Memcached

Ryan Lv

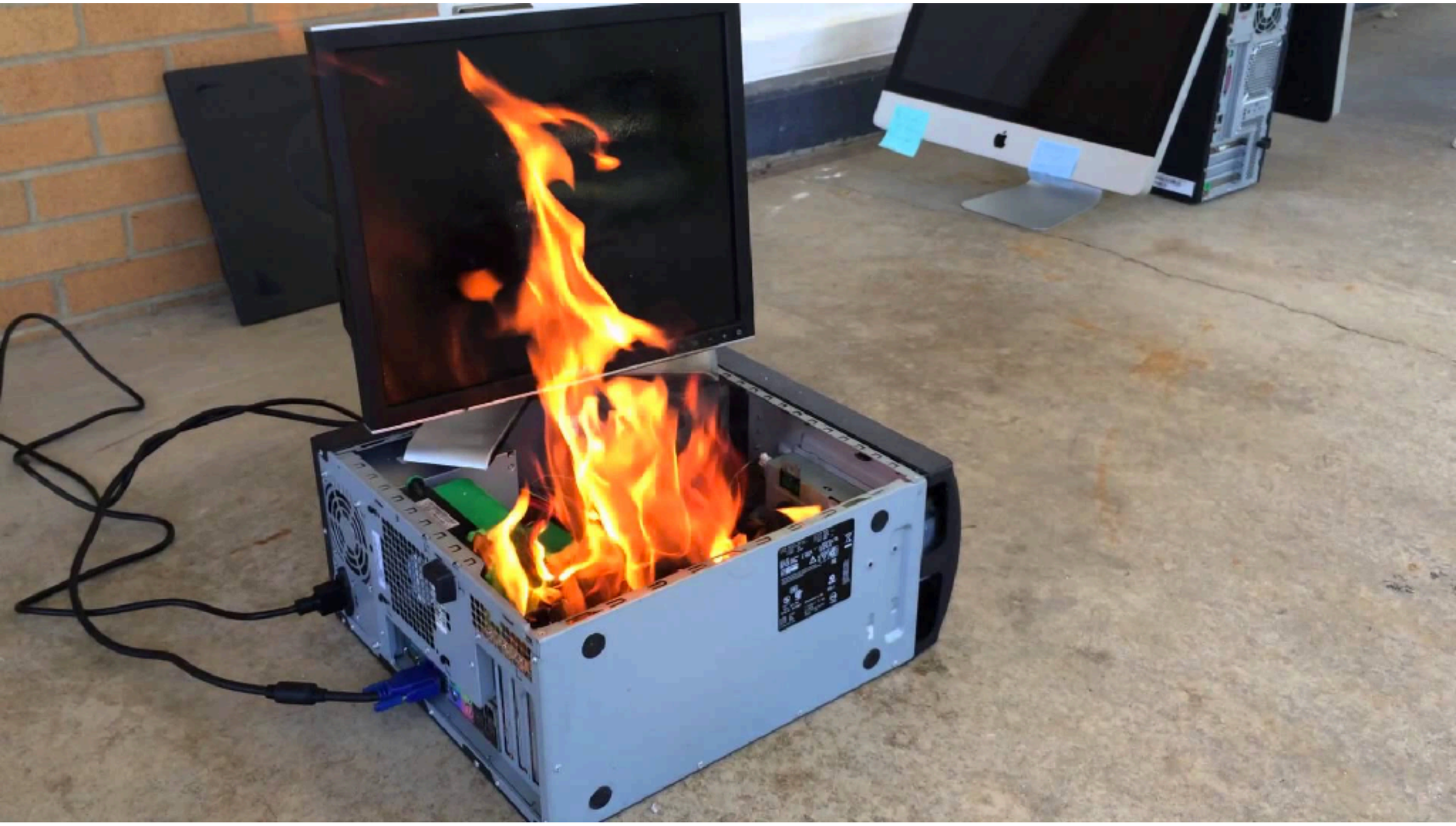
What is memcached used for?

Persistent

It’s important to understand that Memcached is not a persistent store.

What's the difference?

Memached Vs Database



It’s designed for caching and so there should be no real problem if the data is lost.

How many cache stores does Rails provide?

each method is quite different and carry their own set oof pros and cons.

- FileStore
- MemoryStore
- MemCacheStore

Why not FileStore?

The file store works well for smaller applications but isn't very efficient as reading from and writing to the hard drive is relatively slow. If we use this for a cache that's accessed frequently we'd be better off using something else.

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns
Send 1K bytes over 1 Gbps network       10,000   ns    0.01 ms
Read 4K randomly from SSD*             150,000   ns    0.15 ms
Read 1 MB sequentially from memory     250,000   ns    0.25 ms
Round trip within same datacenter      500,000   ns    0.5  ms
Read 1 MB sequentially from SSD*     1,000,000   ns    1    ms  4X memory
Disk seek                           10,000,000   ns   10    ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20    ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150    ms

Notes
-----
1 ns = 10-9 seconds
1 ms = 10-3 seconds
* Assuming ~1GB/sec SSD
```

Credit

By Jeff Dean: http://research.google.com/people/jeff/
Originally by Peter Norvig: http://norvig.com/21-days.html#answers

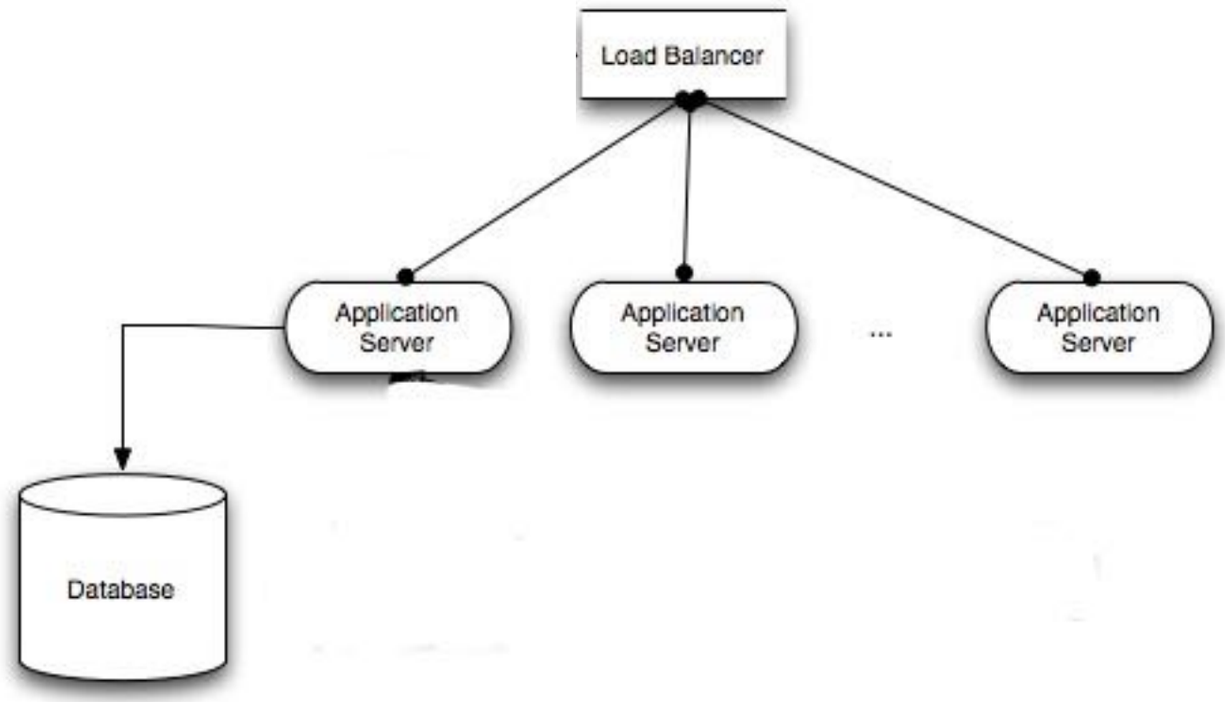
Contributions

Some updates from: https://gist.github.com/2843375
Great 'humanized' comparison version: https://gist.github.com/2843375
Visual comparison chart: http://i.imgur.com/k0t1e.png
Nice animated presentation of the data: http://prezi.com/pdkvgys-r0y6/latency-numbers-for-programmers-web-development/

Why not MemoryStore?

It's fast, but...

Cache can't be shared between process/ servers



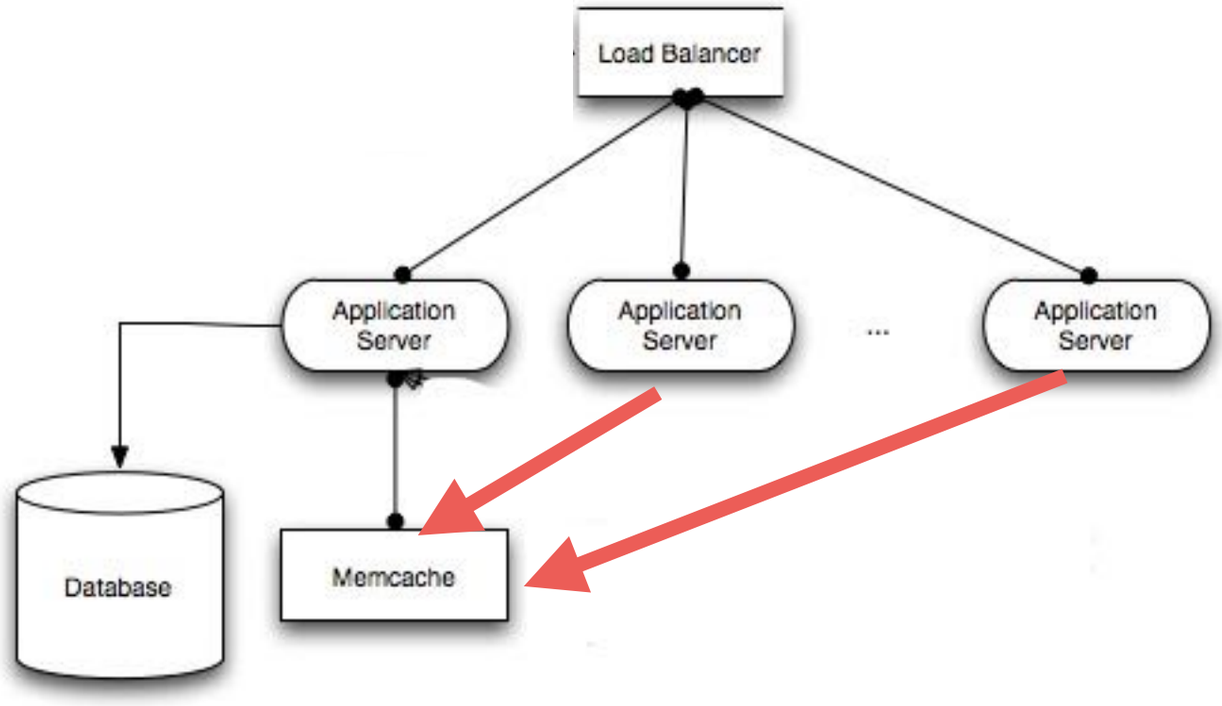
The default used to be a memory store which stored the cache in local memory of that Rails process. This issue with this is that in production we often have multiple Rails instances running and each of these will have their own cache store which isn’t a good use of resources.

Cache can't be shared between process/ servers

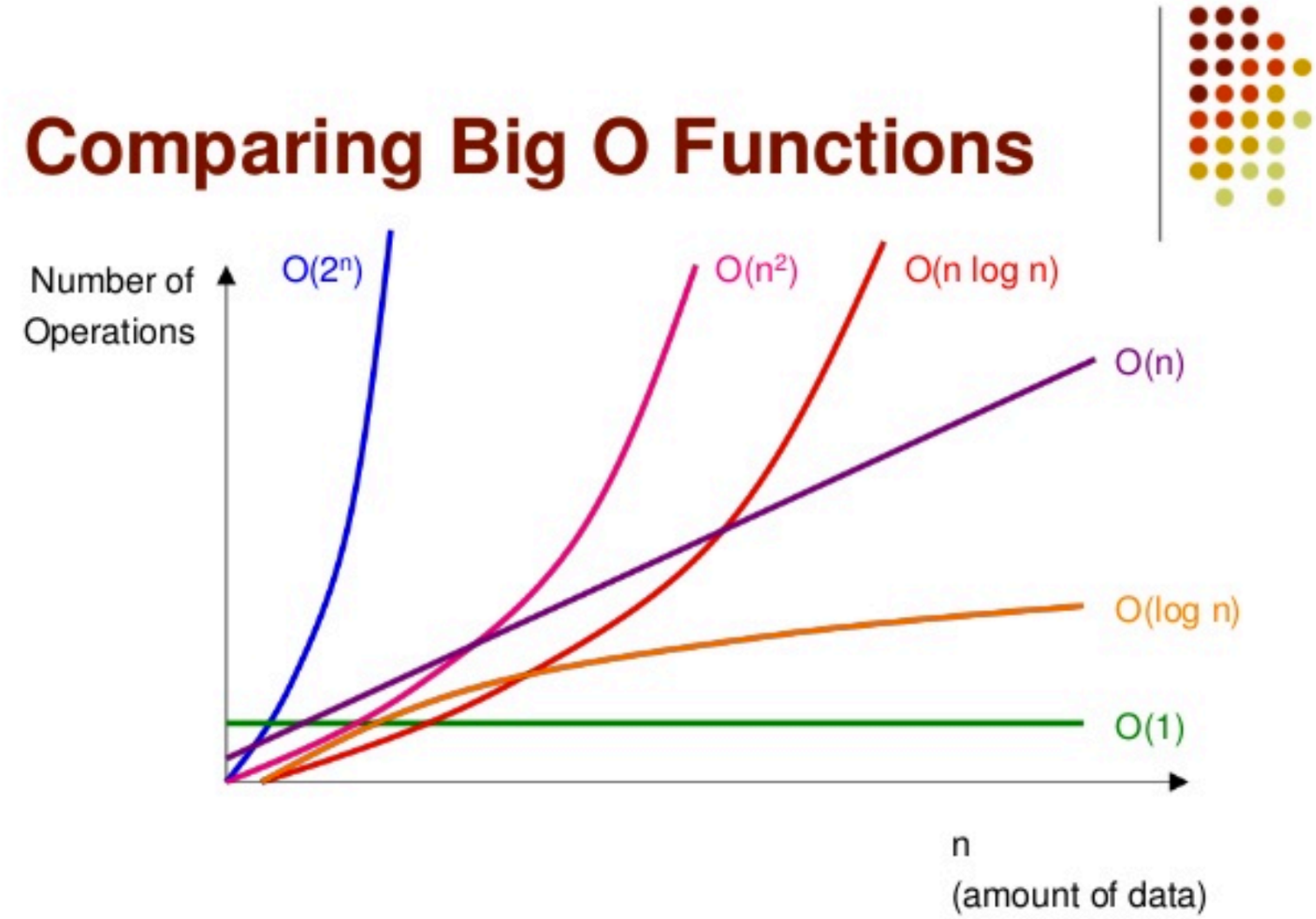
The default used to be a memory store which stored the cache in local memory of that Rails process. This issue with this is that in production we often have multiple Rails instances running and each of these will have their own cache store which isn’t a good use of resources.

