How to design TinyURL?

What is the problem?

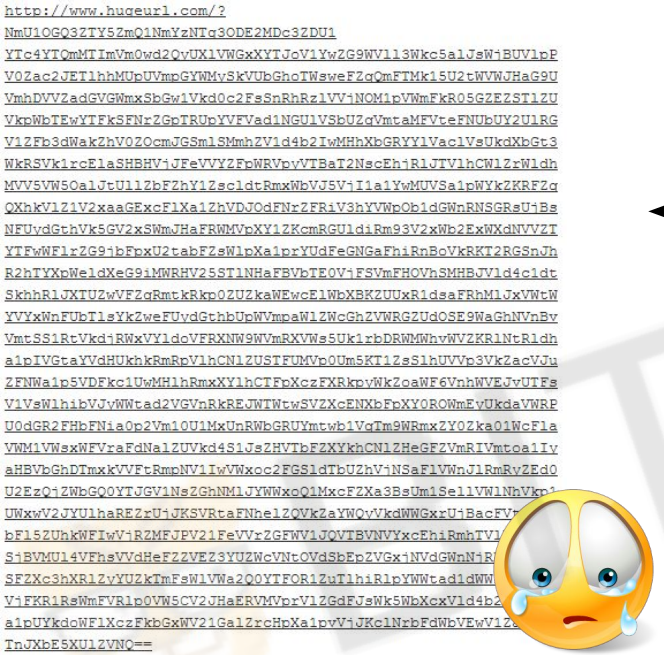
http://www.hugeurl.com/?
NmU1OGQ3ZTY5ZmQ1NmYzNTg3ODE2MDc3ZDU1
YTc4YTQmMTImVm0wd2QyUXlVWGxXYTJoV1YwZG9WVll3Wkc5alJsWjBUVlpP
V0Zac2JETlhhMUpUVmpGYWMySkVUbGhoTWsweFZqQmFTMk15U2tWVWJHaG9U
VmhDVVZadGVGWmxSbGw1Vkd0c2FsSnRhRzlVVjNOM1pVWmFkR05GZEZSTlZU
VkpWbTEwYTFkSFNrZGpTRUpYVFVad1NGU1VSbUZqVmtaMFVteFNUbUY2UlRG
V1ZFb3dWakZhV0ZOcmJGSmlSMmhZV1d4b2IwMHhXbGRYYlVaclVsUkdXbGt3
WkRSVk1rcElaSHBHVjJFeVVYZFpWRVpyVTBaT2NscEhjRlJTVlhCWlZrWldh
MVV5VW5OalJtUllZbFZhY1ZscldtRmxWbVJ5VjI1a1YwMUVSa1pWYkZKRFZq
QXhkVlZ1V2xaaGExcFlXa1ZhVDJOdFNrZFRiV3hYVWpOb1dGWnRNSGRsUjBs
NFUydGthVk5GV2xSWmJHaFRWMVpXY1ZKcmRGUldiRm93V2xWb2ExWXdNVVZT
YTFwWFlrZG9jbFpxU2tabFZsWlpXa1prYUdFeGNGaFhiRnBoVkRKT2RGSnJh
R2hTYXpWeldXeG9iMWRHV25STlNHaFBVbTE0VjFSVmFHOVhSMHBJVld4c1dt
SkhhRlJXTUZwVFZqRmtkRkp0ZUZkaWEwcElWbXBKZUUxR1dsaFRhMlJxVWtW
YVYxWnFUbTlsYkZweFUydGthbUpWVmpaWlZWcGhZVWRGZUdOSE9WaGhNVnBv
VmtSS1RtVkdjRWxVYldoVFRXNW9WVmRXVWs5Uk1rbDRWMWhvWVZKRlNtRldh
a1pIVGtaYVdHUkhkRmRpVlhCNlZUSTFUMVp0Um5KT1ZsSlhUVVp3VkZacVJu
ZFNWa1p5VDFkc1UwMHlhRmxXYlhCTFpXczFXRkpyWkZoaWF6VnhWVEJvUTFs
V1VsWlhibVJyWWtad2VGVnRkREJWTWtwSVZXcENXbFpXY0ROWmEyUkdaVWRP
U0dGR2FHbFNia0p2Vm10U1MxUnRWbGRUYmtwb1VqTm9WRmxZY0Zka01WcFla
VWM1VWsxWFVraFdNalZUVkd4S1JsZHVTbFZXYkhCNlZHeGFZVmRIVmtoa1Iy
aHBVbGhDTmxkVVFtRmpNV1IwVWxoc2FGSldTbUZhVjNSaFlVWnJlRmRyZEd0
U2EzQjZWbGQ0YTJGV1NsZGhNMlJYWWxoQ1MxcFZXa3BsUm1SellVWlNhVkp1
UWxwV2JYUlhaREZrUjJKSVRtaFNhelZQVkZaYWQyVkdWWGxrUjBacFVteHdl
bFl5ZUhkWFIwVjRZMFJPV21FeVVrZGFWV1JQVTBVNVYxcEhiRmhTVlhCS1Zq
SjBVMU14VFhsVVdHeFZZVEZ3YUZWcVNtOVdSbEpZVGxjNVdGWnNjREJVVlZK
SFZXc3hXRlZyYUZkTmFsWlVWa2Q0YTFOR1ZuTlhiRlpYWWtad1dWWkhlR0Za
VjFKR1RsWmFVRlp0VW5CV2JHaERVMVprVlZGdFJsWk5WbXcxVld4b2MxWnNX
a1pUYkdoWFlXczFkbGxWV21GalZrcHpXa1pvVjJKclNrbFdWbVEwV1ZaWmVG
TnJXbE5XUlZVNQ==

