Databases intro

- Anubhav Bindlish (software engineer, Rockset)

“The world’s most valuable resource is no longer oil, but data”

- The Economist

- The differences between “tech” companies and others are reducing
- Every industry leverages tech
    - HBO, Disney v/s Netflix, Youtube
    - Walmart v/s Amazon
    - Digital media companies v/s Facebook (Meta)
- User interactions generate data
- Data can be used to
    - Introduce efficiency
    - Improve user experience
    - Etc

Scale of modern companies is huge

Q. Amount of data on internet?

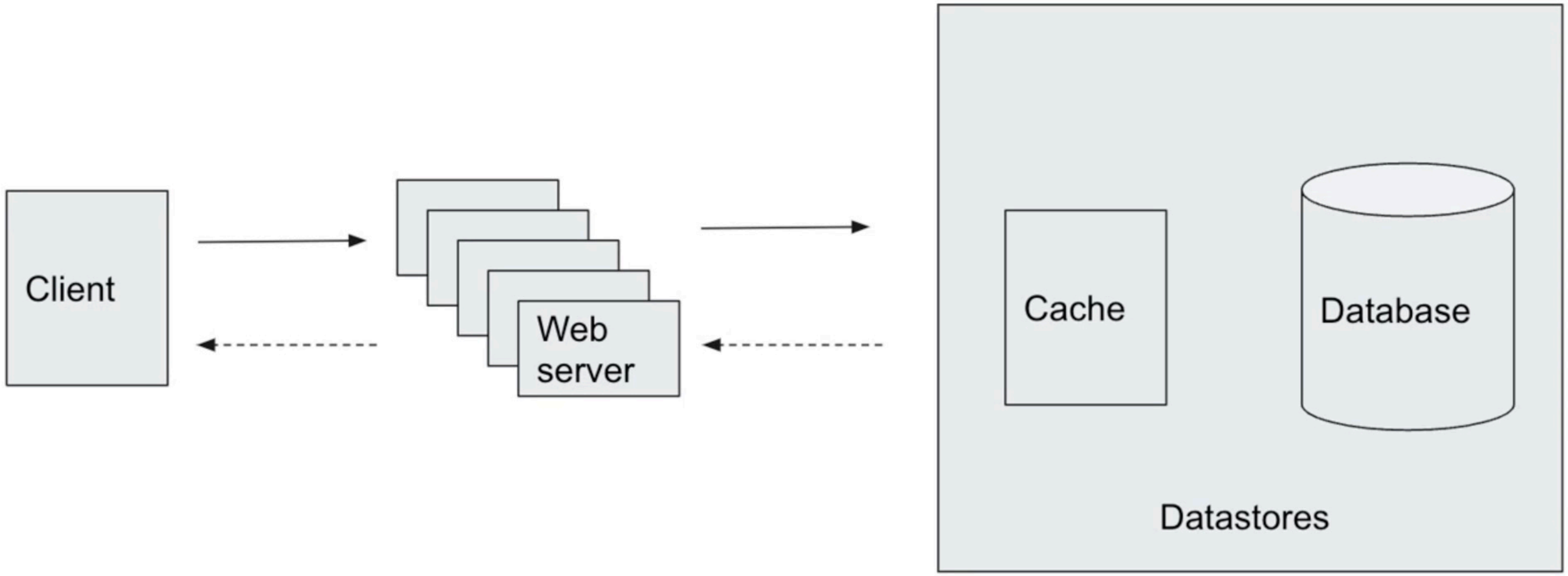
A. Estimated to be in range of **10s of zettabyte**

1 zettabtye is 10^9 Terabytes (or 50Billion USD in western digital SSDs, at 50$ per TB)

