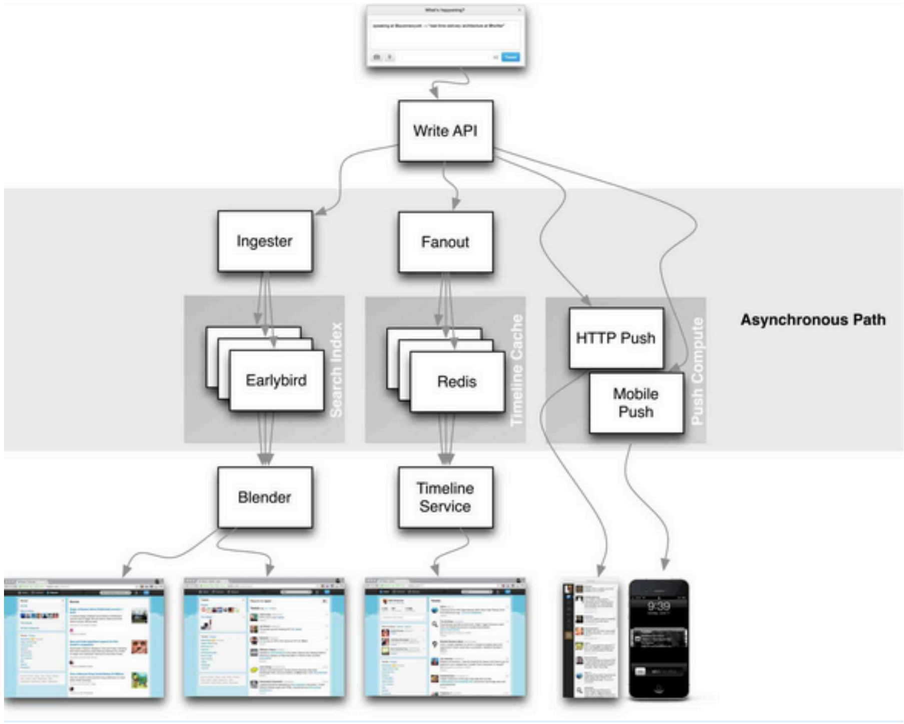
UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

DSD Overview

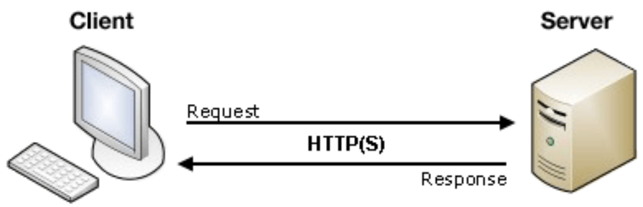
Sami Rollins

Why study distributed systems?

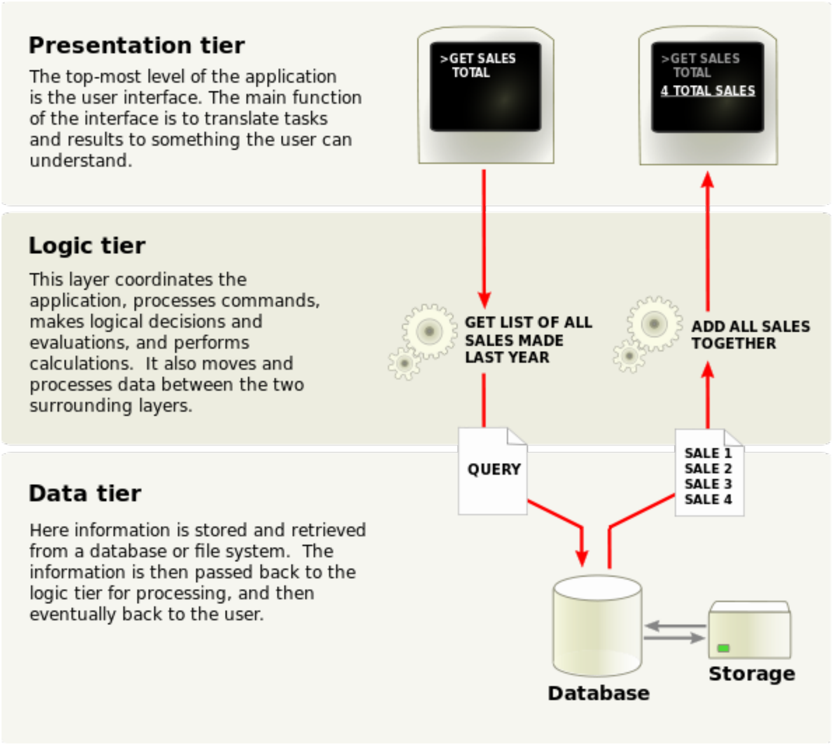
- **You use them everyday!**
- **Scale is the name of the game**
- **Distributed systems are complex**
  - Understanding what is under the hood is important