Such a long url is common!

崩溃的瞬间：

- 在文章中引用链接……10行结束了
- 在报纸上发现这个链接想浏览一下
- 复制粘贴到浏览器……落下几行
- 在Twitter中share该链接……140个字符根本不够用
- ……

How to solve it?



http://tinyURL/a1d3isf

- TinyURL is a URL shortening web service, which provides short aliases for redirection of long URLS.
- First launched by Gilbertson in January 2002

Review: SNAKE

- Scenario: case/interface
- Necessary: constrain/hypothesis
- Application: service/algorithm
- Kilobit: data
- Evolve

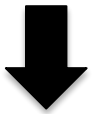
Scenario

Question: How many services should TinyURL provide?

1. One
2. Two
3. Three
4. Four

It requires at least **TWO** basic services:

- Given a long url, it generates a **unique** short url
- Given a short url, it is able to retrieve the corresponding long url, thus,sends http request to the correct address



short_url insert(long_url)
long_url lookup(short url)

Review: Performance v.s. Scalability

Question: Which describes performance? Which describes scalability?

1. How long does it take to respond to a request?

2. How many simultaneous users can you support?

3. Measured by the simultaneous users to support

4. Measured by delay from sending requests to displaying response

5. Influenced by code quality, which database to use, database schema, web framework, etc.

6. Influenced by performance per machine, how many machines to use, and how are they coordinated to work, etc.

Performance:

? ? ?

