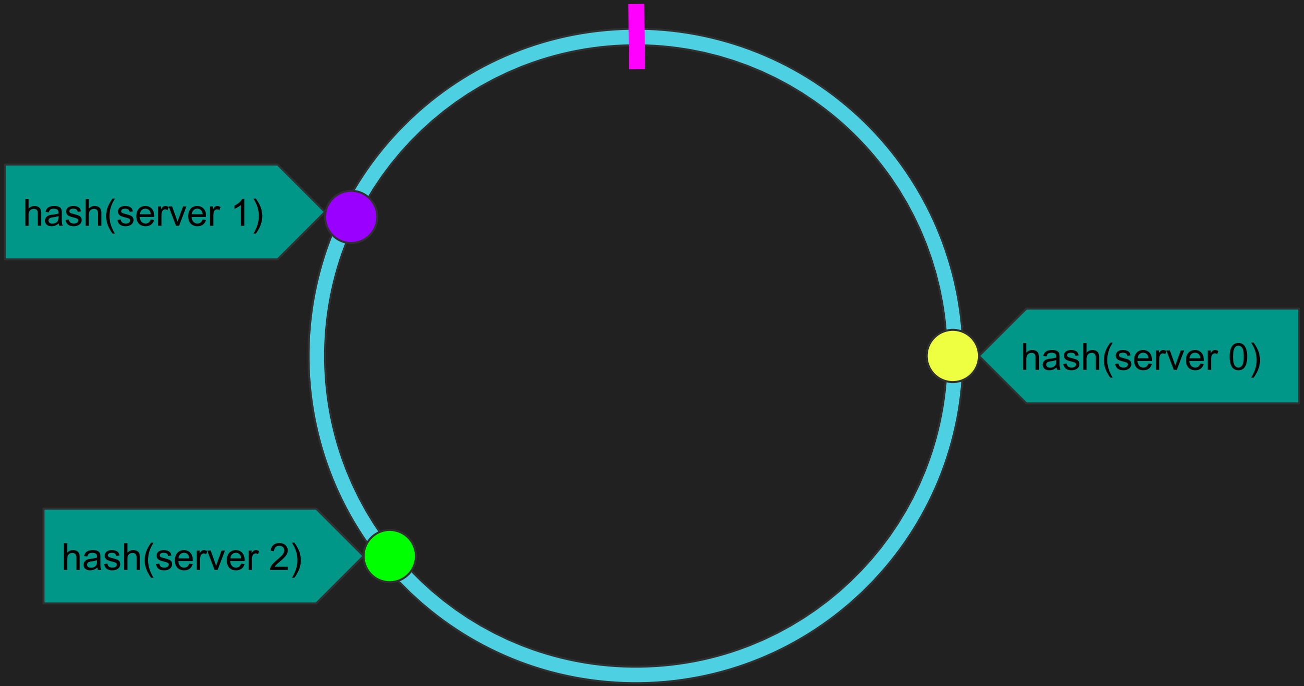
~ fraction of remapped keys

HELP!

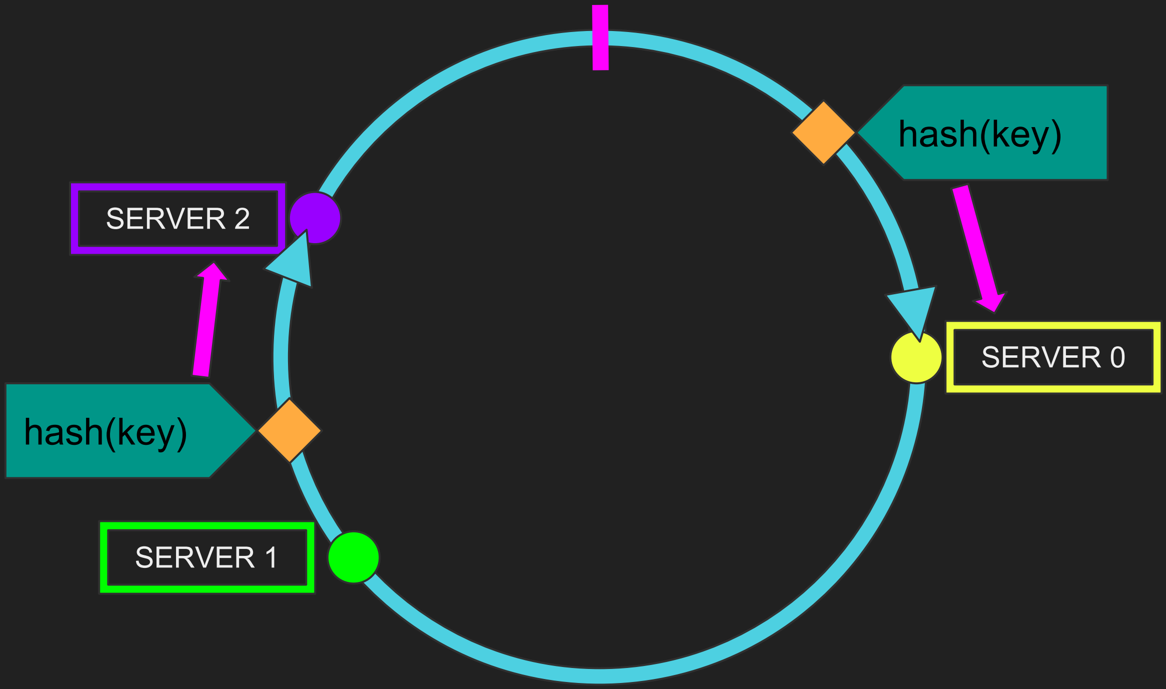
we need consistency

The Hash Ring

Place your servers on the continuum (ring)