Why memcached?

- Cache is stored in memory
- Cache is shared accross multiple Rails instances or even separate servers.
- Scalable



How will it scale when you Increase the amount of cache?



For Databases

select * from users where id = 3

O(log n)

select * from users join orders on users.id = orders.user_id

$O(n^2)$

Things may become worse.

For Memcached

O(1)

Most of memcache functionality (add, get, set, flush etc) are o(1). This means they are constant time functions. It does not matter how many items there are inside the cache, the functions will take just as long as they would with just 1 item inside the cache.

What would happen when memory are run out?

2017-05-11

What does this timestamp mean?

2017-05-10

2017-05-07

2017-05-08

Internally, all objects have a “counter”. This counter holds a timestamp. Every time a new object is created, that counter will be set to the current time. When an object gets FETCHED, it will reset that counter to the current time as well. As soon as memcache needs to “evict” an object to make room for newer objects, it will find the lowest counter. That is the object that isn’t fetched or is fetched the longest time ago (and probably isn’t needed that much, otherwise the counter would be closed to the current timestamp).

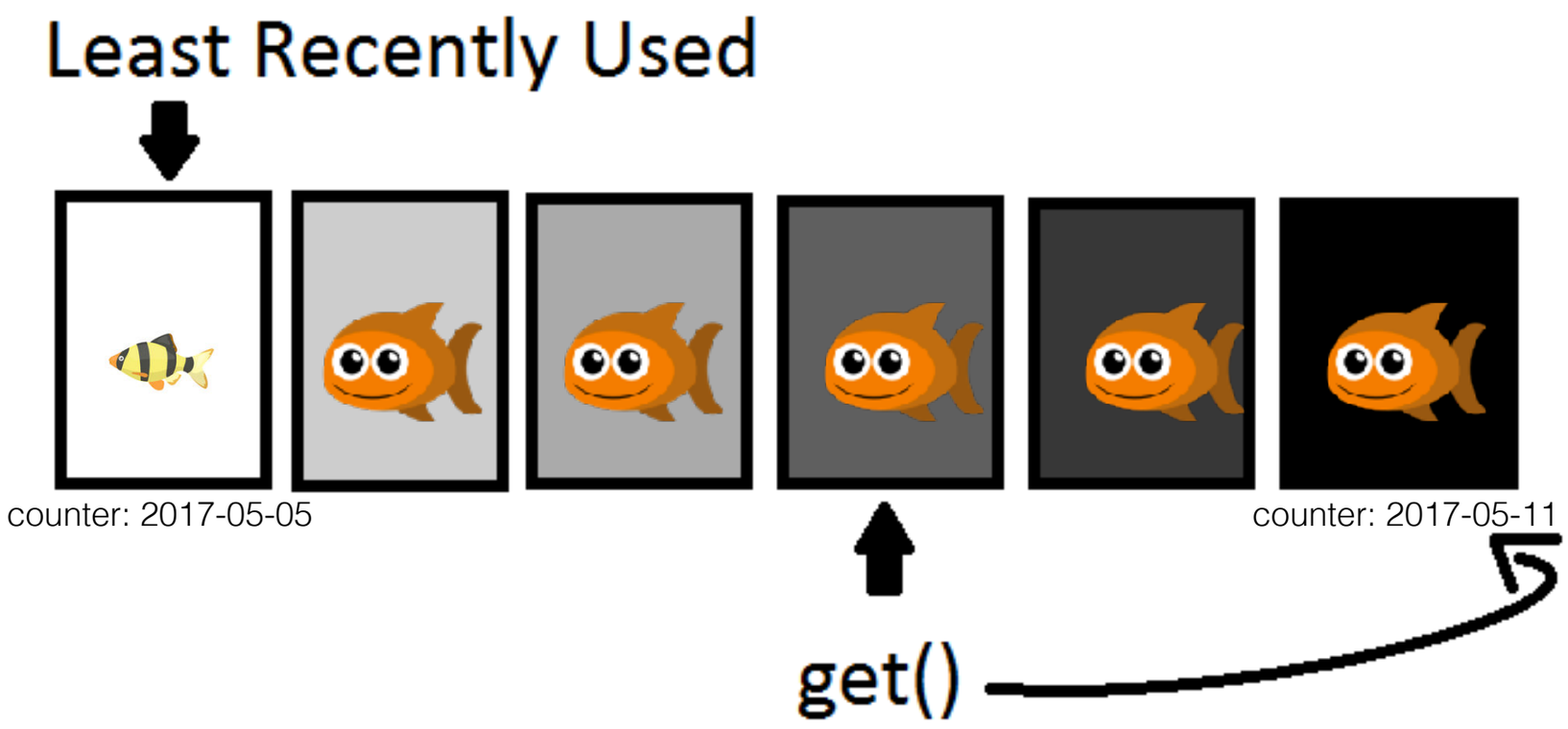
https://www.adayinthelifeof.nl/2011/02/06/memcache-internals/

2017-05-09

2017-05-05

2017-05-06

LRU


Least Recently Used
counter: 2017-05-05
counter: 2017-05-11
get()

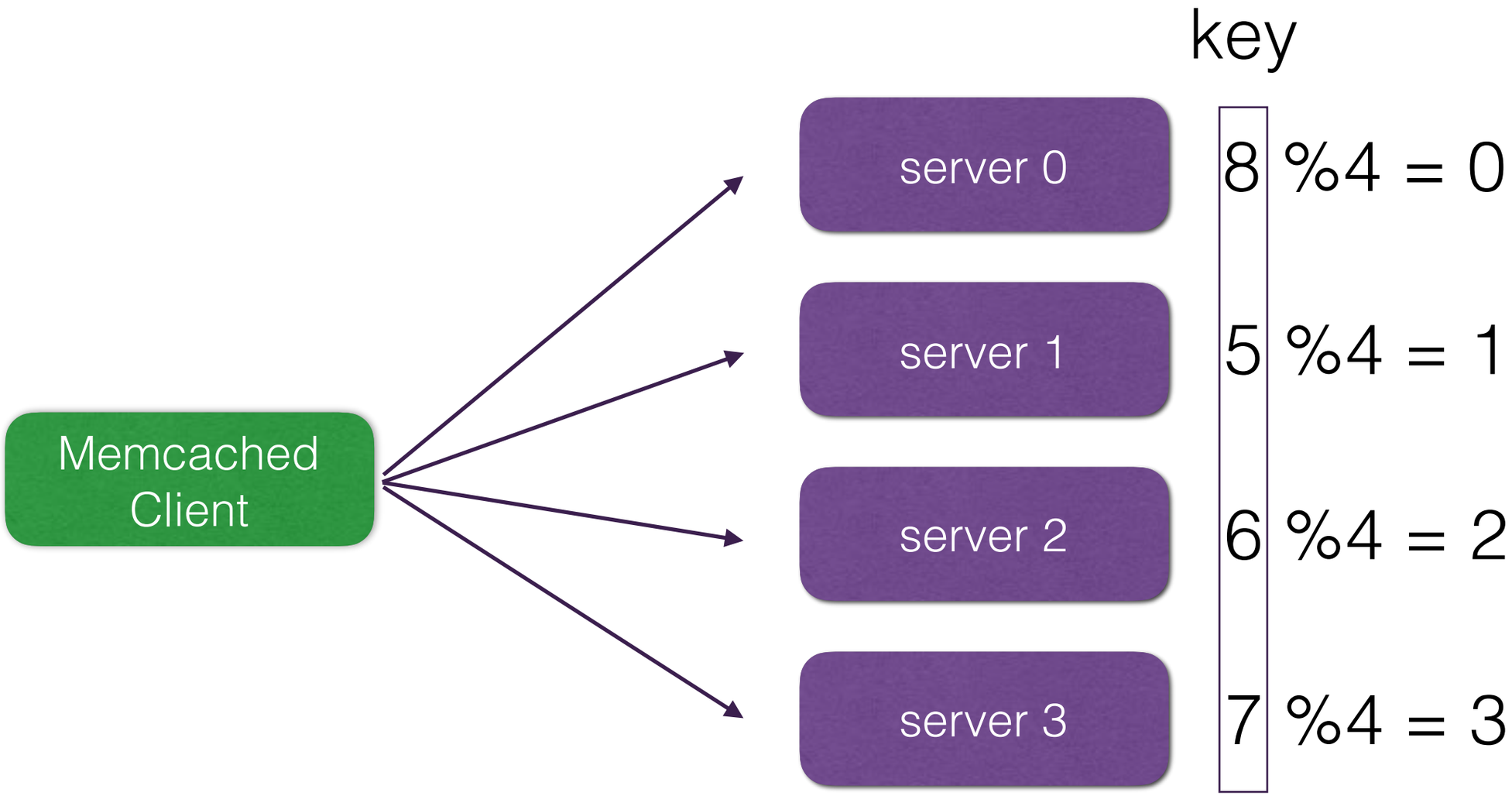
Memcached is distributed

But, how?

Assumption 1

which server = hash(cache_key) % number_of_servers

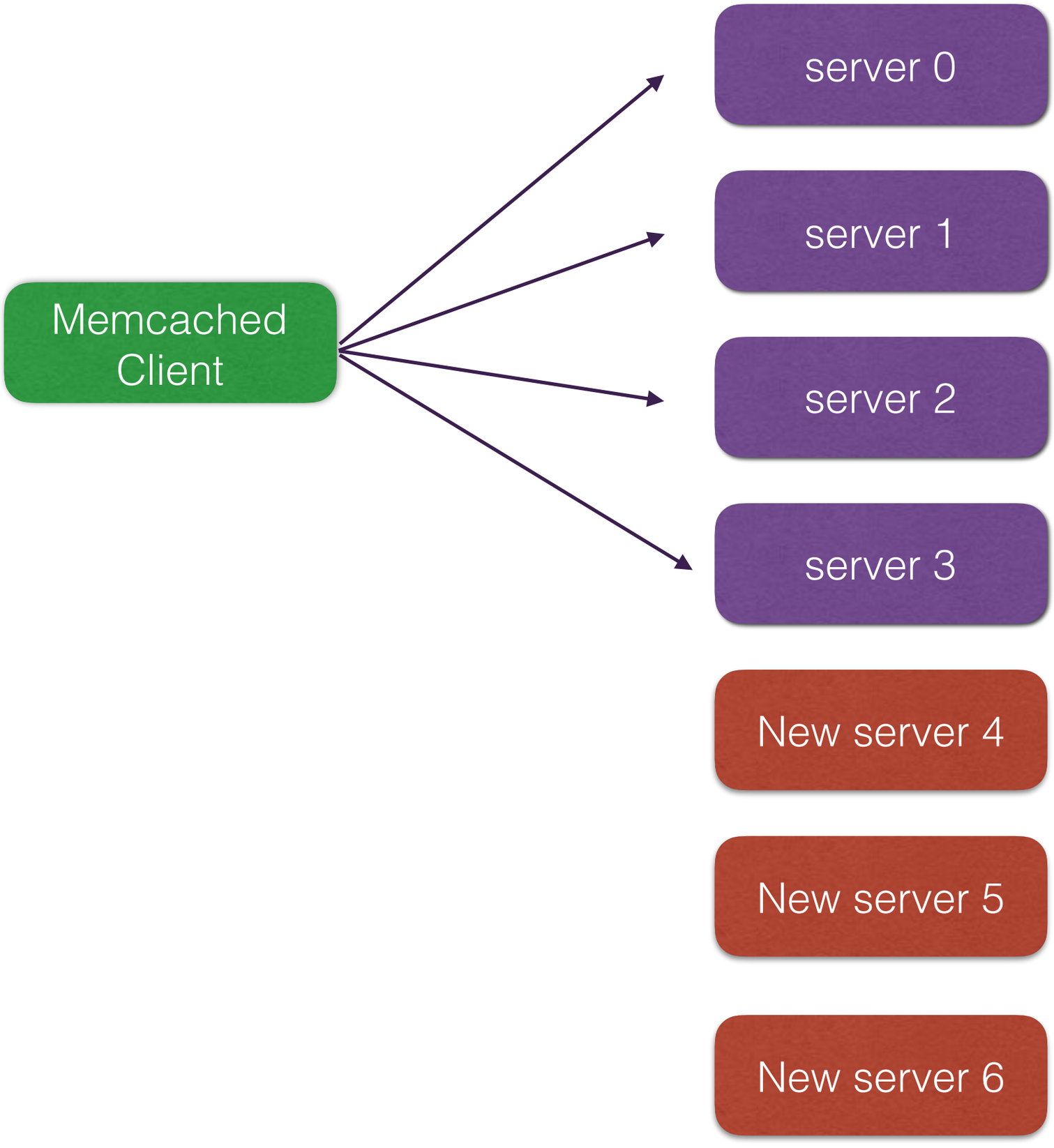
Client library decides which instance to write or read from.