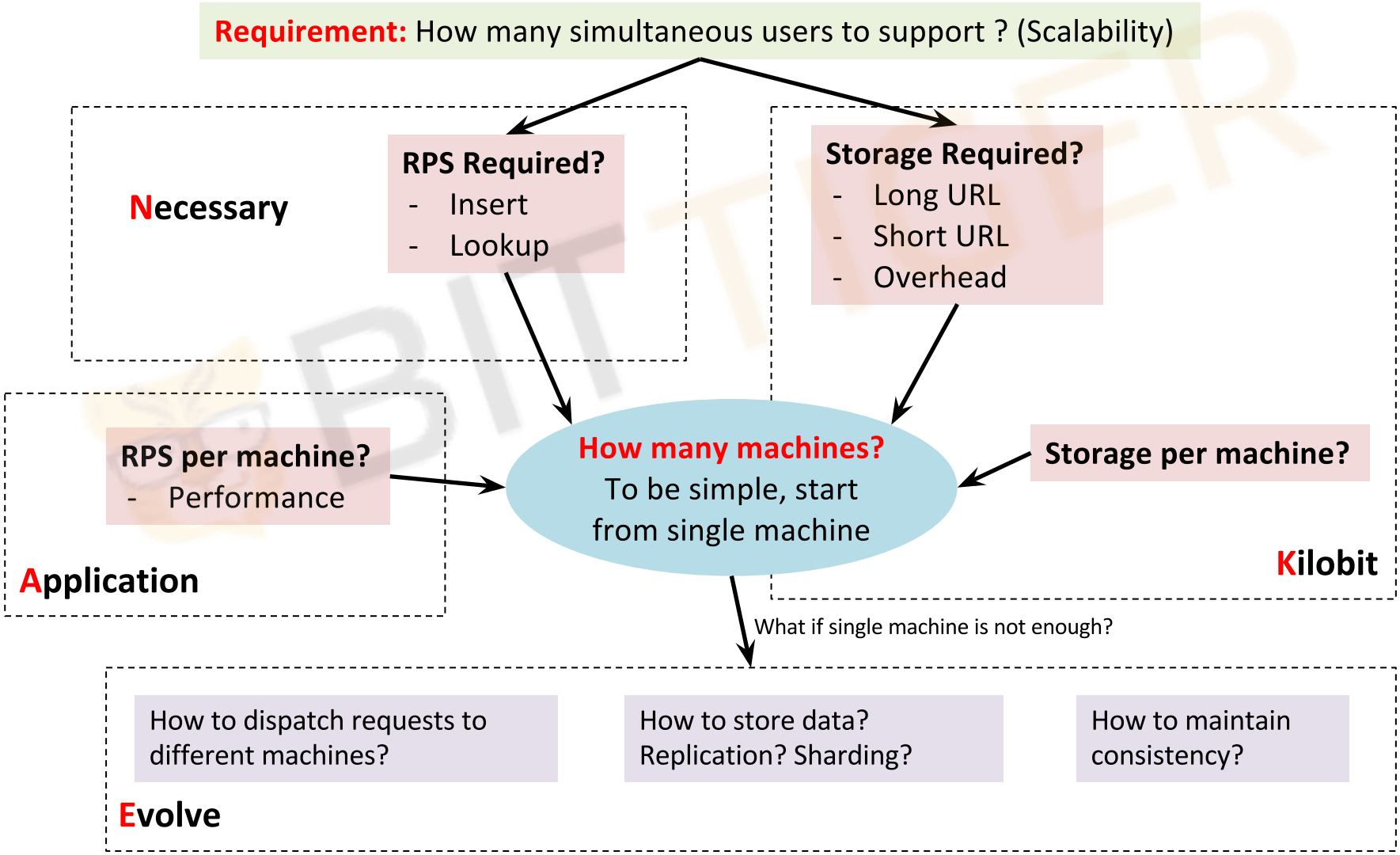
1 4 5

Scalability:

? ? ?

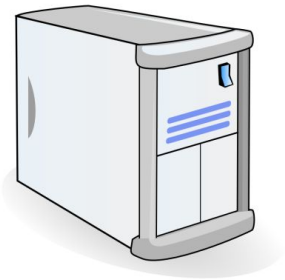
2 3 6

Decision Process



Necessary

- Requirement: 1 Million Daily Active Users
- Insert Operations
  - 1,000,000 * 1% (function usage) * 10 (function frequency) = 100,000
  - RPS: 100,000 / 86400 = 1.2
- Lookup Operations
  - 1,000,000 * 100% (function usage) * 3 (function frequency) = 3,000,000
  - RPS: 3,000,000 / 86400 = 35
- Peak Estimation
  - Assume Peak Traffic = 5 * Average Daily Traffic
  - Insertion Peak RPS = 1.2 * 5 = 6
  - Lookup Peak RPS = 35 * 5 = 175



Single Machine

Application – Architecture

Naïve Thought:

short_url insert(long_url)
long_url lookup(short url)

HashTable/HashMap

To be simple, first, assume In-memory storage:

```
Class TinyURL {
        map <longURL, shortURL> LongToShortMap;
        map<shortURL, longURL> ShortToLongMap;

        shortURL insert(longURL) {
                if LongToShortMap not containsKey longURL:
                        generate shortURL;
                        put <longURL, shortURL> into LongToShortMap;
                        put <shortURL, longURL> into ShortToLongMap;
                Return LongToShortMap.get(longURL);
        }

        longURL lookup(shortURL) {
                return ShortToLongMap.get(shortURL);
        }
}
```

How to generate the shortURL?