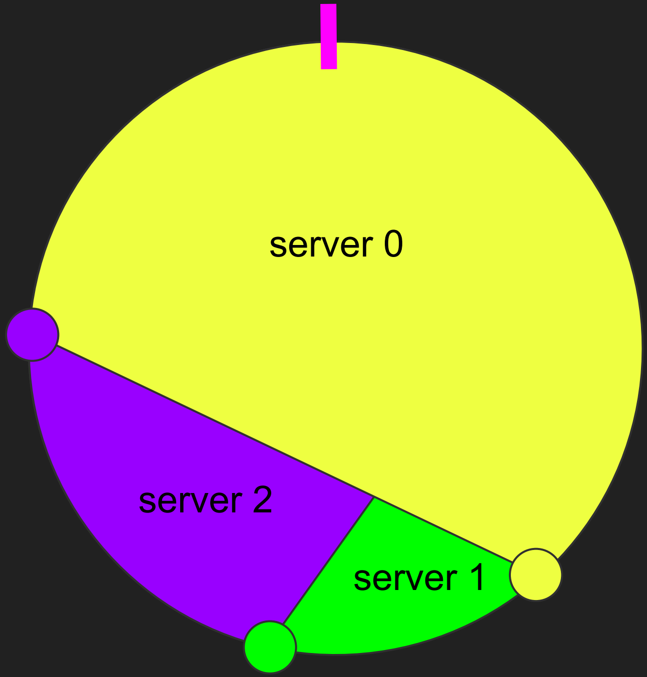
Keys’ bucket is on the next server in the ring



1/n

~ fraction of remapped keys

Uneven partitions lead to hotspots



hash functions are not perfect

Which hash function to use ?

Cryptographic hash functions

- MD5
- SHA1
- SHA256

+ standard

+ adoption

− need conversion to int

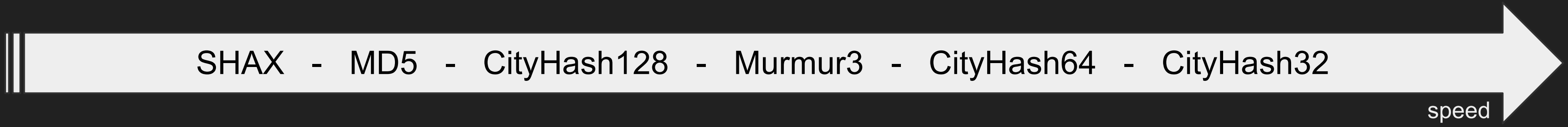
Non cryptographic hash functions

- CityHash (google)
- Murmur (v3)

+ optimized for key lookups

+ fast

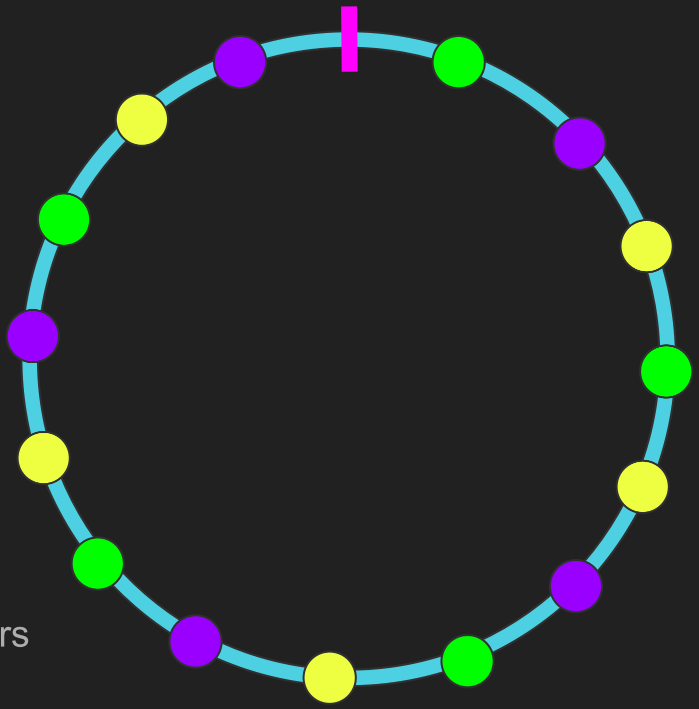
− need of C libs



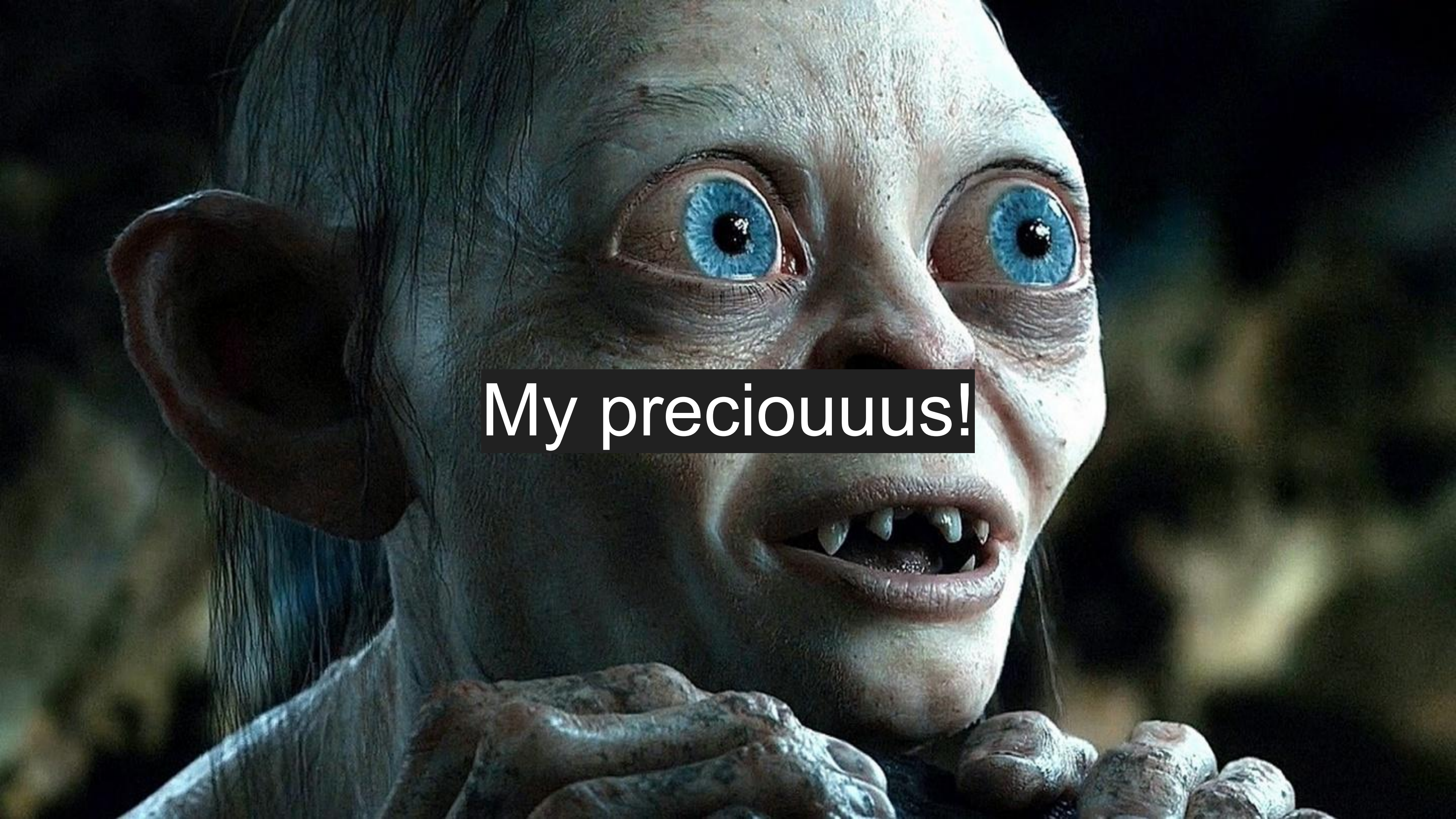
SHAX - MD5 - CityHash128 - Murmur3 - CityHash64 - CityHash32