Elasticsearch route data to shards in this way, the disadvantages are ...

shard = hash(routing) % number_of_primary_shards

Add servers


Memcached Client
server 0
server 1
server 2
server 3
New server 4
New server 5
New server 6

Before

7

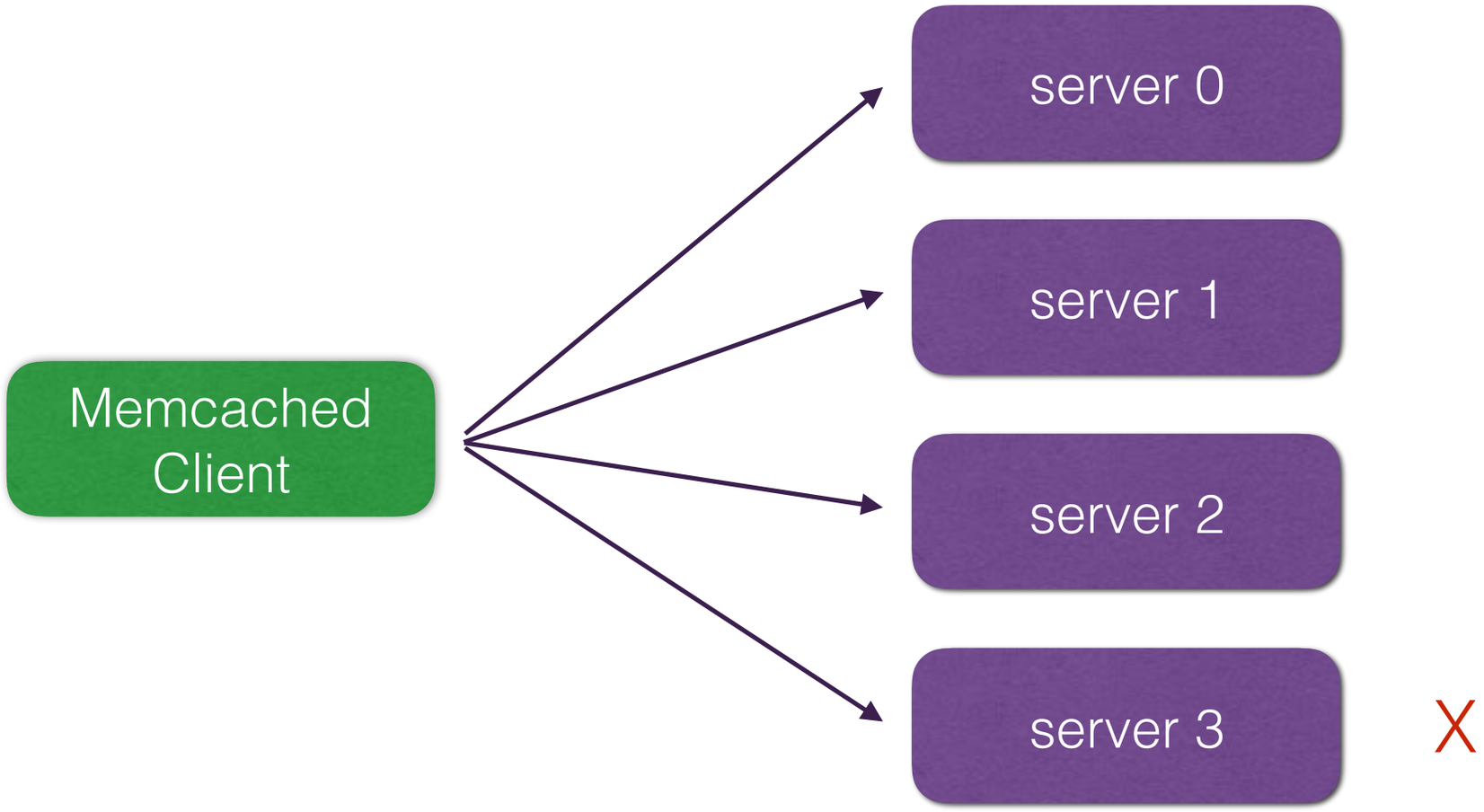
hash("foo") % 4 = server 3

After

7

hash("foo") % 7 = server 0

Remove a server



Before

7

hash("foo") % 4 = server 3

After

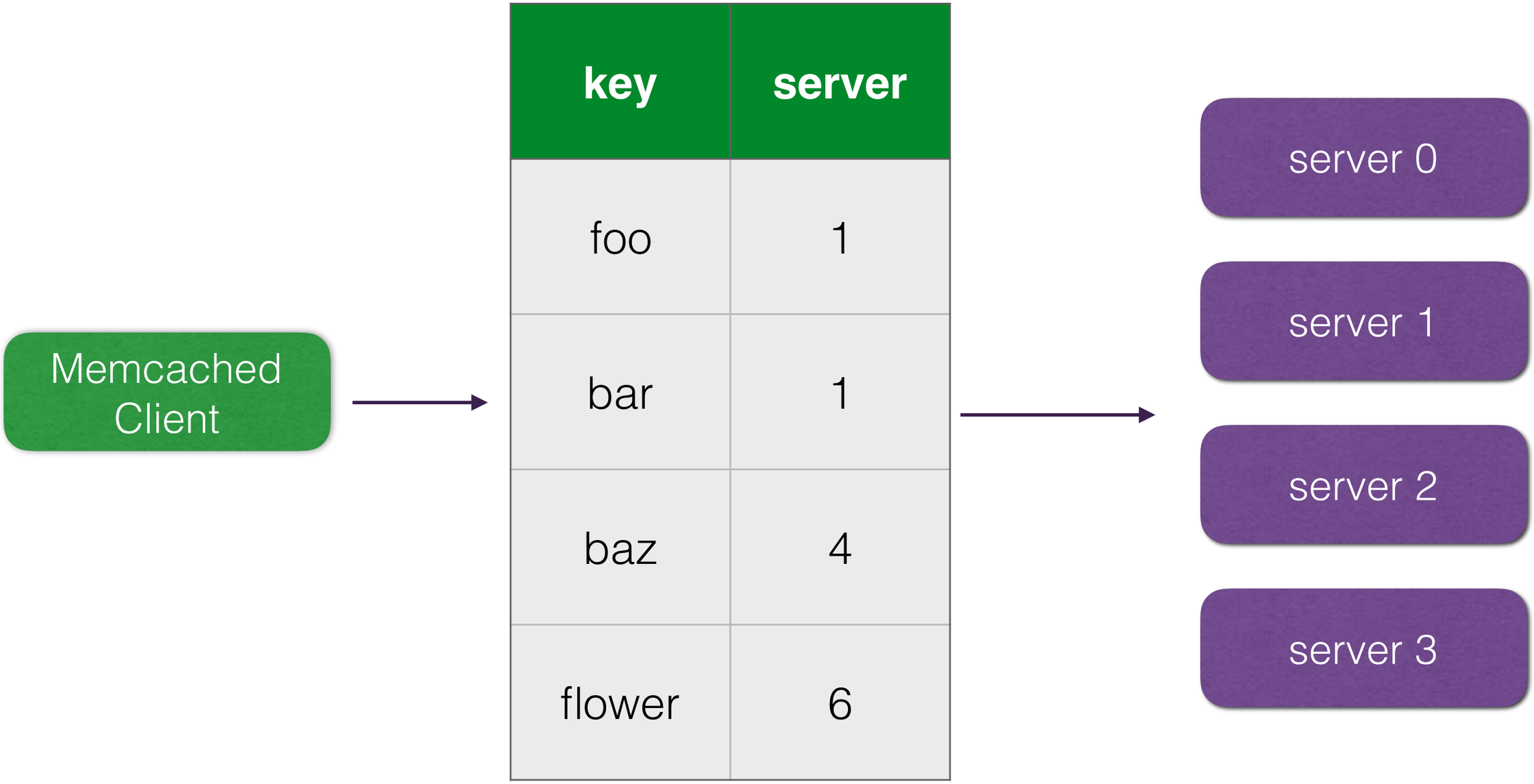
7

hash("foo") % 3 = server 1

The trouble

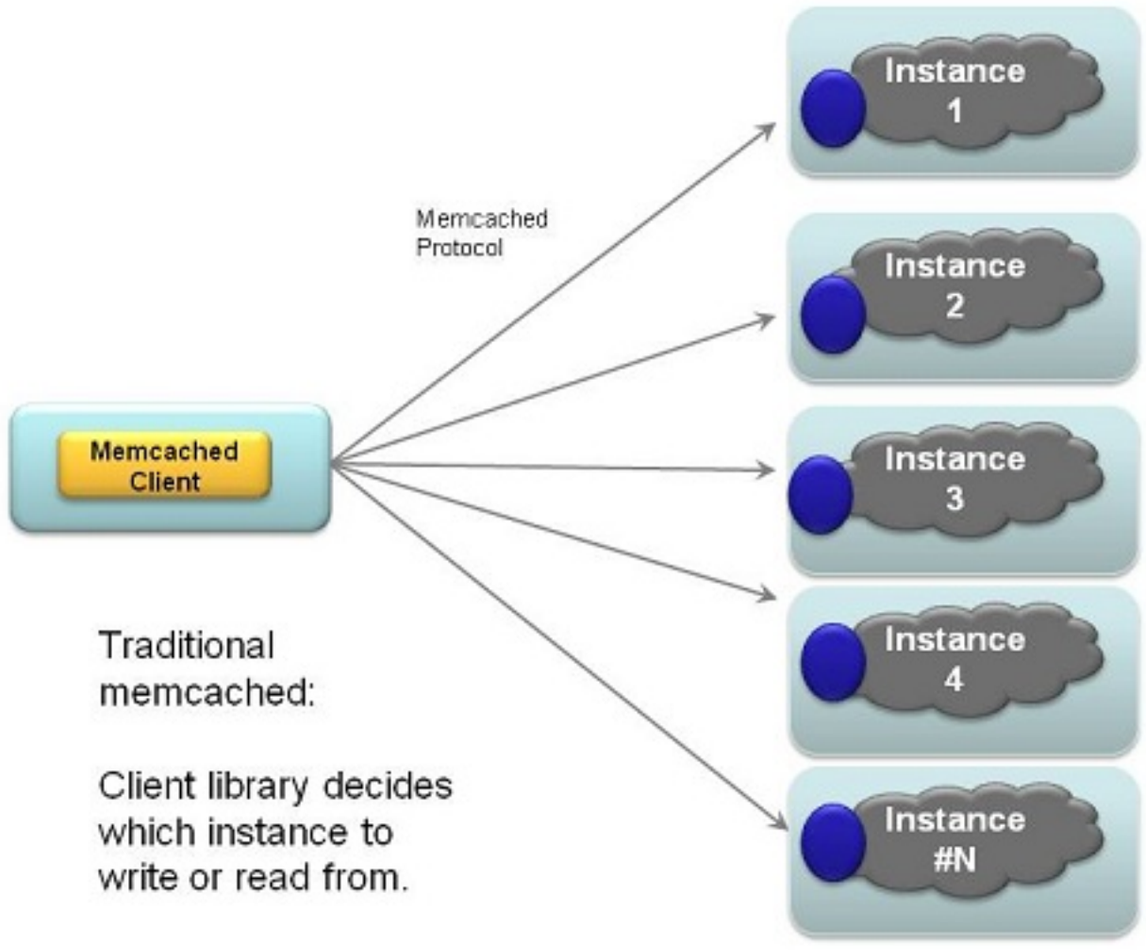
When you change your memcache server count, **almost 100% of all keys** will change server as well.

Assumption 2: big table?



- How to handle mutex?
- How to make this big table scalable?
- How to add server?
- How to remove server?