How to define the hash function from longURL to shortURL?

Application – Traditional Hash

Question: How are you familiar with Hash Function?

1. Know a lot
2. Learn a little
3. Have no idea

http://www.construx.com/Blogs/10x_Software_Development/?id=15082

Hash Method	Hash Result	Length
Adler32	399014e3	8 (32bit)
CRC32	78aa9d1a	8 (32bit)
MD2	286c50c2db4fcad77adb4edeb3a937b2	32 (128bit)
MD4	387ac3f6aae7956c4fab176271bb4518	32 (128bit)
MD5	f061a171dfc30635462850684f98b886	32 (128bit)
SHA-1	3c93b6d332091b2970fb660d644d0ba3d756e322	40 (160bit)

Each Hexadecimal digit represents four binary digits

Application – Traditional Hash

Assume we use **CRC32**

```
shortURL GenerateShortURL(longURL) {
        return CRC32(longURL)
}
```

- **How many different URL can it support?**

  2^32 = 4,294,967,296 ~ 4 billion regardless of storage capacity

- **How many attempts might cause a collision?**

  According to birthday paradox,P=50%, 77,000; P=75%, 110,000

Basically you can expect to see the first collision after hashing 2^(n/2) items, where n is number of bits. Say, 32bit, 2^16 = 65536

Application – Traditional Hash

What are the pros and cons?

PROS

1. No need to write addition hash function, easy to implement.
2. Are randomly distributed
3. Support url clean

CONS

Problem	Possible Solution
Not short enough (At least 4 bytes)	Only retrieve first 4-5 digit of the hash result
Collision cannot be avoided	Use (long_url + timestamp) as hash argument,if conflict happens,try again (timestamp changes) -> multiple attempts, highly possible conflicts when data is big
Hash computation costs a lot	

Feasible, but not efficient enough!

Application – Base10

Break Assumption: Do we need to generate direct relationship between long_url and short_url, that is, String to String?

What we want to do: Given a long_url, in any way, find a shorter sth, i.e.

- Keep only one-to-one relationship (no conflict)
- Easy to find the shorter sth when we want to insert another long_url

AUTO-INCREMENTAL INT -> Index (Size)

LongToShortMap

longURL	shortURL
Bittiger.io	0
www.inoreader.com/all_articles	1
http://www.construx.com/Blogs/10x_Software_Development/?id=15082	2
……	3

ShortToLongMap

shortURL	longURL
0	Bittiger.io
1	www.inoreader.com/all_articles
2	http://www.construx.com/Blogs/10x_Software_Development/?id=15082
3	……

Application – Base10

Assume we use **Size of map**

```
shortURL GenerateShortURL() {
        return ShortToLongMap.size();
}
```

- **How many URL can it support?**

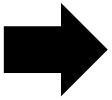
  2^31 - 1 = 2,147,483,647 ~ 2 billion in in-memory map
  basically as big as the capacity in database
  (if consider multiple machines)

- **How many attempts might cause a collision?**

  Within the capacity, no collision.

- **How long will be the url?**

  NumOfURL = 10^Length
  Length = Log_{10}(NumOfURL)



BASE What is important!

Application – Base62

Use more representations in one digit: a-z, A-Z, 0-9

```
shortURL GenerateShortURL() {
        return ConvertTo62(ShortToLong.size());
}

shortURL ConverTo62(mapSize) {
        char Encode[62] = {'0', …, '9', 'a', ..., 'z', 'A', ..., 'Z'};
        shortURL = ""
        while (mapSize > 0) {
                shortURL = Encode[mapSize % 62] + shortURL;
                mapSize /= 62;
        }
        return shortURL;
}
```

- **How many URL can it support?**

  Same as Base10

- **How many attempts might cause a collision?**

  Within the capacity, no collision.

- **How long will be the url?**

  NumOfURL = 62^Length

  Length = Log_{62}(NumOfURL)

Let’s see the Difference

Assume average 100,000 url are inserted each day

	Base10	Base62
Year URL	36,500,000	36,500,000
Usable characters	[0-9] = 10	[0-9a-zA-Z] = 62
Encoding length	$Log_{10}(36,500,000) \sim 8$	$Log_{62}(36,500,000) \sim 5$
Example	Goo.gl/36500000	Goo.gl/2t9jG

Application – Base62

What are the pros and cons?