speed

Hash Rings vnodes & weights mitigate hotspots



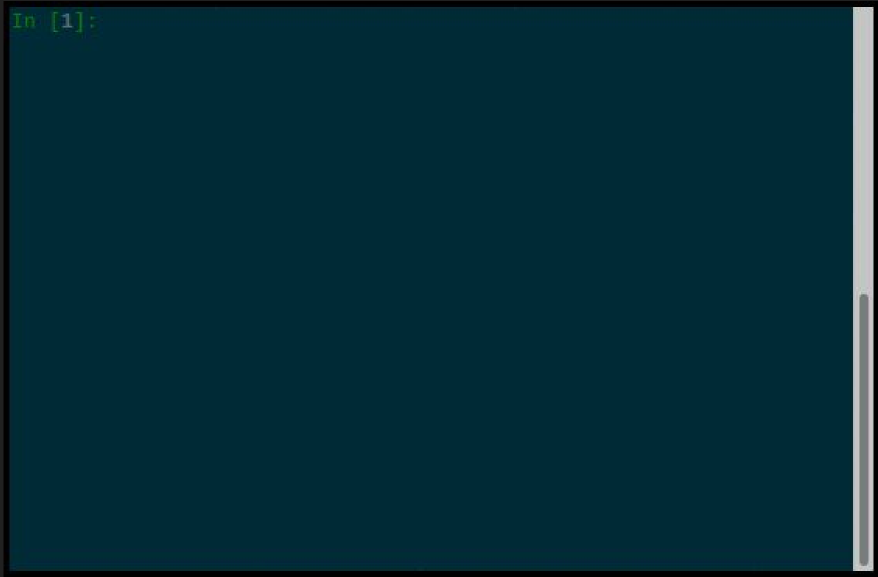
reduces load variance on servers



My preciouuus!

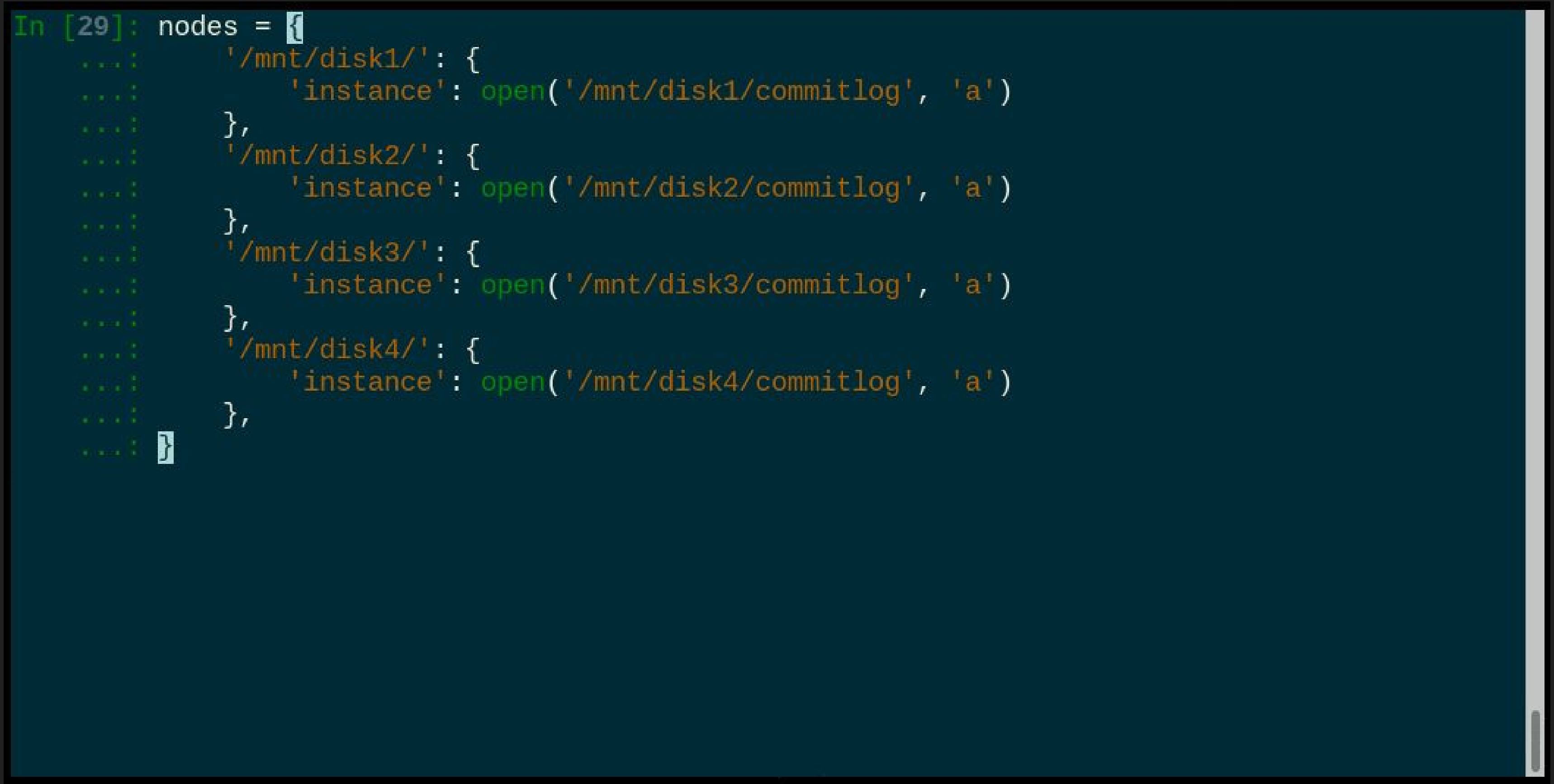
Consistent Hashing implementations in python