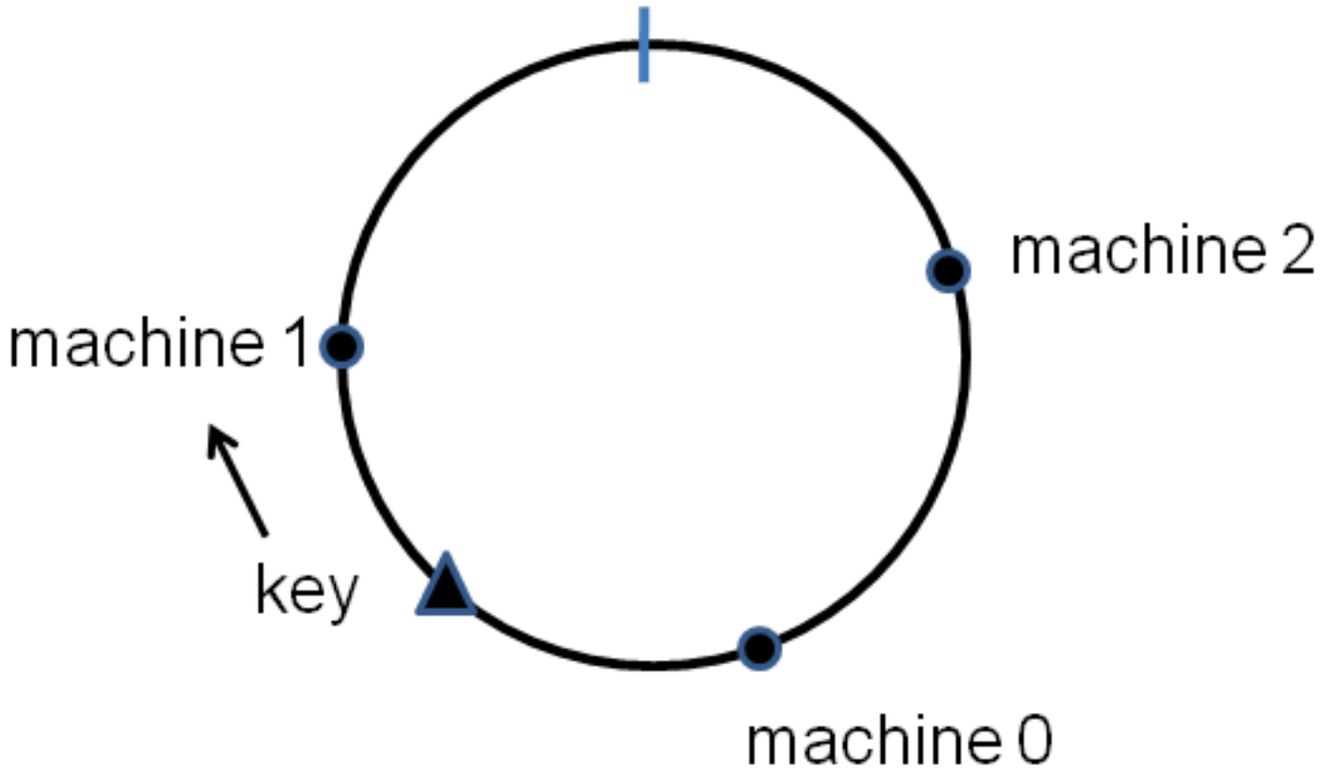
Memcached's Implementation



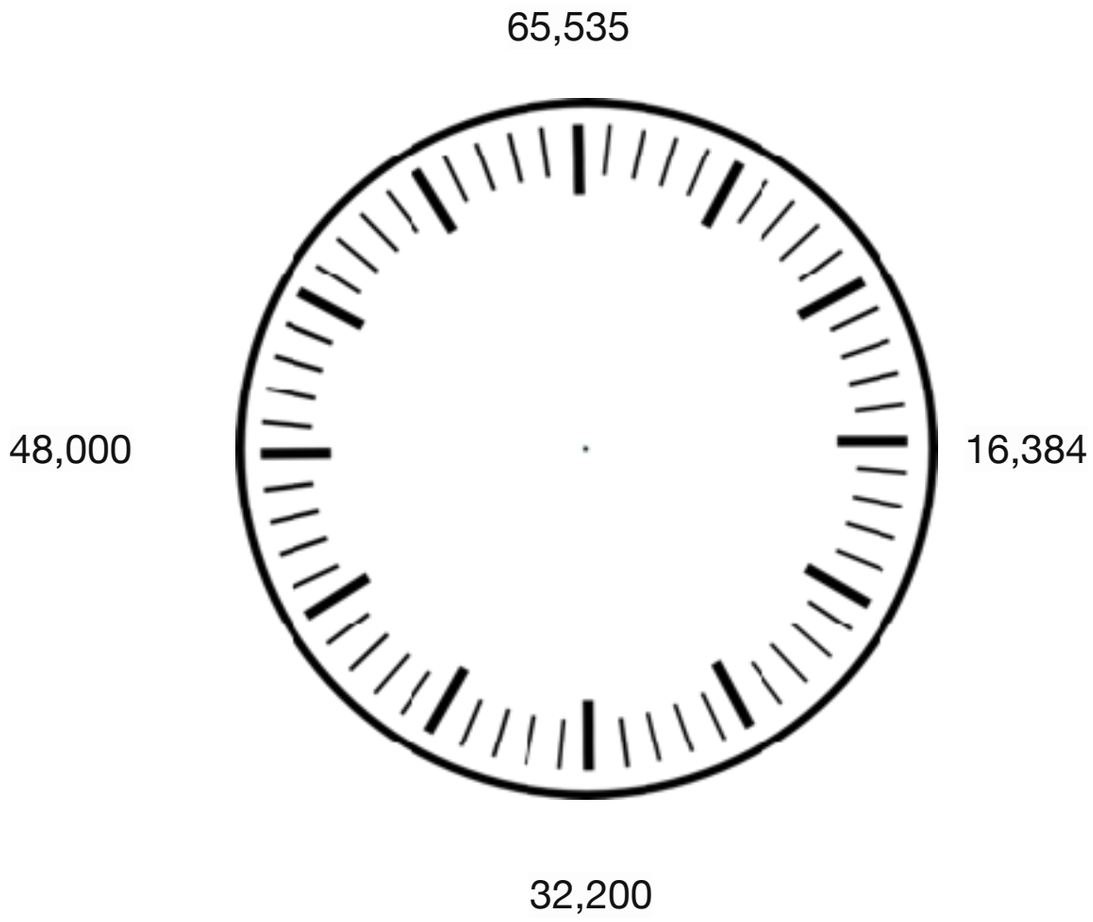
Consistent Hasing

souce code fo dalli

https://github.com/petergoldstein/dalli/blob/fa3d136a16510d4ef47da7cb54cd0eccc



step1: create a clock

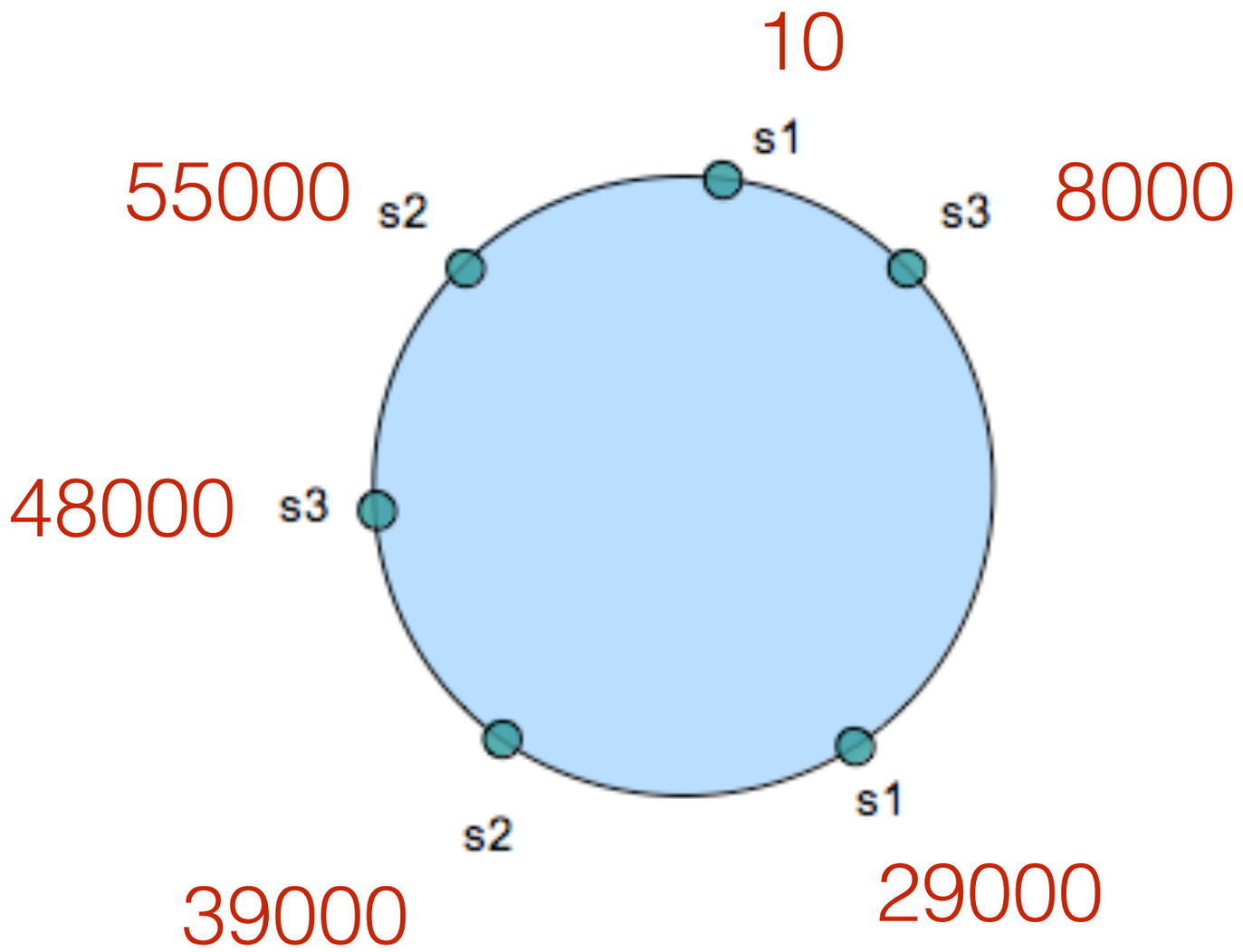


Step2: create 2 dots for each server

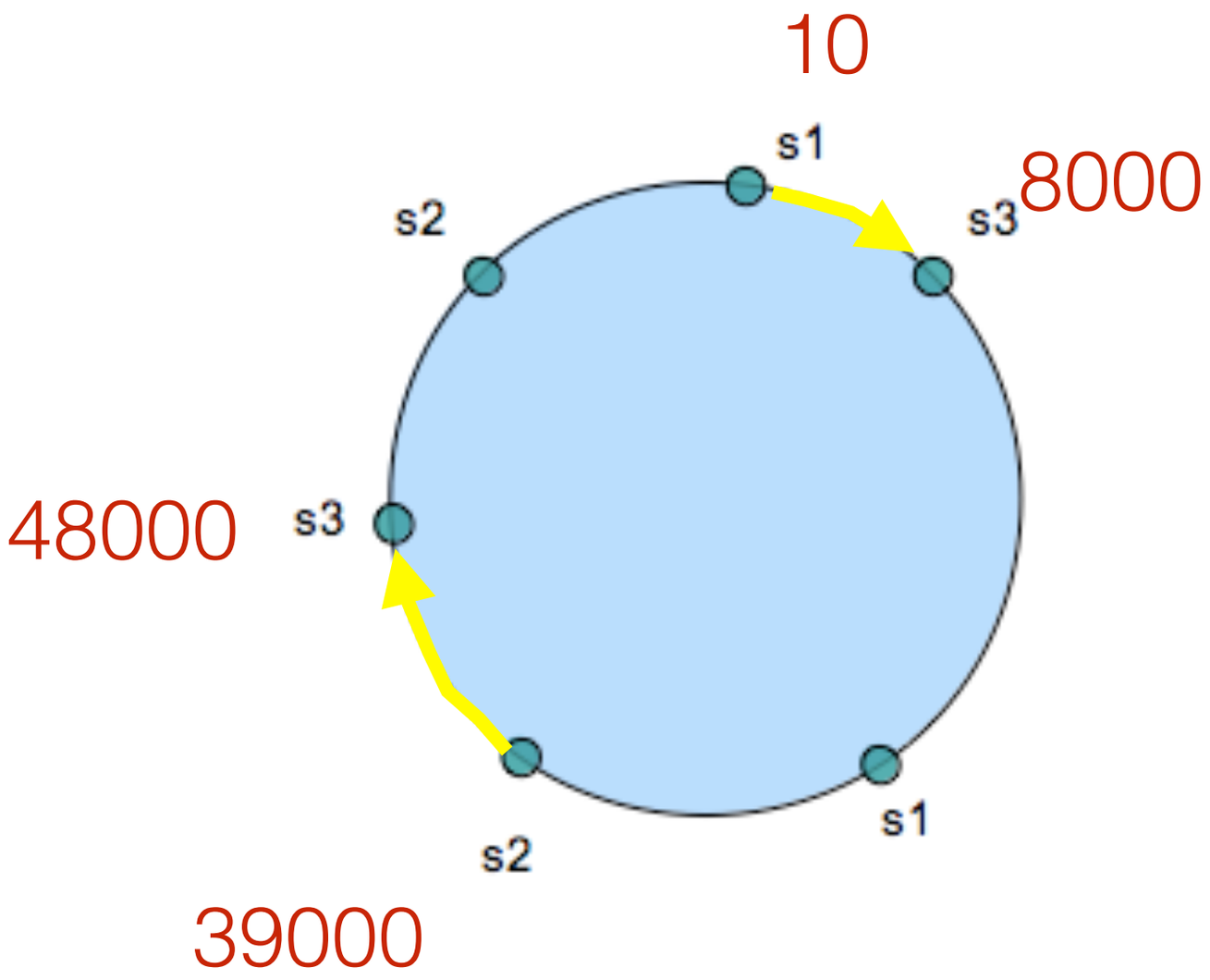
```ruby
servers.each do |server|
  2.times do |idx|
    hash = Digest::SHA1.hexdigest("#{server.name}:#{idx}")