History lesson

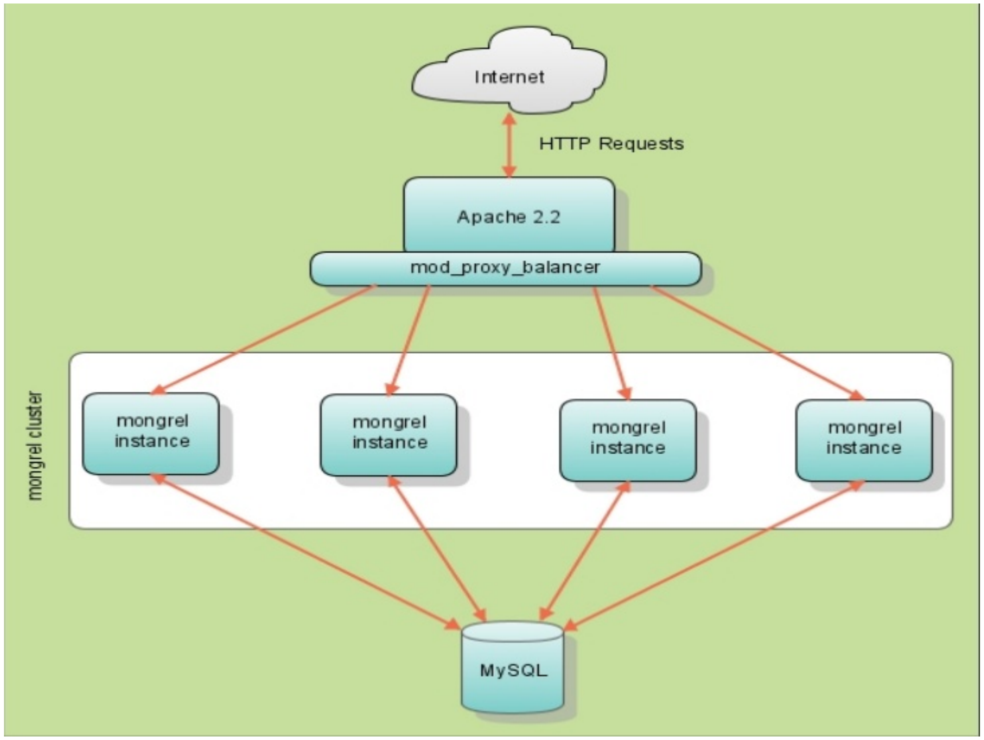


- **Client/server or n-tier Architecture**
  - Limited scalability
  - Database often a bottleneck