ConsistentHashing	A simple implement of consistent hashing
consistent_hash	The algorithm is the same as libketama
hash_ring	Using md5 as hashing function
python-continuum	Using md5 as hashing function
uhashring	Full featured, ketama compatible



```
In [1]:
```

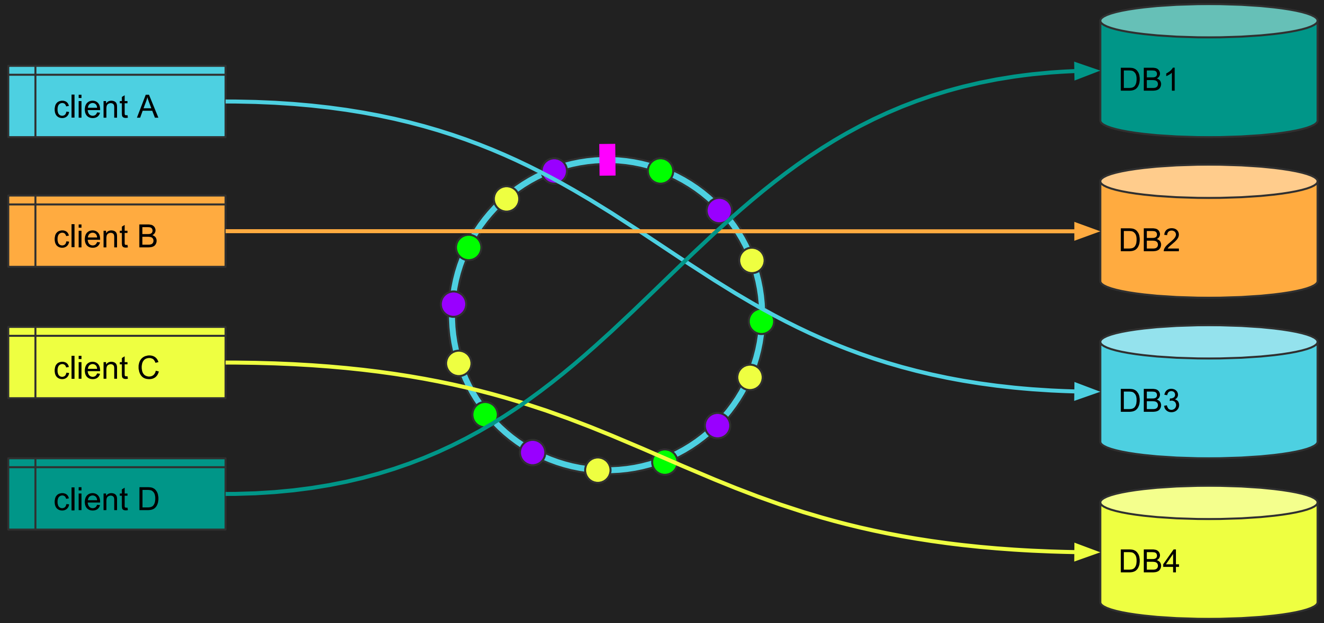
uhashring



```
In [29]: nodes = {
    ...:     '/mnt/disk1/': {
    ...:         'instance': open('/mnt/disk1/commitlog', 'a')
    ...:     },
    ...:     '/mnt/disk2/': {
    ...:         'instance': open('/mnt/disk2/commitlog', 'a')
    ...:     },
    ...:     '/mnt/disk3/': {
    ...:         'instance': open('/mnt/disk3/commitlog', 'a')
    ...:     },
    ...:     '/mnt/disk4/': {
    ...:         'instance': open('/mnt/disk4/commitlog', 'a')
    ...:     },
    ...: }
```