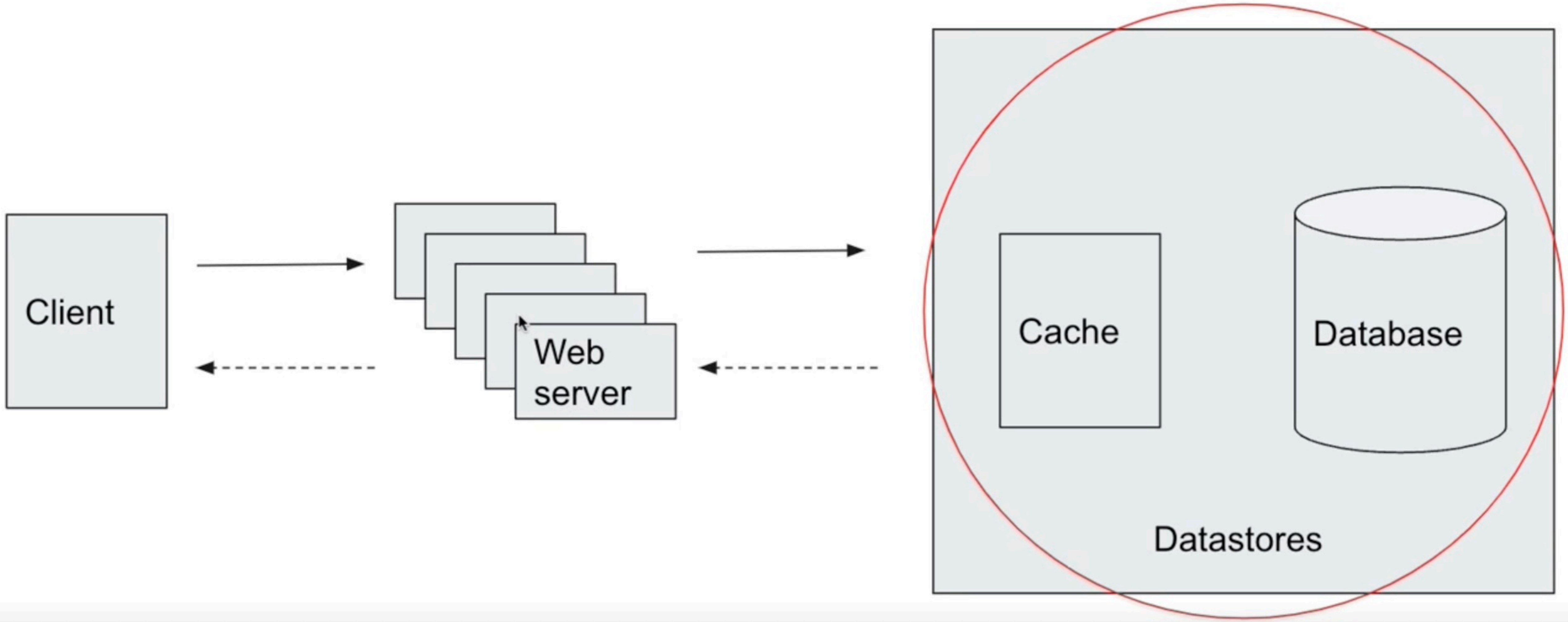
- How do companies deal with all this data coming in?

- Where is it stored?
- How is it searched?
- What problems do they face?