PROS

1. Short url -> save space
2. Simple computation
3. No conflits

CONS

1. No support for url clean

Application – In Database

We need a two-way lookup:

long_url -> short_url (index)

short_url (index) -> long_url

A table is enough (auto-increment id, long_url):

- Auto-increment id is index as default
- Establish another index on long_url

INSERT

LongURL → Search long_url

Long_url	id
http://bittiger.io	1
http://google.com	2
…	3

If contains such long_url

return convertBase10toBase62(id)

If not contains such long_url

Add the long_url to the end

return convertBase10toBase62(id)

LOOKUP

ShortURL → convertBase62toBase10 → id in database → Search id

id	long_url
1	http://bittiger.io
2	http://google.com
3	…

return long_url

Kilobit

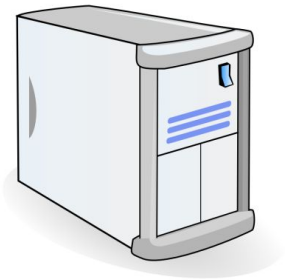
Average size of longURL = 100 bytes

Average size of id = 4 bytes (int)

Status = 4 bytes

Daily new URL = 100,000 * 108 = 10.8MB

Yearly URL = 10.8 MB * 365 ~ 4GB



Single Machine

Evolve – Single Machine

- How to implement time-limited service?
  - Set a expiration period (in status)
    * delete url in 3 years after its creation
    * delete url if it is not used during the past 3 years
- How to cache?
  - Pre-load: load data into memory when starting server
  - Replacement: LRU Cache

Evolve – CAP

Guarantees	Definition
Consistency	Every read receives the most recent write or an error
Availability	Every request receives a response, without guarantee that it contains the most recent version of the information
Partition Tolerance	The system continues to operate despite an arbitrary number of messages being dropped by the network between nodes

ONLY PICK TWO!

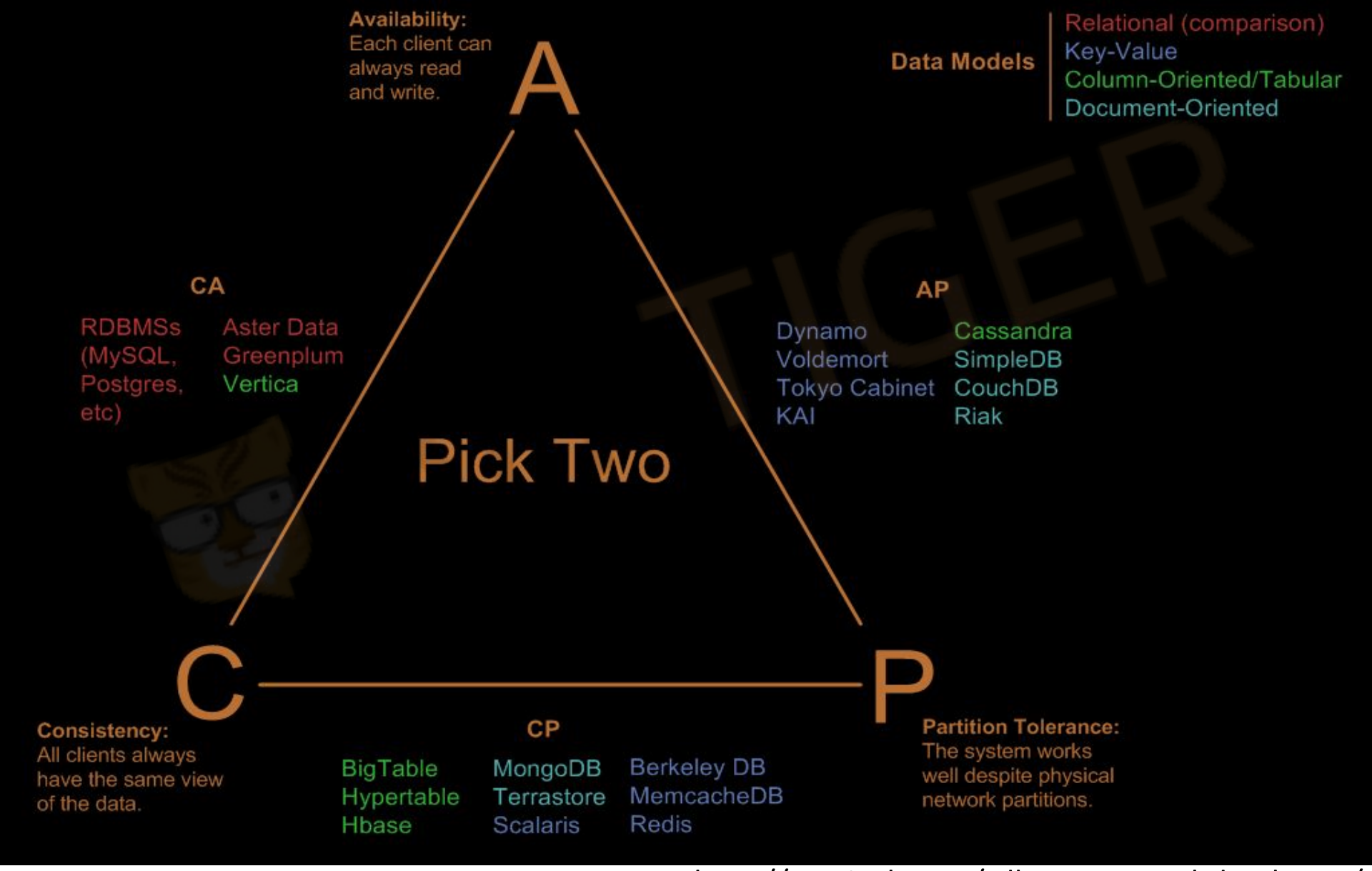
https://en.wikipedia.org/wiki/CAP_theorem