    value = Integer("0x#{hash[0..7]}")
  end
end
```

s1 = {10, 29000}
s2 = {39000, 55000}
s3 = {8000, 48000}

Step2: create 2 buckets for each server



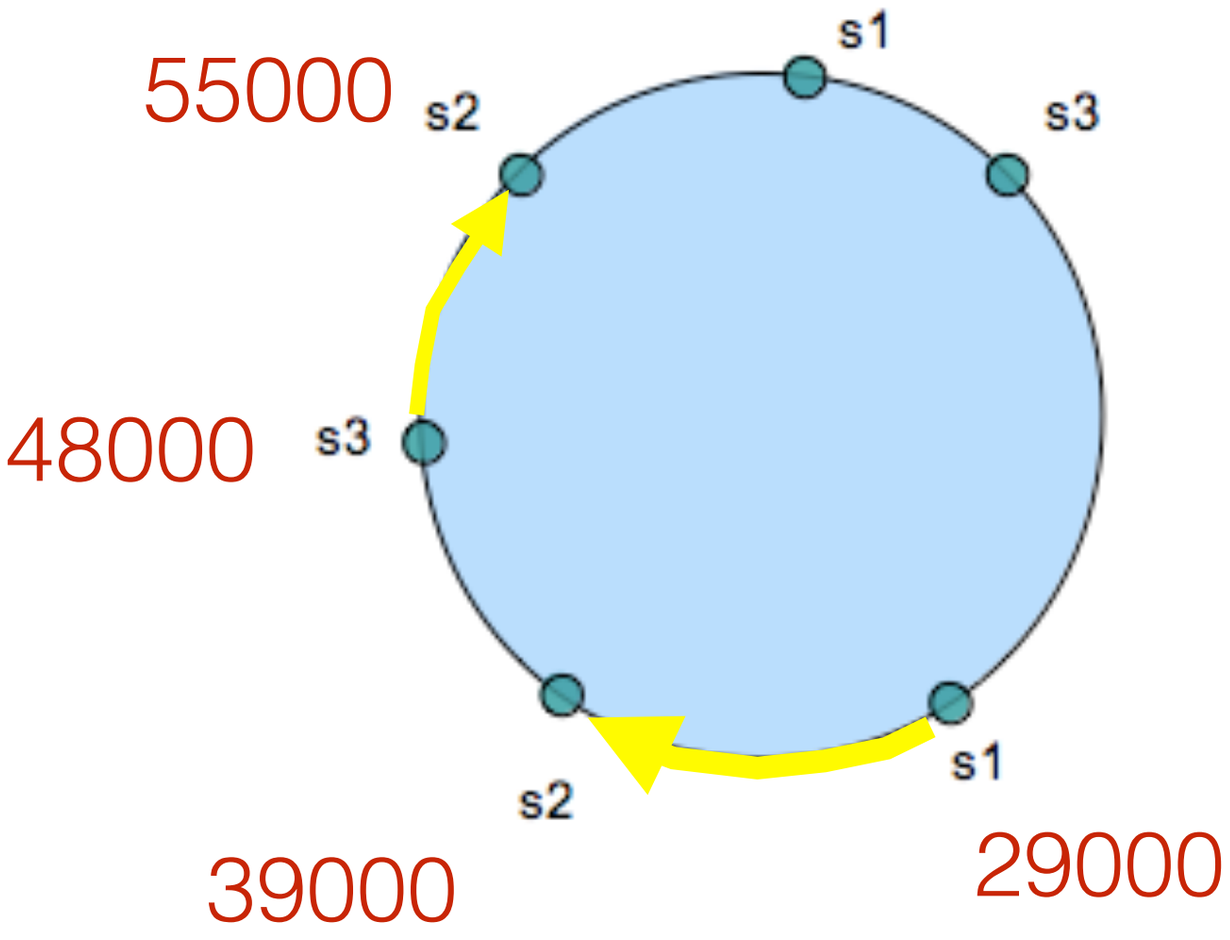
s3 = {8000, 48000}



in charge of

- 10 < x < 8000
- 39000 < x < 48000

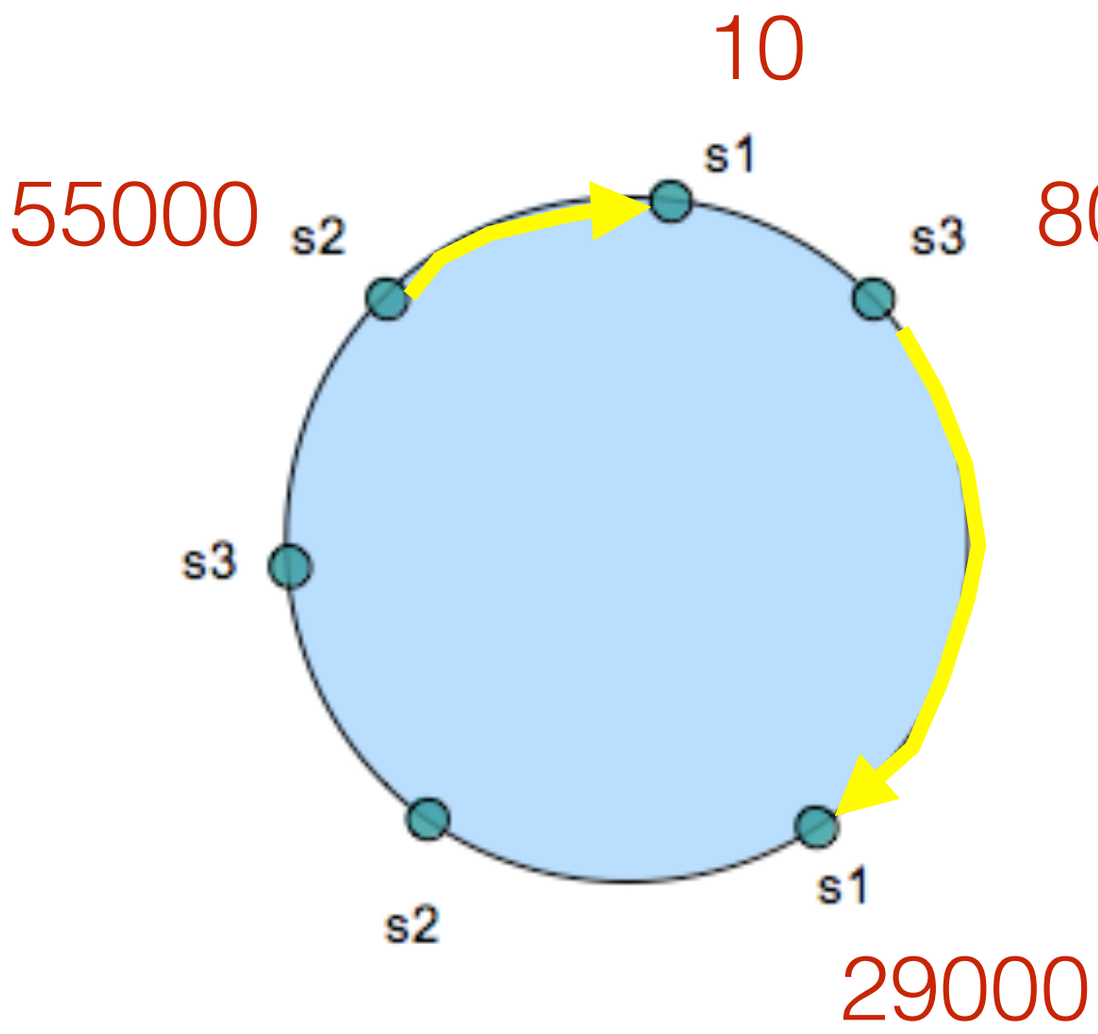
s2 = {39000, 55000}



in charge of

- 29000 < x < 39000
- 48000 < x < 55000

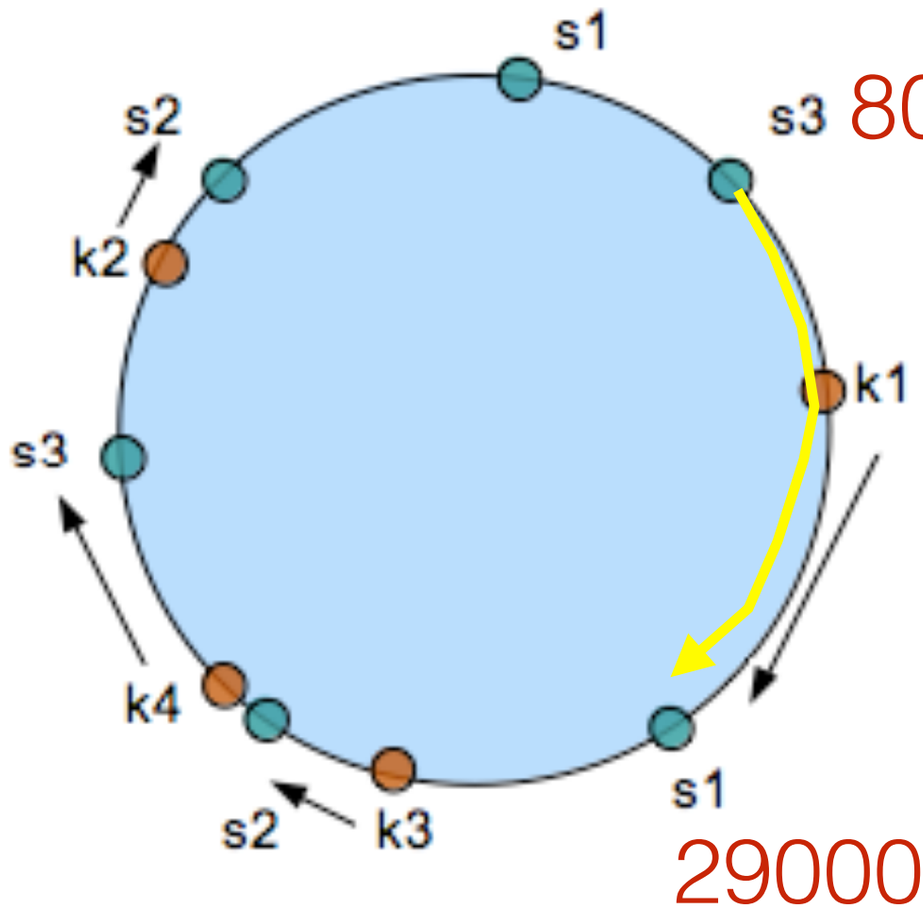
s1 = {10, 29000}



in charge of

- 55000 < x < 65535
- 0 < x < 10
- 8000 < x < 29000