Let’s start with a high level flow of a web request

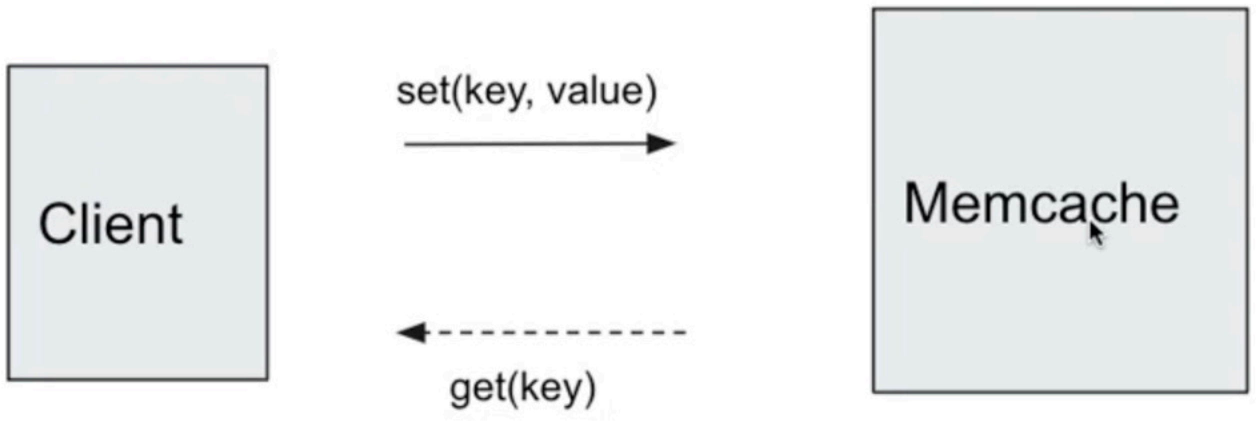


Let's start with a high level flow of a web request



Caches

- Let’s say you’re responsible for building a caching service
- Keeping things in memory would be faster
- What’s the simplest interface you can imagine?



Client

set(key, value)

get(key)

Memcache

What about backend data itself?

- Lots of things to decide here

- Data model
- Data structures to implement said model
    - Efficiency is paramount
    - A 2x improvement in efficiency saves of Billions of dollars
- How to handle failures
- Etc

This is a hard computing problem!

Data model

- Largely two: relational (SQL) v/s non-relational (key/value)

- Relational
  - Highly structured data
    - For eg: student records
  - Point query/updates

- Non-structured
  - Some sort of key->value mapping
  - Scaling is easier (fewer invariants)

MySQL

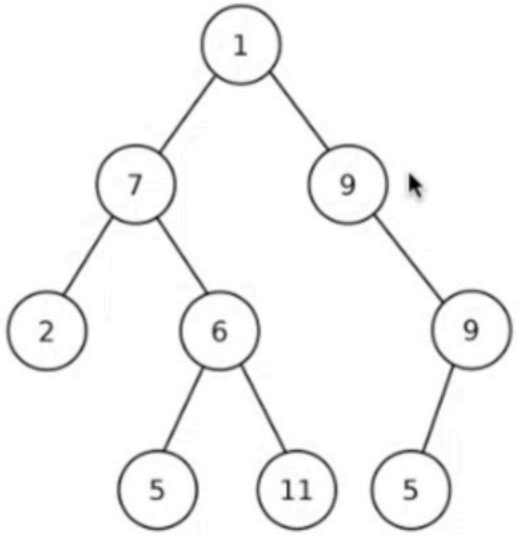
mongoDB

cassandra

Data structures

- Heart of everything
- You will learn plenty on this in your next 3-4 years of undergrad
- You want fast reads and writes

Linear log (list / array / etc)