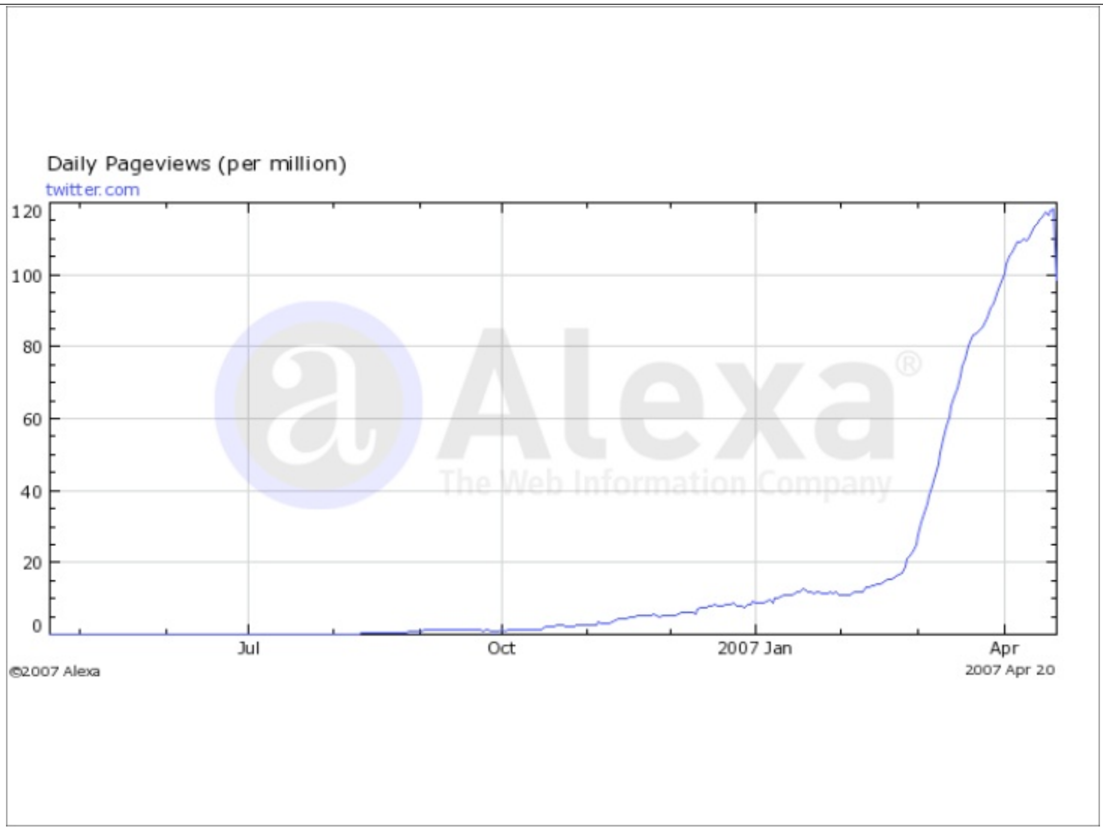


Twitter case study

- **Original Twitter architecture**
  - Ruby on Rails
  - MySQL
    - 1 server
    - 2400 requests/second
  - Mongrel (web server)
  - Memcached



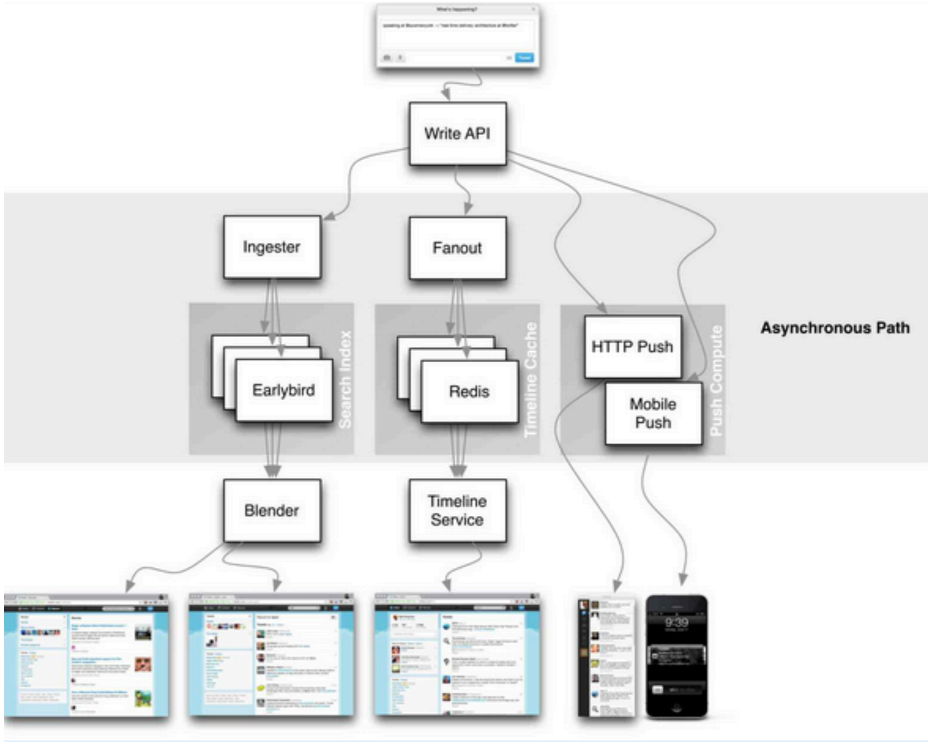
Twitter case study



UNIVERSITY OF SAN FRANCISCO

Twitter case study

- **Optimize reads**
  - insert
    - for each follower, insert new tweet in redis cache
- **Service-oriented architecture**
  - System is a set of loosely coupled services



Non-web distributed systems?

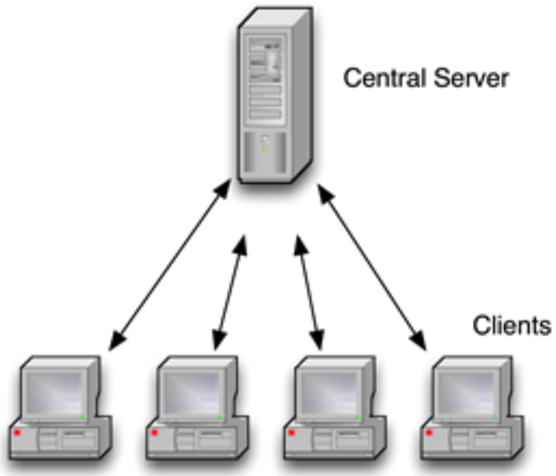
- **Twitter, Pinterest, Google, Etsy**
  - All are web-based applications