Step2: help k1 to find actual server



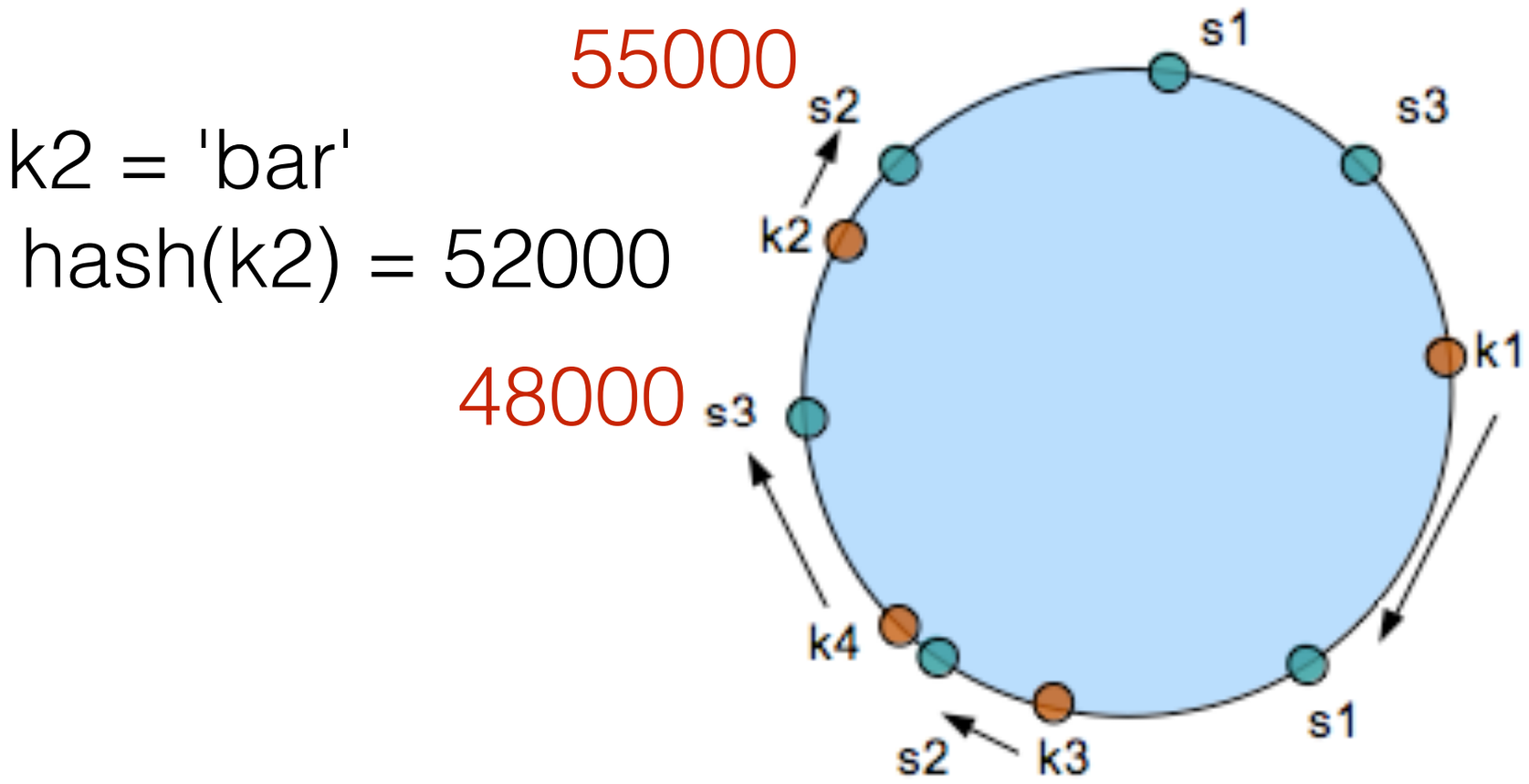
k1 = 'foo'

hash(k1) = 15000

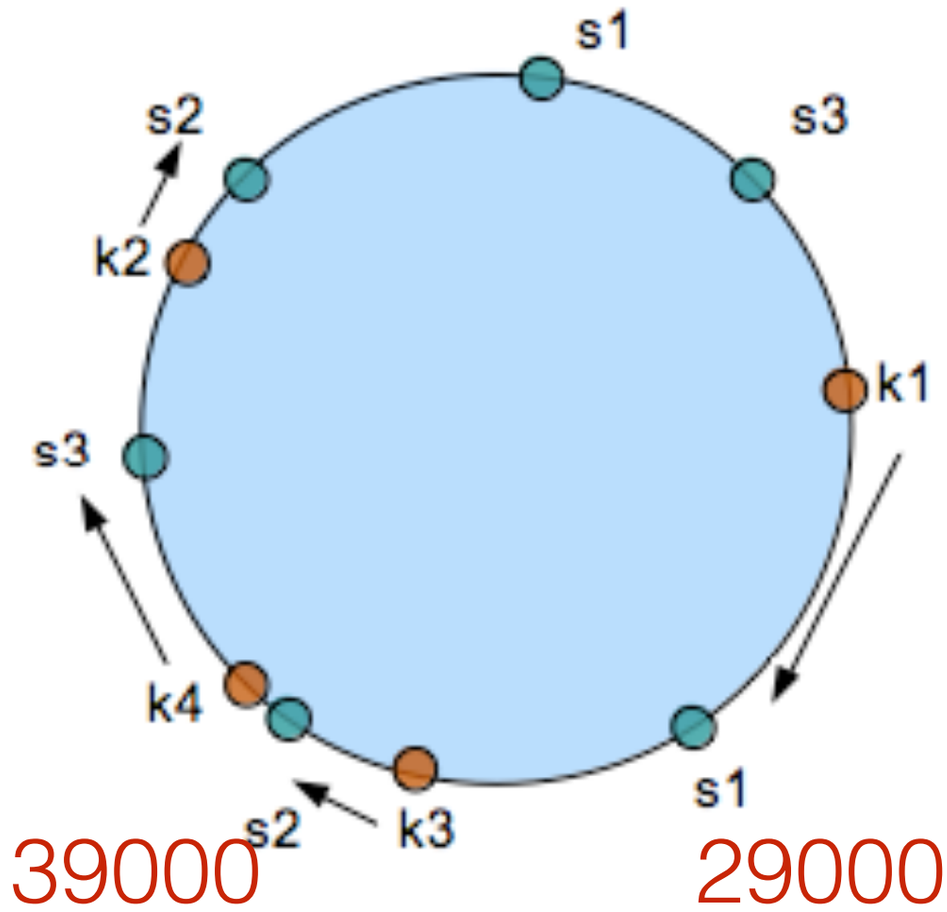
Step2: help k2 to find actual server

k2 = 'bar'

hash(k2) = 52000



Step2: help k3 to find actual server

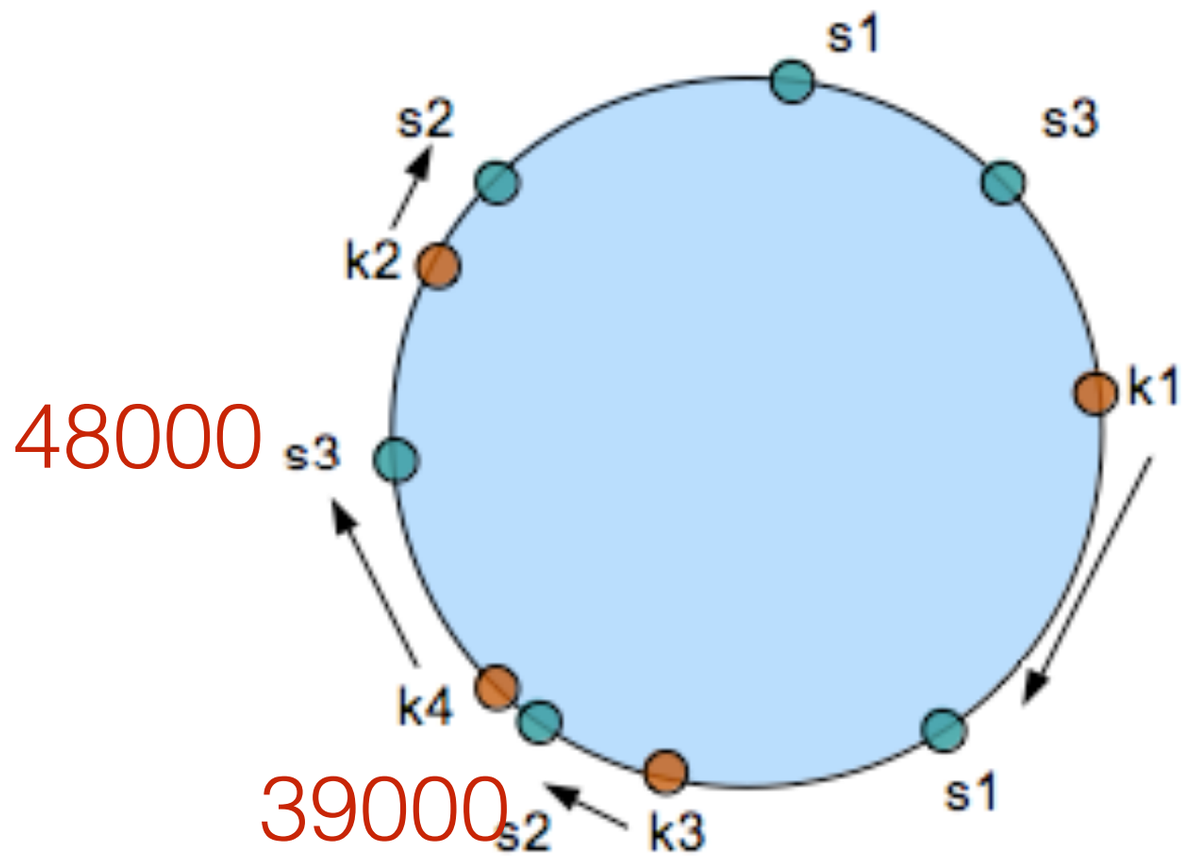


k3 = 'cat'

hash(k3) = 34000

Step2: help k4 to find actual server

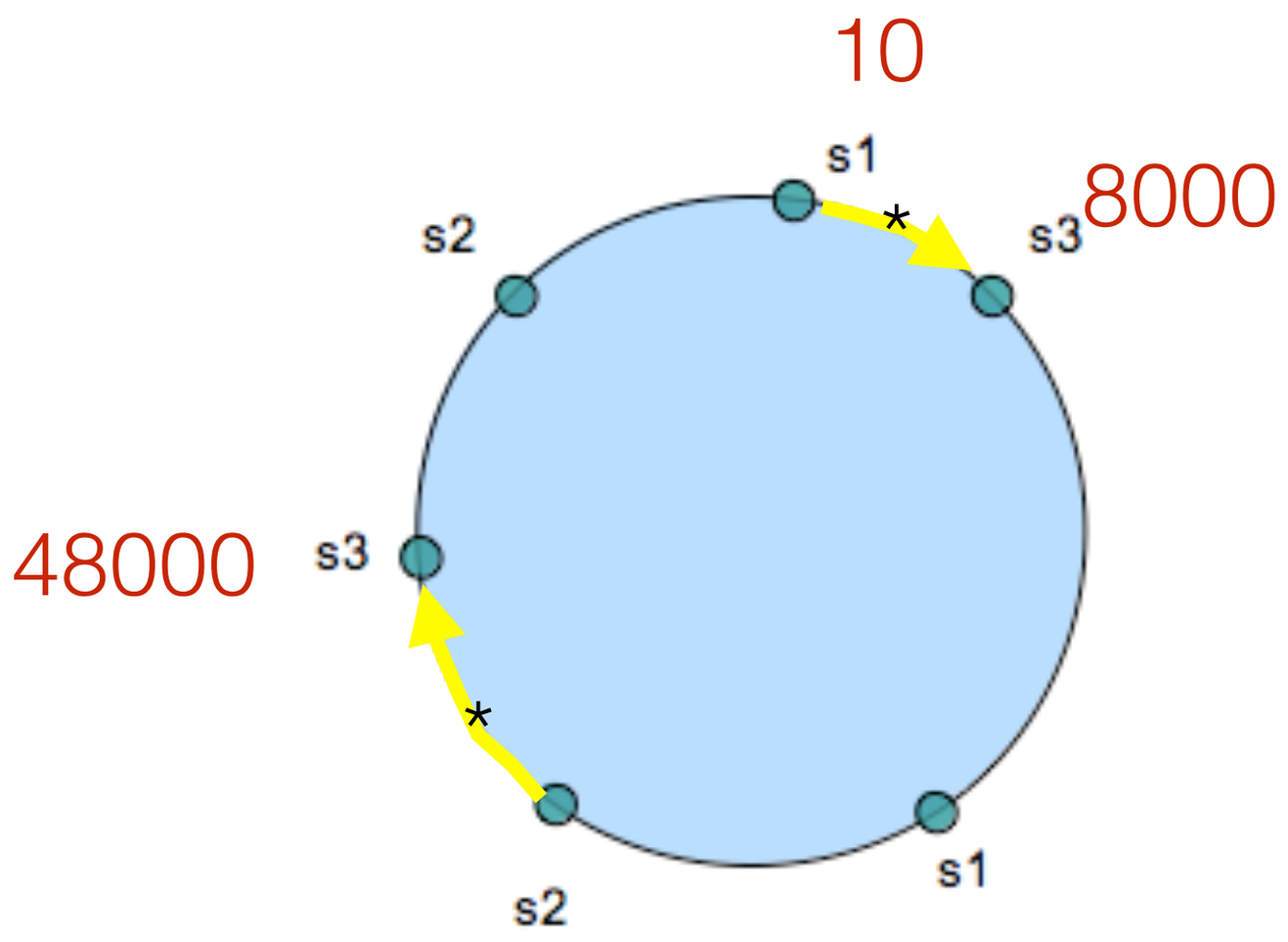
k4 = 'dog'
hash(k4) = 38000

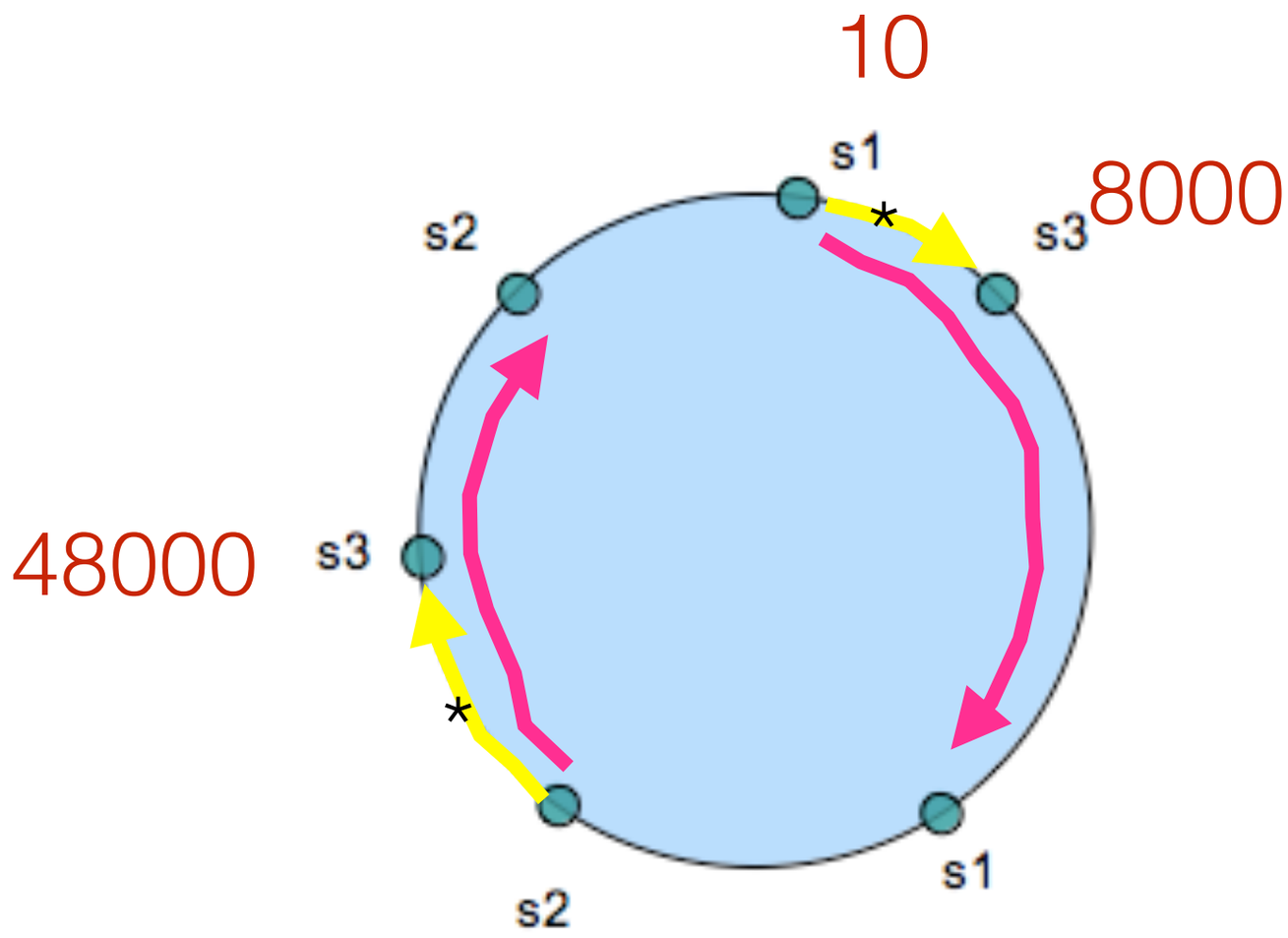


Why is it called consistent hashing?