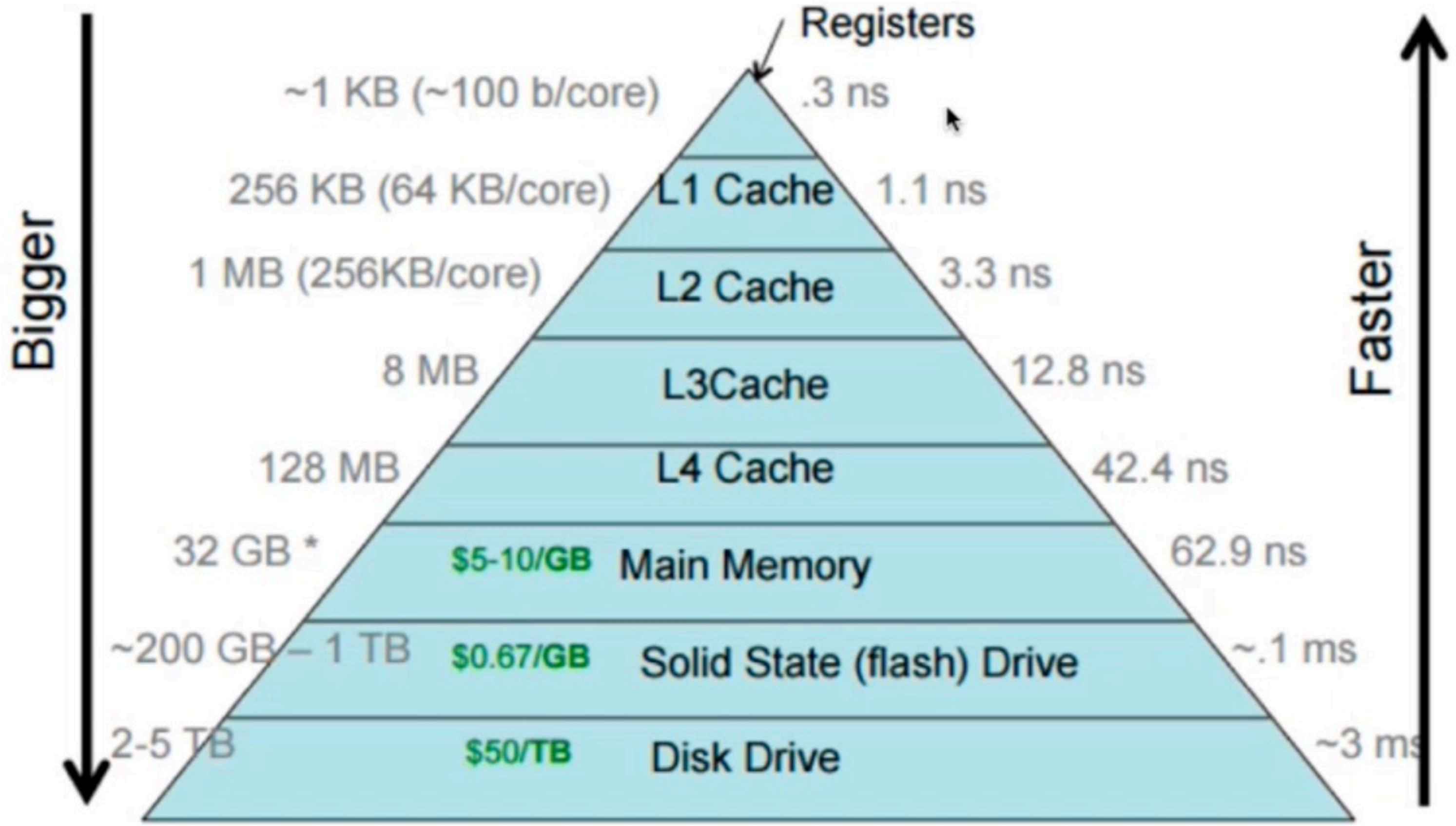


Binary tree

Not just this

- Disk v/s memory
- CPU cache v/s memory
- Which CPU cache (L1/L2/L3)
- Trust no-one: is the compiler doing the right thing?