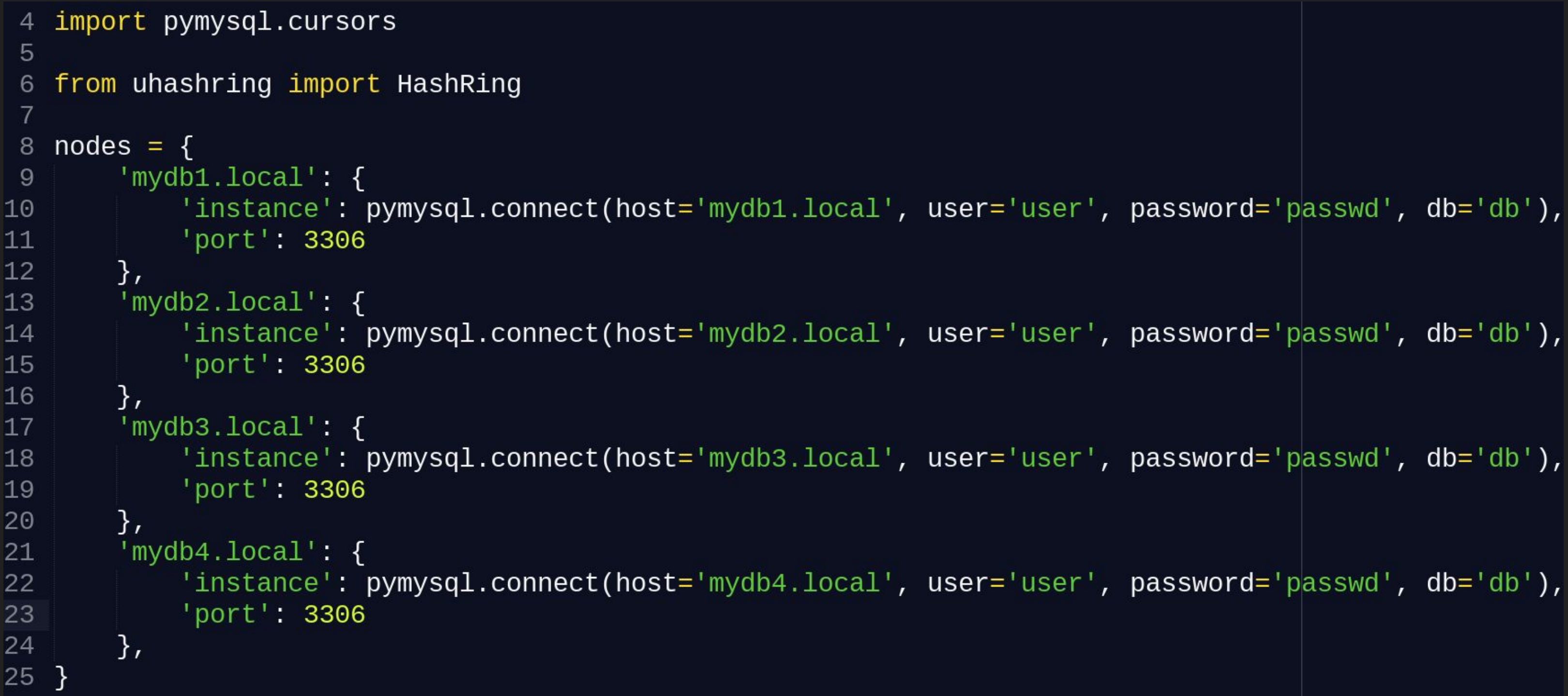
Example use case #1

Database instances distribution



Example use case #1

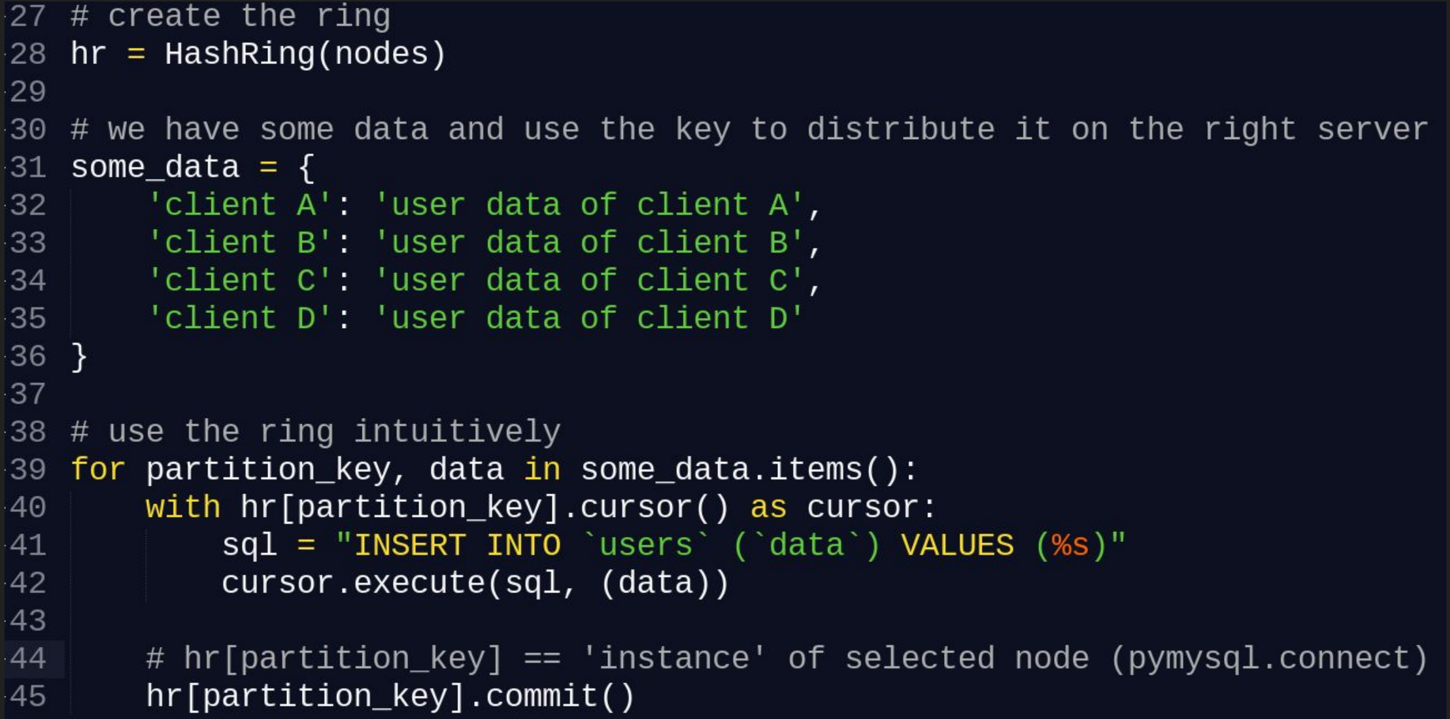
Database instances distribution



```
import pymysql.cursors

from uhashring import HashRing

nodes = {
    'mydb1.local': {
        'instance': pymysql.connect(host='mydb1.local', user='user', password='passwd', db='db'),
        'port': 3306
    },
    'mydb2.local': {
        'instance': pymysql.connect(host='mydb2.local', user='user', password='passwd', db='db'),
        'port': 3306
    },
    'mydb3.local': {
        'instance': pymysql.connect(host='mydb3.local', user='user', password='passwd', db='db'),
        'port': 3306
    },
    'mydb4.local': {
        'instance': pymysql.connect(host='mydb4.local', user='user', password='passwd', db='db'),
        'port': 3306
    },
}
```