Distributed Systems

- Distributed System Features:
  - No distributed system is safe from network failure
  - Networking partition has to be tolerated
- Choose between Consistency and Availability in the presence of a partition
- In the absence of network failure, both availability and consistency can be satisfied

https://en.wikipedia.org/wiki/CAP_theorem

Visual Guide to NoSQL Systems



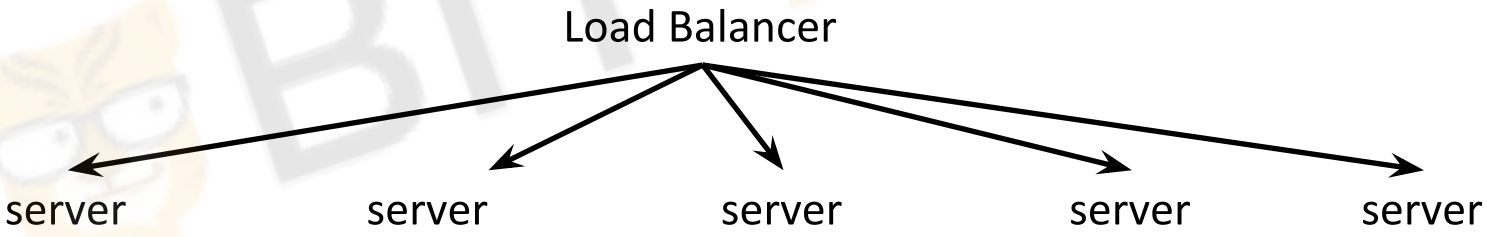
http://maxivak.com/rdbms-vs-nosql-databases/

Evolve – Multiple Machine

Problem: What if a large amount of users frequently visit limited url?

Demand: High RPS for lookup, limited amount of frequent url

REPLICATION – store full data on all machines



Problem:

- Write-Write conflict:

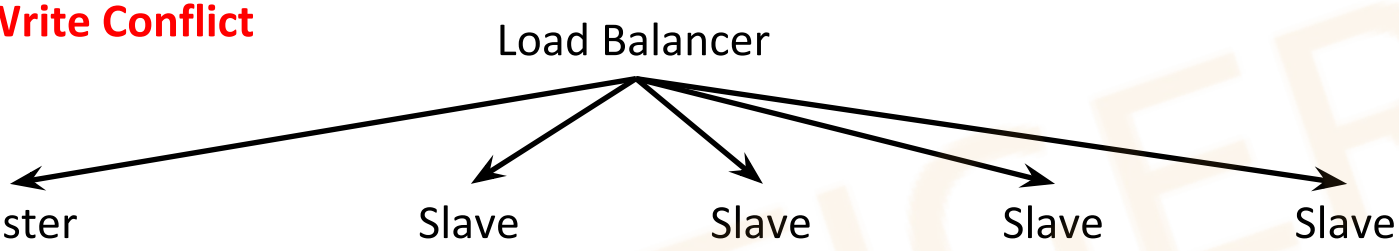
  What if A writes google.com on server1, while B writes google.com on server2?