Know your hardware



From https://cs.brown.edu/courses/csci0300/2022/assign/labs/lab4.html

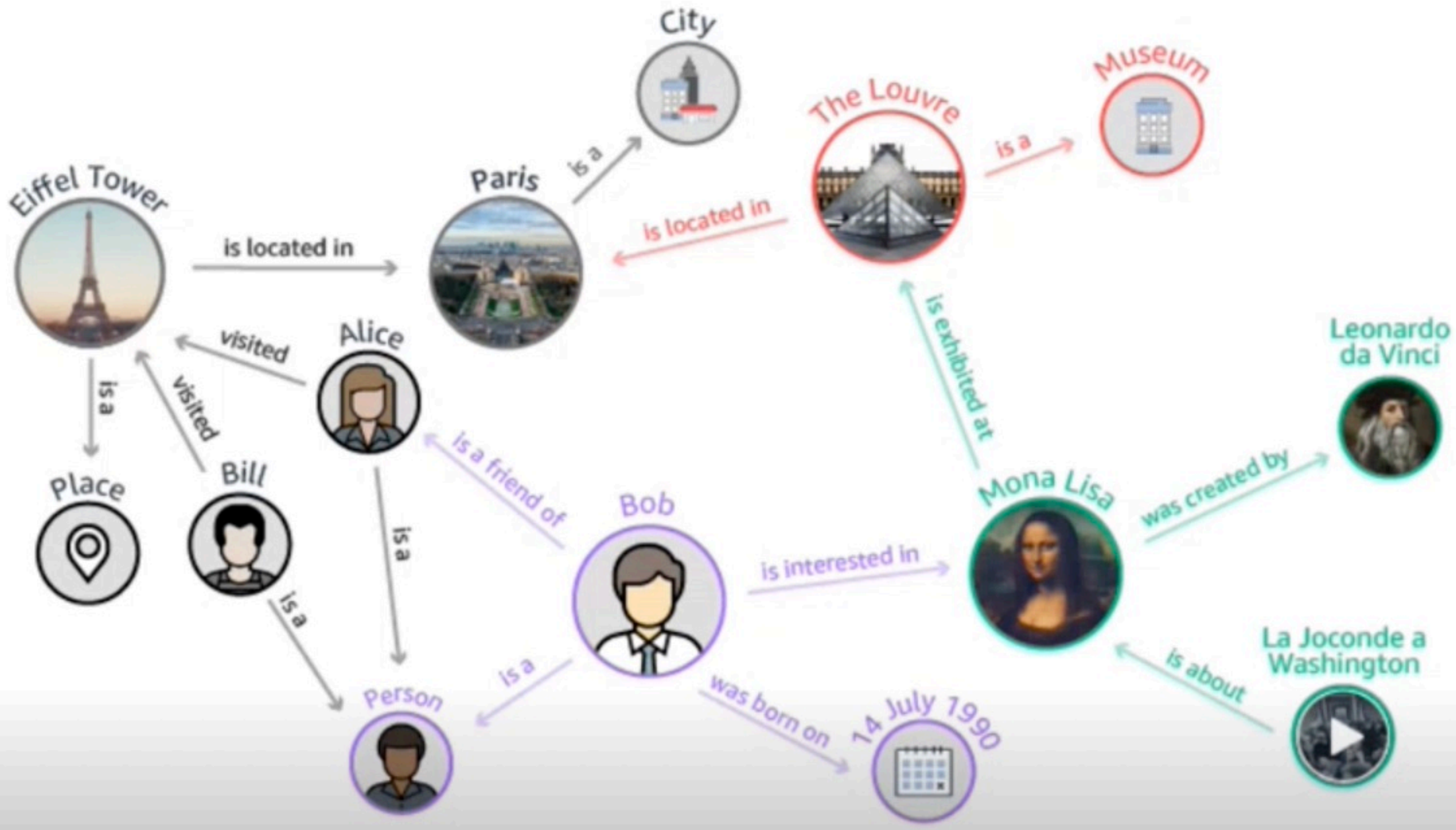
Case study: TAO (Meta)

- circa 2005 Facebook was Memcache + MySQL
- MySQL implementation internal assumptions didn’t match facebook workloads
- Memcache API left a lot for product developers
  - “for this web-request, what is the key?”
  - “must call DB on miss”
- Developers should be able to “move fast”

From https://engineering.fb.com/2013/06/25/core-data/tao-the-power-of-the-graph/

Case study: TAO (Meta)

- In 2007 Facebook engineers built object and association API
    - Entirely client side (PHP)
    - Immediate success