Example use case #1

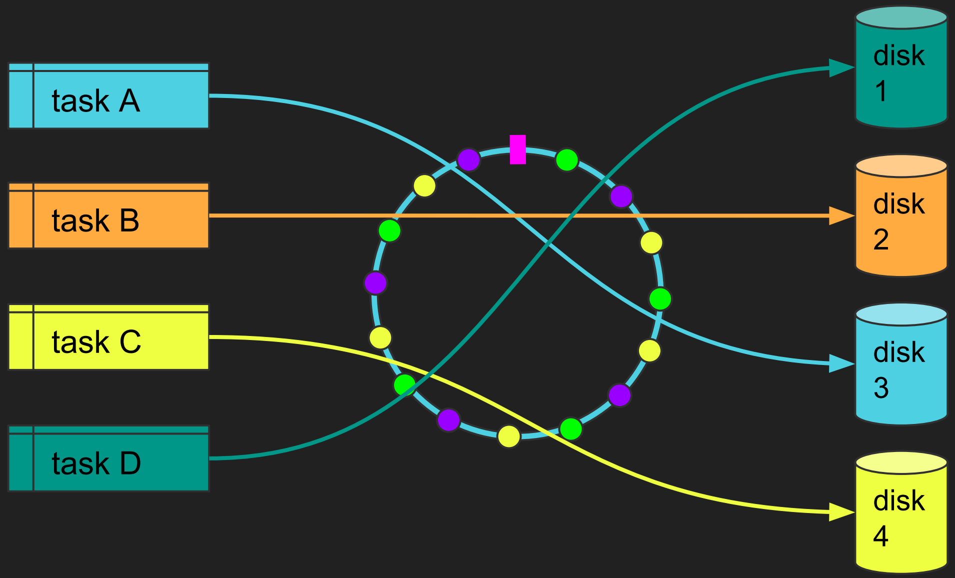
Database instances distribution

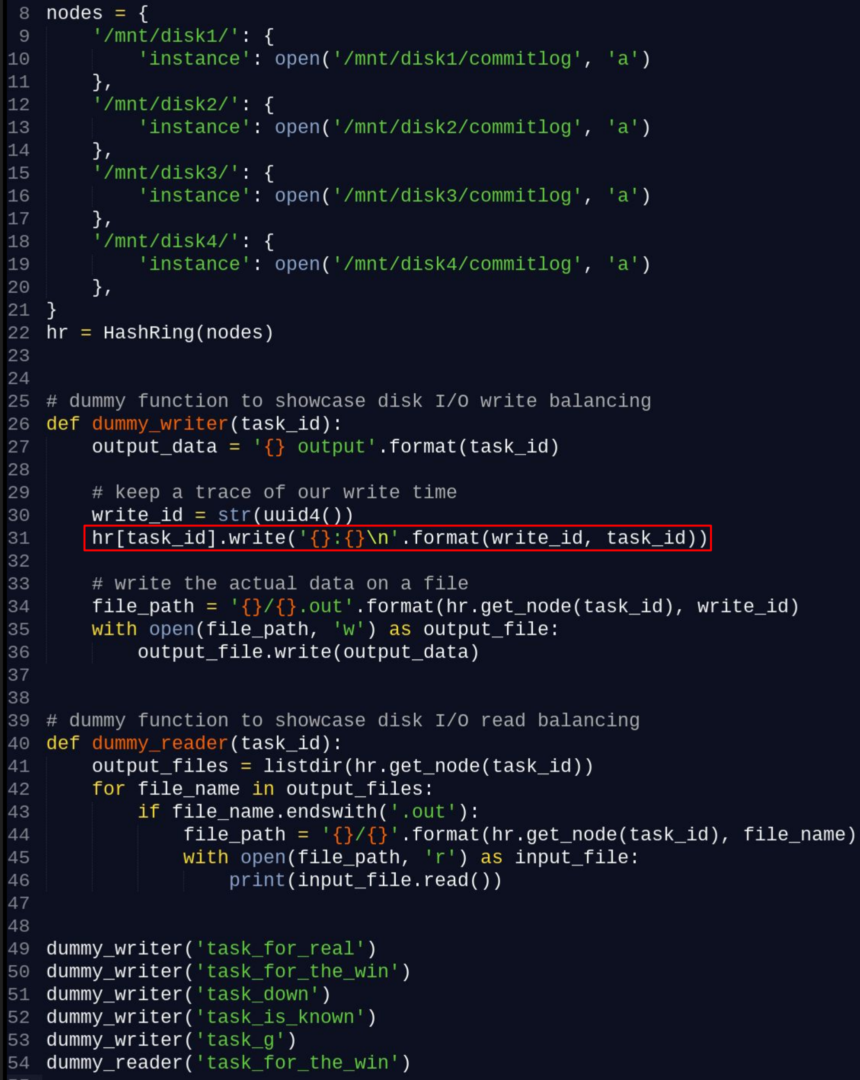


```python
# create the ring
hr = HashRing(nodes)

# we have some data and use the key to distribute it on the right server
some_data = {
    'client A': 'user data of client A',
    'client B': 'user data of client B',
    'client C': 'user data of client C',
    'client D': 'user data of client D'
}

# use the ring intuitively
for partition_key, data in some_data.items():
    with hr[partition_key].cursor() as cursor:
        sql = "INSERT INTO `users` (`data`) VALUES (%s)"
        cursor.execute(sql, (data))

    # hr[partition_key] == 'instance' of selected node (pymysql.connect)
    hr[partition_key].commit()
```