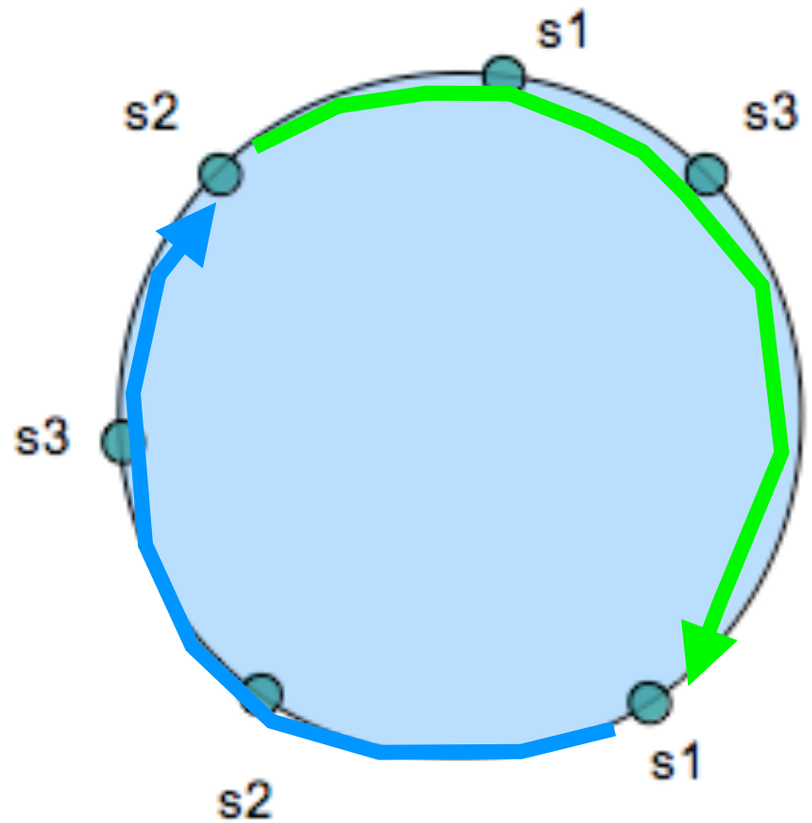
imagine s3 is down

What will happen?




10
s1
8000
s3
s2
48000
s3
s1
s2
39000

Final Distribution



1. s3 is replaced by s2 and s1
2. Cache key stored in s2/s1 is not affected

Memory Allocation

Concepts

- Chunk, to store item.
- Slab Class, to define chunk size.
- Page, to be assigned to slab class.

Chunk, to store item

Pages

divided

Slab Class, to define chunk size

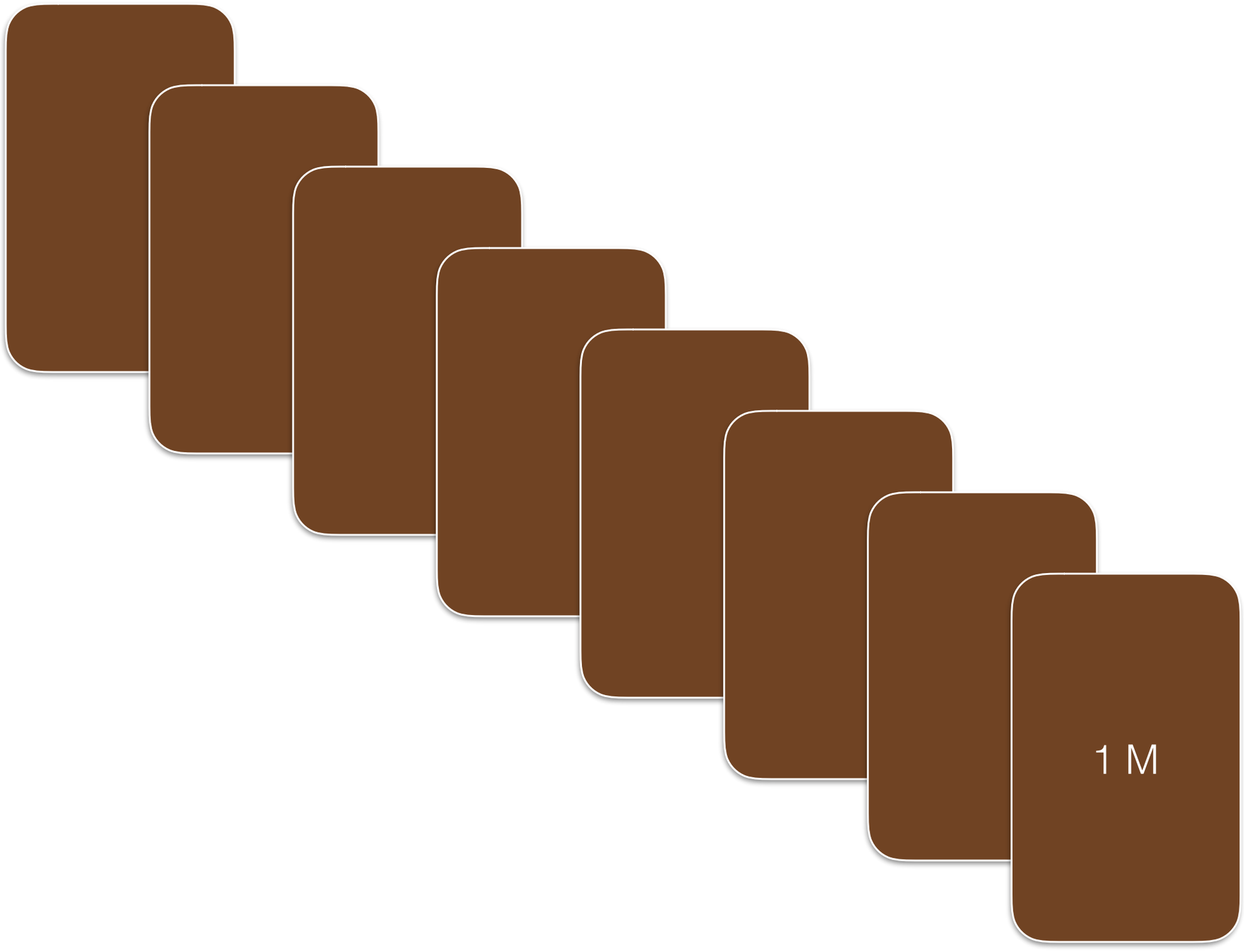
1 M / 31 kb = 33 chunks

1 M / 200 kb = 5 chunks

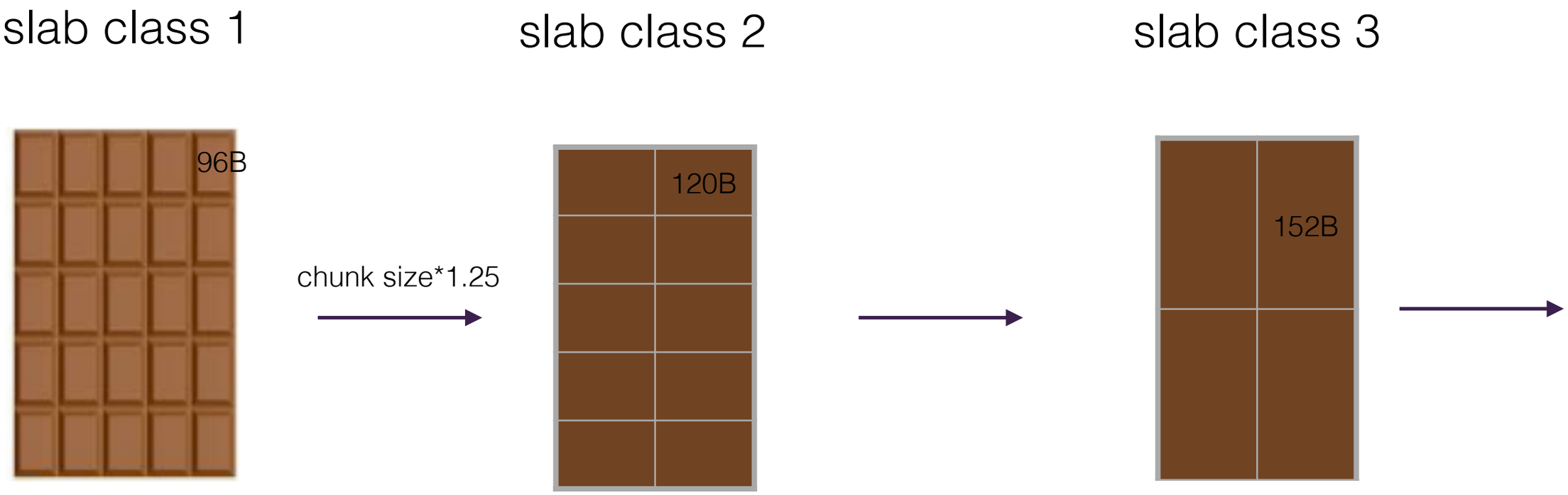
```
>  memcached -vv
slab class   1: chunk size        96 perslab   10922
slab class   2: chunk size       120 perslab    8738
slab class   3: chunk size       152 perslab    6898
slab class   4: chunk size       192 perslab    5461
slab class   5: chunk size       240 perslab    4369
slab class   6: chunk size       304 perslab    3449
slab class   7: chunk size       384 perslab    2730
slab class   8: chunk size       480 perslab    2184
slab class   9: chunk size       600 perslab    1747
slab class  10: chunk size       752 perslab    1394
slab class  11: chunk size       944 perslab    1110
slab class  12: chunk size      1184 perslab     885
slab class  13: chunk size      1480 perslab     708
slab class  14: chunk size      1856 perslab     564
slab class  15: chunk size      2320 perslab     451
slab class  16: chunk size      2904 perslab     361
slab class  17: chunk size      3632 perslab     288
slab class  18: chunk size      4544 perslab     230
slab class  19: chunk size      5680 perslab     184
slab class  20: chunk size      7104 perslab     147
slab class  21: chunk size      8880 perslab     118
slab class  22: chunk size     11104 perslab      94
slab class  23: chunk size     13880 perslab      75
slab class  24: chunk size     17352 perslab      60
slab class  25: chunk size     21696 perslab      48
slab class  26: chunk size     27120 perslab      38
slab class  27: chunk size     33904 perslab      30
slab class  28: chunk size     42384 perslab      24
slab class  29: chunk size     52984 perslab      19
slab class  30: chunk size     66232 perslab      15
slab class  31: chunk size     82792 perslab      12
slab class  32: chunk size    103496 perslab      10
slab class  33: chunk size    129376 perslab       8
slab class  34: chunk size    161720 perslab       6
slab class  35: chunk size    202152 perslab       5
slab class  36: chunk size    252696 perslab       4
slab class  37: chunk size    315872 perslab       3
slab class  38: chunk size    394840 perslab       2
slab class  39: chunk size    493552 perslab       2
slab class  40: chunk size    616944 perslab       1
slab class  41: chunk size    771184 perslab       1
slab class  42: chunk size   1048576 perslab       1
```

How does it allocate memory?