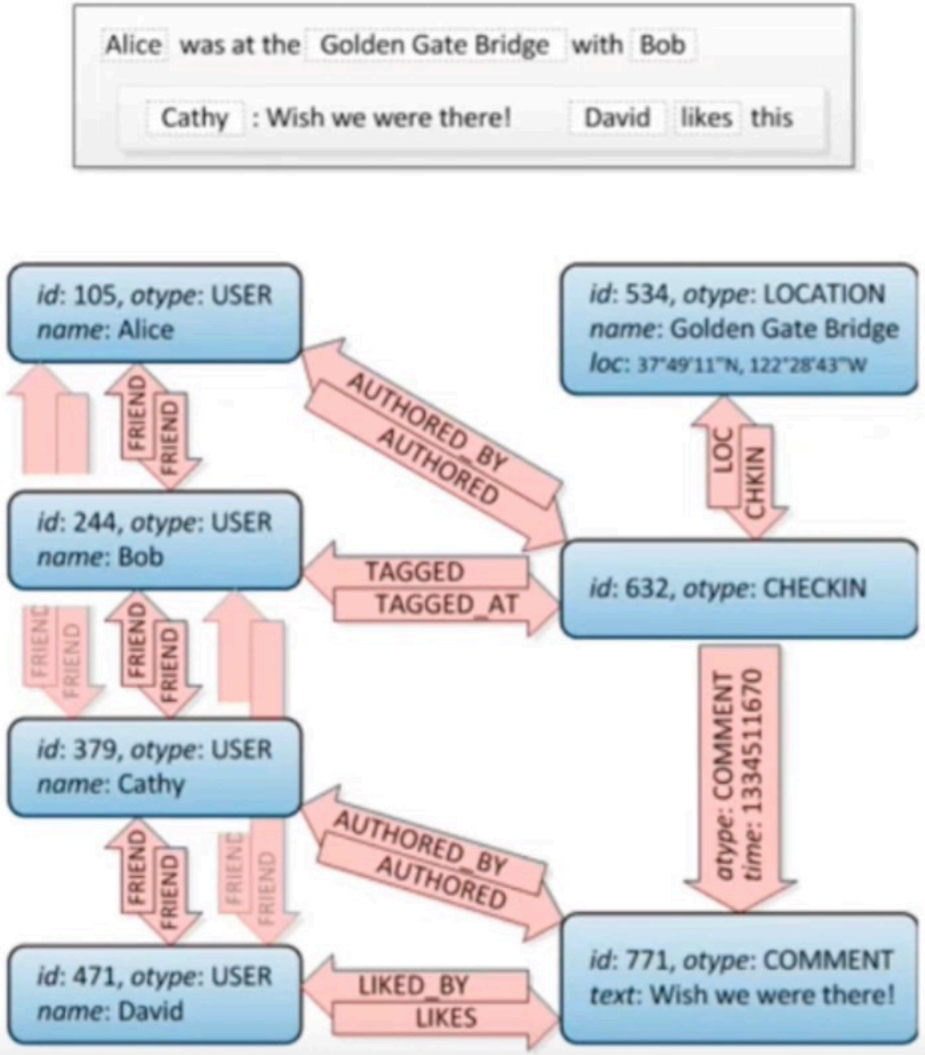


Case study: TAO (Meta)

- Limitations
  - Cache invalidation: when one edge updated, entire object invalidated
  - **Get(friends who like Virat Kohli)** had to transfer entire list from backend -> web
  - Others

- So eventually, custom server side implementation

Case study: TAO (Meta)



- Objects and associations
- Inverse edges
- Simple API: get / set / delete

Case study: TAO (Meta)

- Use cases

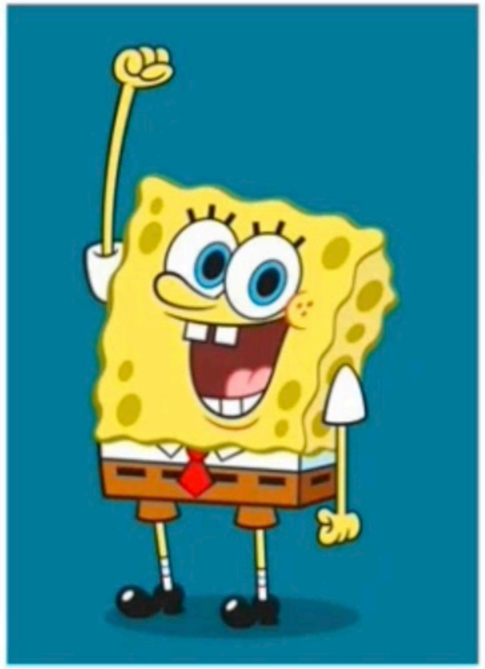
- Is Alice a friend of Bob?
    - Point lookup: **get(id1, id2, assoc-type)**
- All friends of Bob?
    - Range lookup: **get(id1, assoc-type)**
- How many friends of Bob?
    - Count: g**etCount(id1, assoc-type)**

- Why count separate API and not covered by range lookup?
- Efficiency!!