Example use case #2

Disk & network I/O distribution

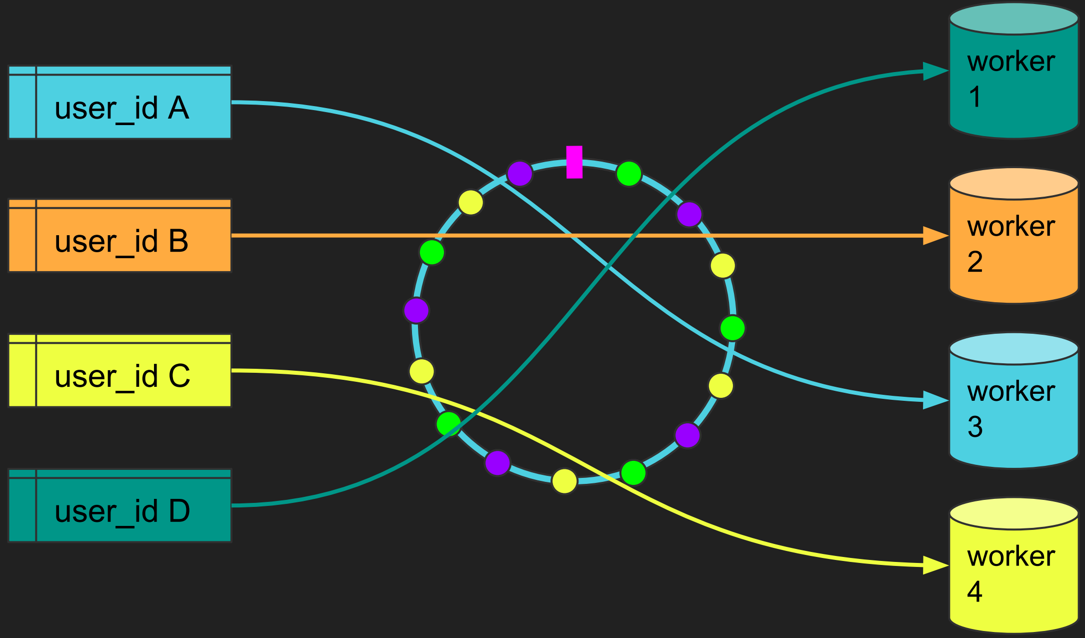


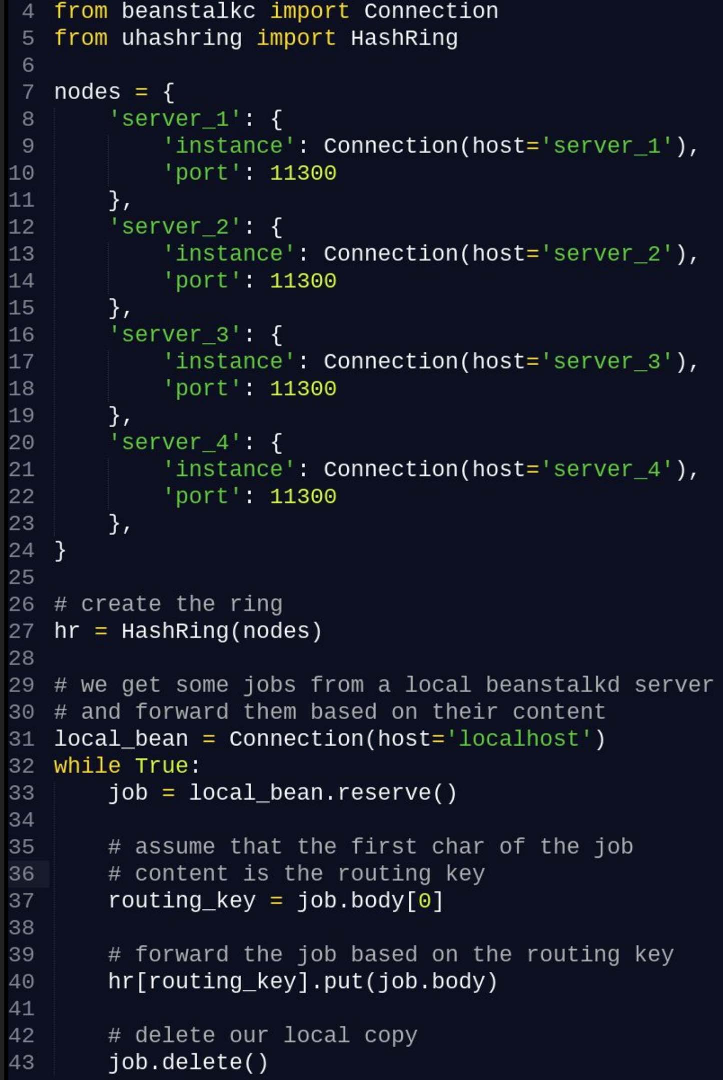


```python
nodes = {
    '/mnt/disk1/': {
        'instance': open('/mnt/disk1/commitlog', 'a')
    },
    '/mnt/disk2/': {
        'instance': open('/mnt/disk2/commitlog', 'a')
    },
    '/mnt/disk3/': {
        'instance': open('/mnt/disk3/commitlog', 'a')
    },
    '/mnt/disk4/': {
        'instance': open('/mnt/disk4/commitlog', 'a')
    },
}
hr = HashRing(nodes)


# dummy function to showcase disk I/O write balancing
def dummy_writer(task_id):
    output_data = '{} output'.format(task_id)

    # keep a trace of our write time
    write_id = str(uuid4())
    hr[task_id].write('{}:{}\n'.format(write_id, task_id))

    # write the actual data on a file
    file_path = '{}/{}.out'.format(hr.get_node(task_id), write_id)
    with open(file_path, 'w') as output_file:
        output_file.write(output_data)


# dummy function to showcase disk I/O read balancing
def dummy_reader(task_id):
    output_files = listdir(hr.get_node(task_id))
    for file_name in output_files:
        if file_name.endswith('.out'):
            file_path = '{}/{}'.format(hr.get_node(task_id), file_name)
            with open(file_path, 'r') as input_file:
                print(input_file.read())


dummy_writer('task_for_real')
dummy_writer('task_for_the_win')
dummy_writer('task_down')
dummy_writer('task_is_known')
dummy_writer('task_g')
dummy_reader('task_for_the_win')
```