- Read-Write conflict:

  What if A writes google.com on server1, while B reads google.om on server2?

Evolve – Consistency

Write-Write Conflict



- Do lookup operations
- Do insert operations, and populate to other slaves
- Store Full data

- Only do lookup operations
- Store Full data

Read-Write Conflict

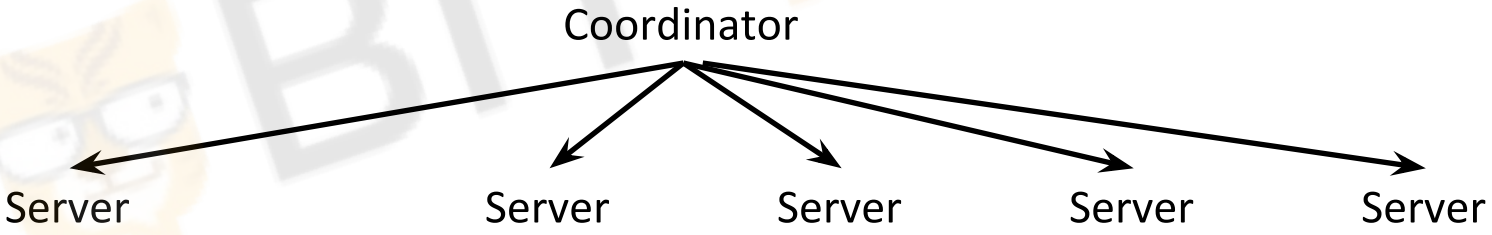
- Before finishing write on master, no response could be gotten from other servers.
- Append-only system, no need to worry update and choice between different versions.
- If allowing update, a choice between consistency and availability must be made.

Evolve – Multiple Machine

Problem: What if a large amount of data to store?

Demand: A huge storage capacity for data, high RPS for both insert and lookup

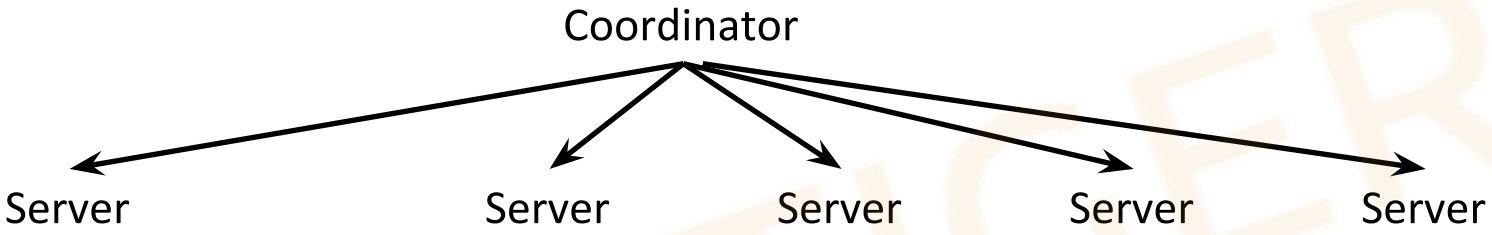
SHARDING



- Do both lookup and insert operations
- Store part of the data in each server
- Coordinator decide which server to send (a hash function from id to server)

Evolve – How to dispatch?

SHARDING



No id conflicts in all servers – mod

- Server #0 ~ 4, only save url that url_count % 5 == #
- For write request, dispatch it randomly. In server i, actual_size = size * 5 + i, use actual_size to calculate short_url
- For lookup request, compute short_url to actual_size, use actual_size % 5 to decide which server to dispatch

Con

Cannot eliminate duplicates for same long_url

Summary

- SNAKE Analysis
- Application:
  - traditional hash
  - base10
  - base62
- Evolve:
  - CAP Theorem in Distributed System
  - Replication & Sharding