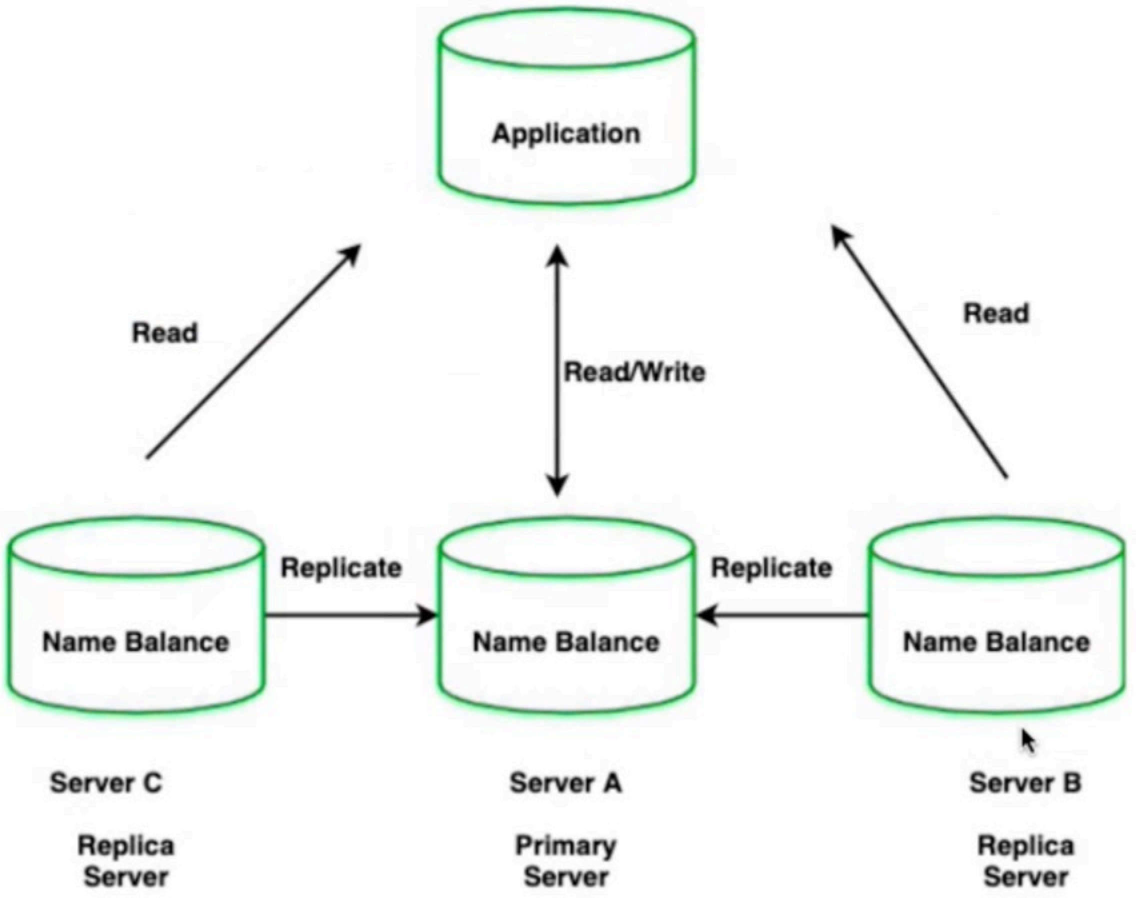
How to scale / handle failures?

- Things will **always** fail
- Your service should be able to operate with failures

- Sharding and replication FTW!!