Example use case #3

Log & tracing consistency

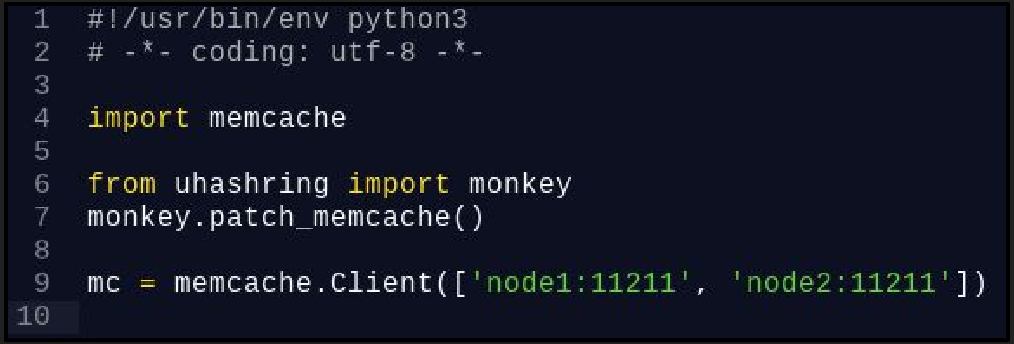




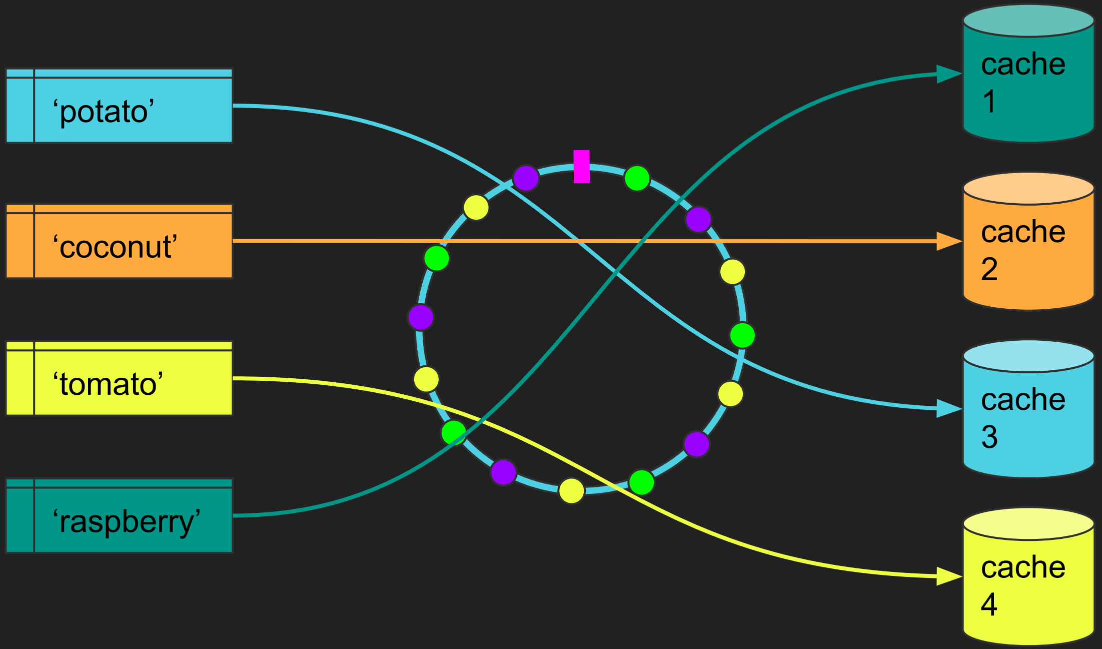
```python
from beanstalkc import Connection
from uhashring import HashRing

nodes = {
    'server_1': {
        'instance': Connection(host='server_1'),
        'port': 11300
    },
    'server_2': {
        'instance': Connection(host='server_2'),
        'port': 11300
    },
    'server_3': {
        'instance': Connection(host='server_3'),
        'port': 11300
    },
    'server_4': {
        'instance': Connection(host='server_4'),
        'port': 11300
    },
}

# create the ring
hr = HashRing(nodes)

# we get some jobs from a local beanstalkd server
# and forward them based on their content
local_bean = Connection(host='localhost')
while True:
    job = local_bean.reserve()

    # assume that the first char of the job
    # content is the routing key
    routing_key = job.body[0]

    # forward the job based on the routing key
    hr[routing_key].put(job.body)

    # delete our local copy
    job.delete()
```

Example use case #4

python-memcached consolidation



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import memcache

from uhashring import monkey
monkey.patch_memcache()

mc = memcache.Client(['node1:11211', 'node2:11211'])

```



Live demo raffle

List of GIFs

One of the GIF is the winner

Every participant is a node (bucket)

hash(WINNER_GIF_URL) picks the winner node

http://ep17.nbly.co

(silly live demo)

Thanks

github.com/ultrabug/ep2017

github.com/ultrabug/uhashring

@ultrabug