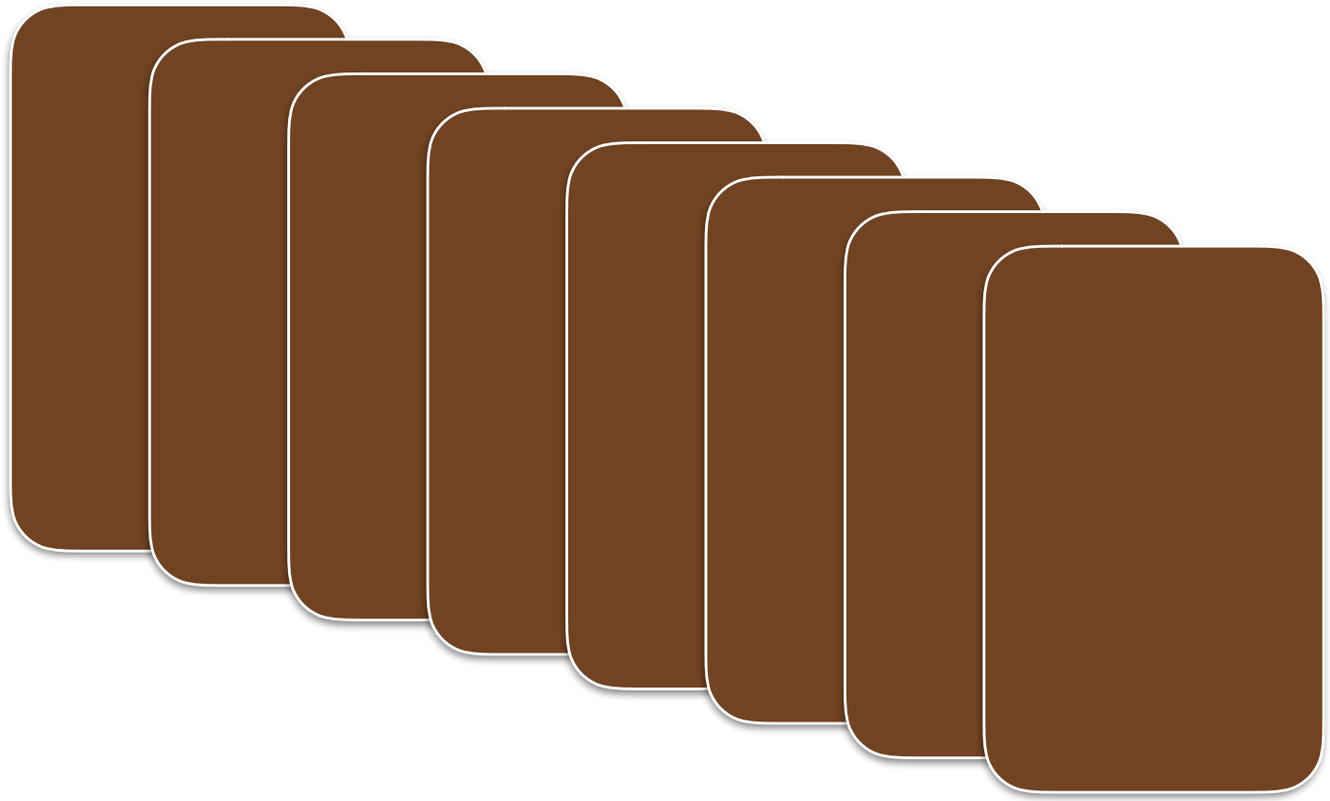
Step 1: Page



Step 2: Slab Class

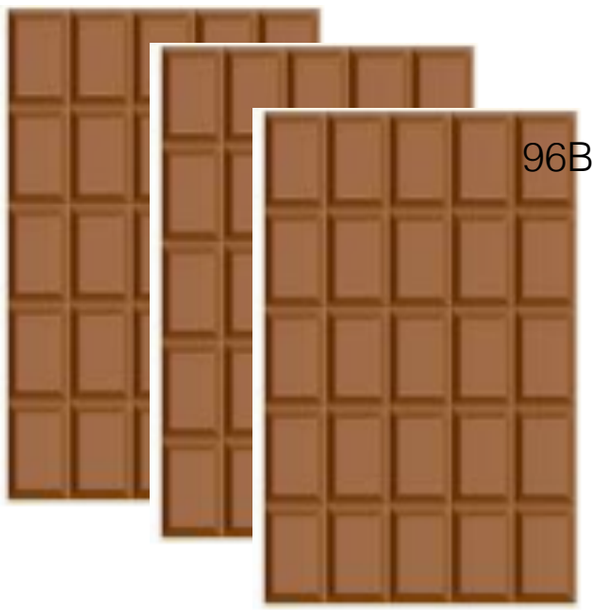


Step 3: Assign page -> chunks

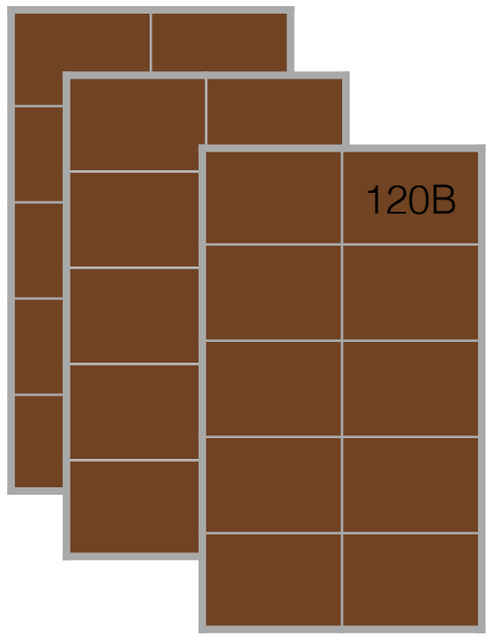


Free pages

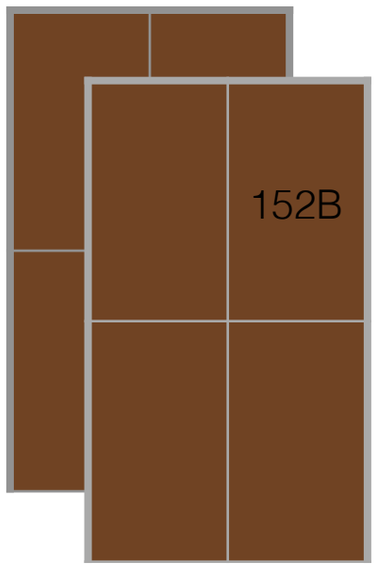
slab class 1



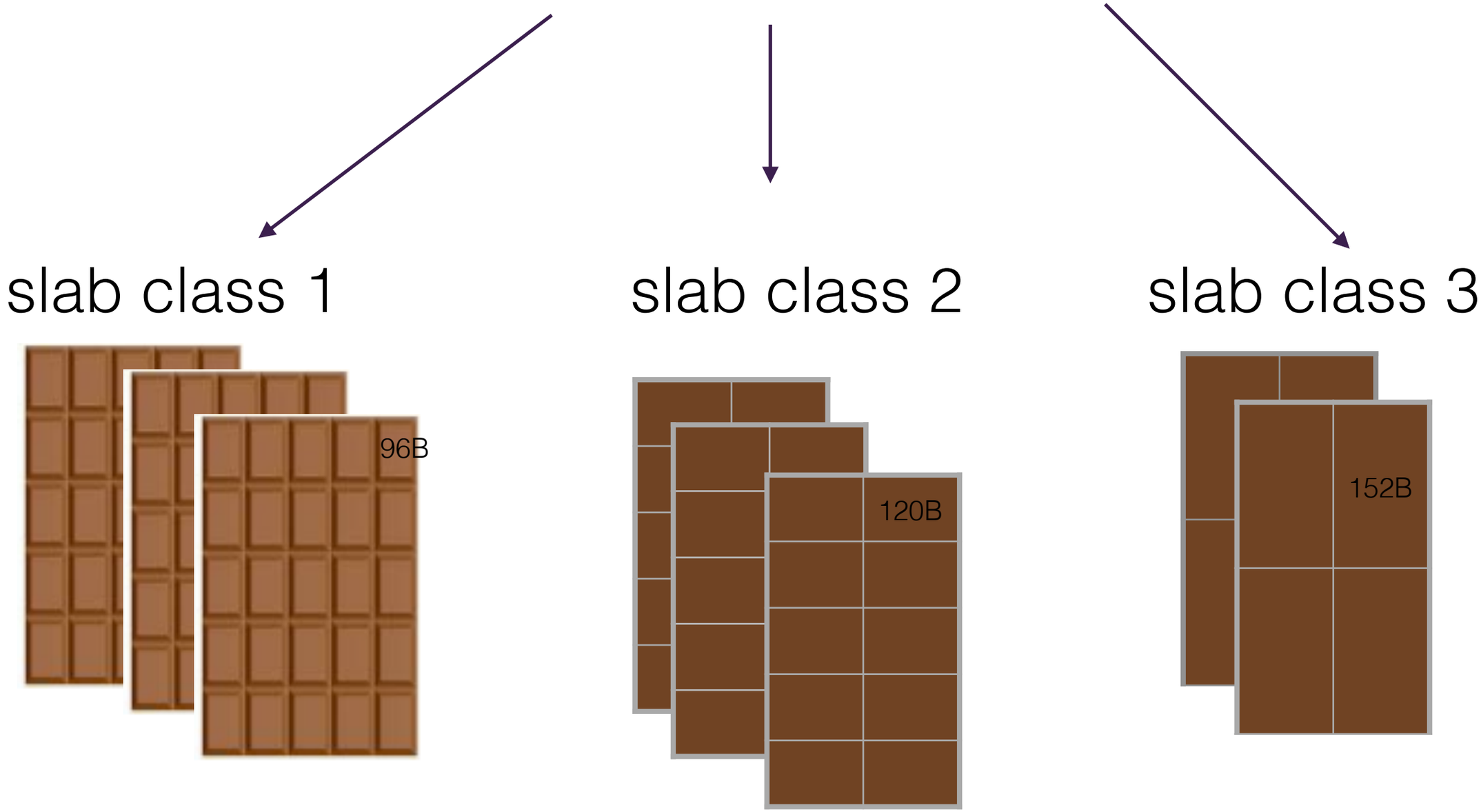
slab class 2



slab class 3



No available pages



This assignment is Permanent. can not be re-assigned.

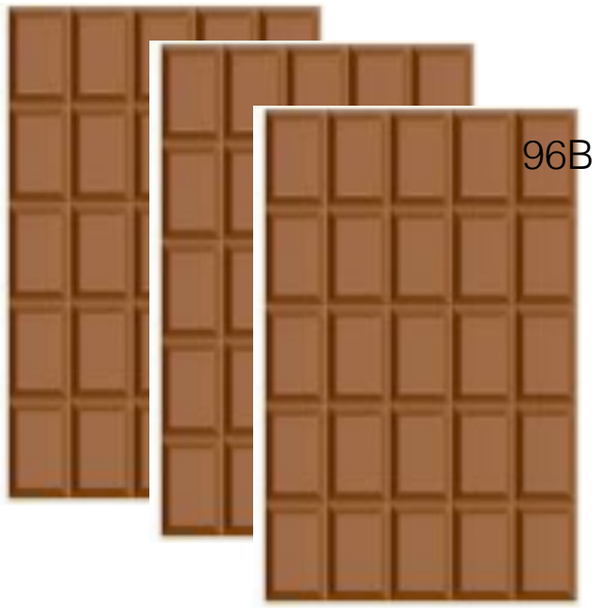
Can pages be re-assigned?

What will happen when there is no free pages?

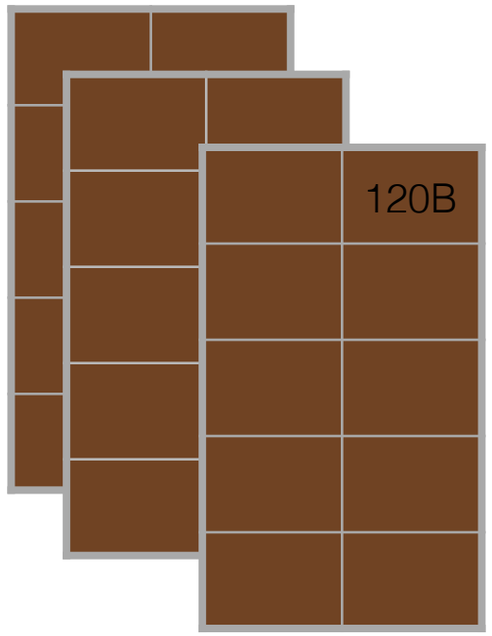
Each slab class has it's own LRU.

Each slab has own LRU

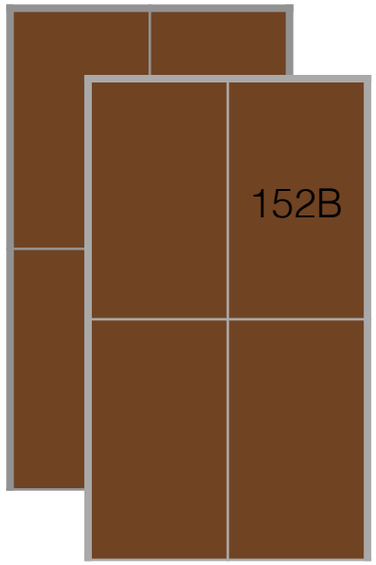
slab class 1



slab class 2



slab class 3



Thank you.

Reference

- Dalli: Consistent Hashing https://goo.gl/80heoh
- What is BigO? https://goo.gl/J5L6QX
- Memcache Internals https://goo.gl/weGsIe
- Memcached for Dummies https://goo.gl/NNSSmi
- Mike Perham: Slabs, Pages, Chunks and Memcached https://goo.gl/oapOjI
- Pycon 2014: Cache me if you can https://goo.gl/ry471l