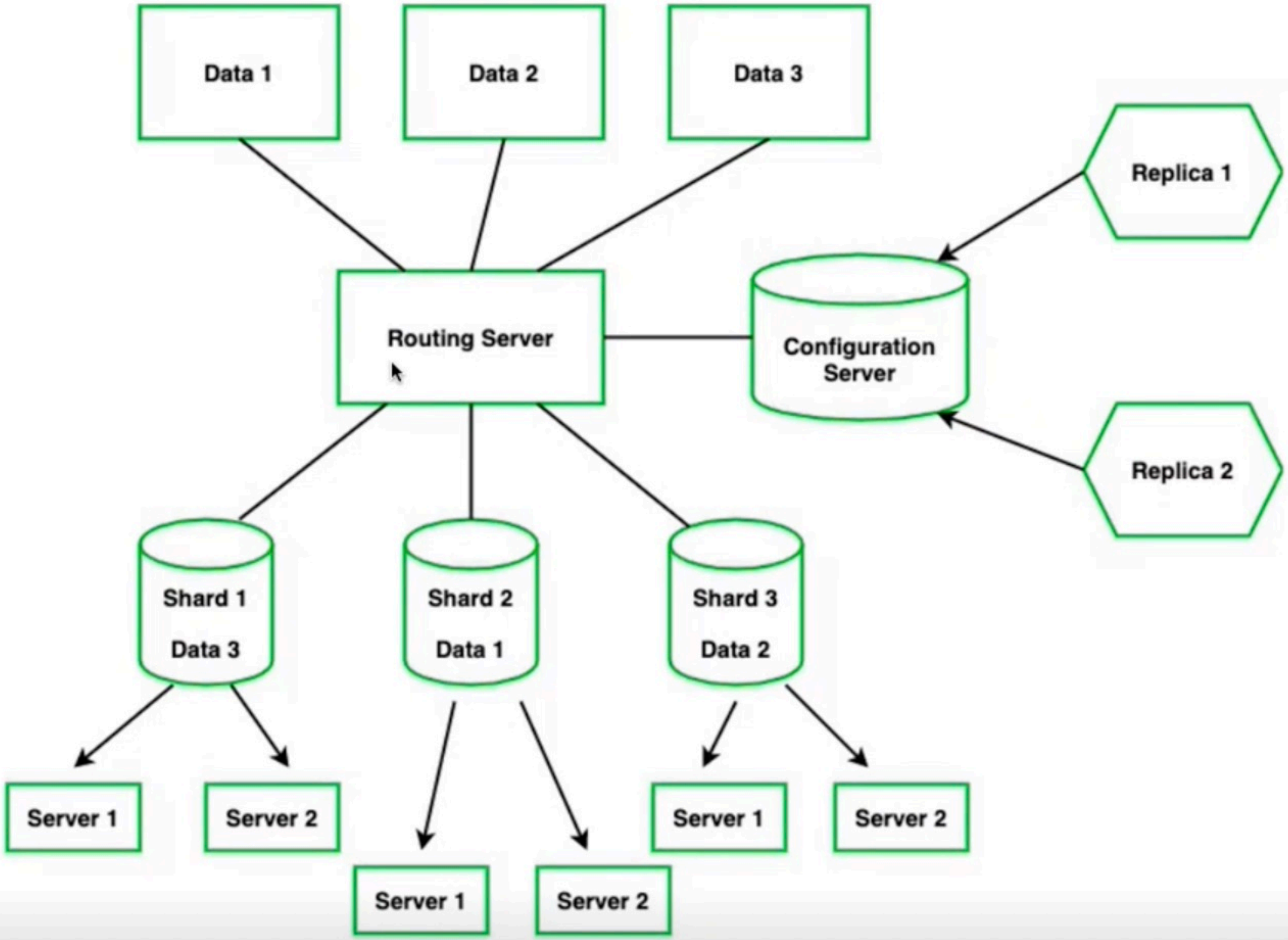


Replication



From https://www.geeksforgeeks.org/mongodb-replication-and-sharding/

Sharding



From https://www.geeksforgeeks.org/mongodb-replication-and-sharding/

Key takeaways

- Data is an important resource, companies tapping into it
- Designing databases: hard computing problem
  - Better solution == Better user experience; cost savings
- Both design and API is important
  - Service should not only be fast, but also **easy to use**
  - Car with good engine but bad seats won’t sell

Thankyou!