- **What about other kinds of applications?**
- **Characteristics**
  - Concurrency
  - Independent failures
  - No global clock
  - Heterogeneity
  - High latency communication
  - System composed of computers spread over a large geographic area, maybe under different administrative domains

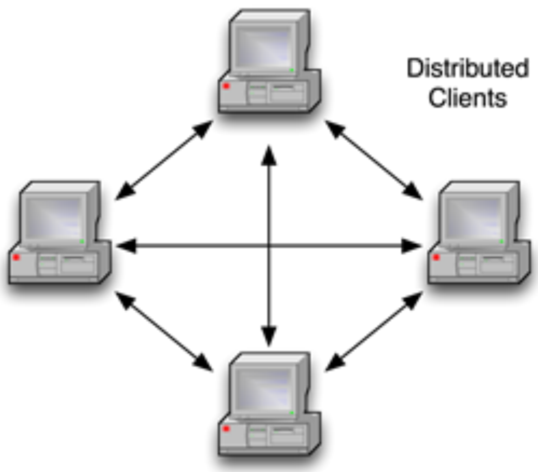
Other distributed systems

- **Peer-to-peer**
  - File sharing/distribution
  - Gnutella?
  - Bit torrent
  - Others?
- **MMOG**
- **DNS**
- **CDNs**
  - Akamai
  - Limelight
  - Netflix Open Connect

Architectures

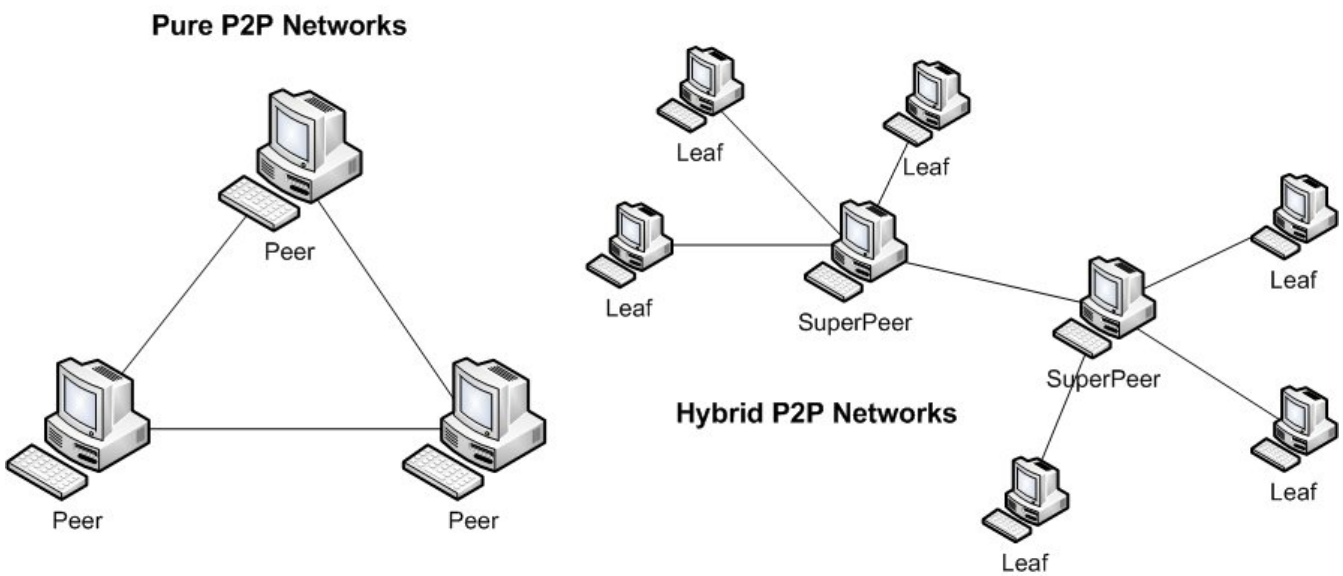


Client / Server



Peer to Peer

Architectures



Bit torrent - http://en.wikipedia.org/wiki/Peer-to-peer

Main challenges?

- **What do you think are the biggest challenges in building a distributed system?**

Challenges

- **Heterogeneity**
  - different networks, OSs, architectures
  - communication protocols must be carefully defined
- **Openness**
  - Can new services/components be easily added?
- **Security**
  - confidentiality, protection against corruption, resistant to attack
- **Scalability**
  - tolerates increase in users and/or resources

Challenges

- **Failure handling**
  - Can the system detect, mask, tolerate, recover from failures?
- **Concurrency**
  - able to handle multiple requests simultaneously
- **Transparency**
  - local and remote resources accessible in the same way

Principles

- **Availability**
  - Enough resources to handle all traffic
  - Failure is handled gracefully
- **Performance**
  - Fast response time
    - Sufficient hardware
    - Geographic placement

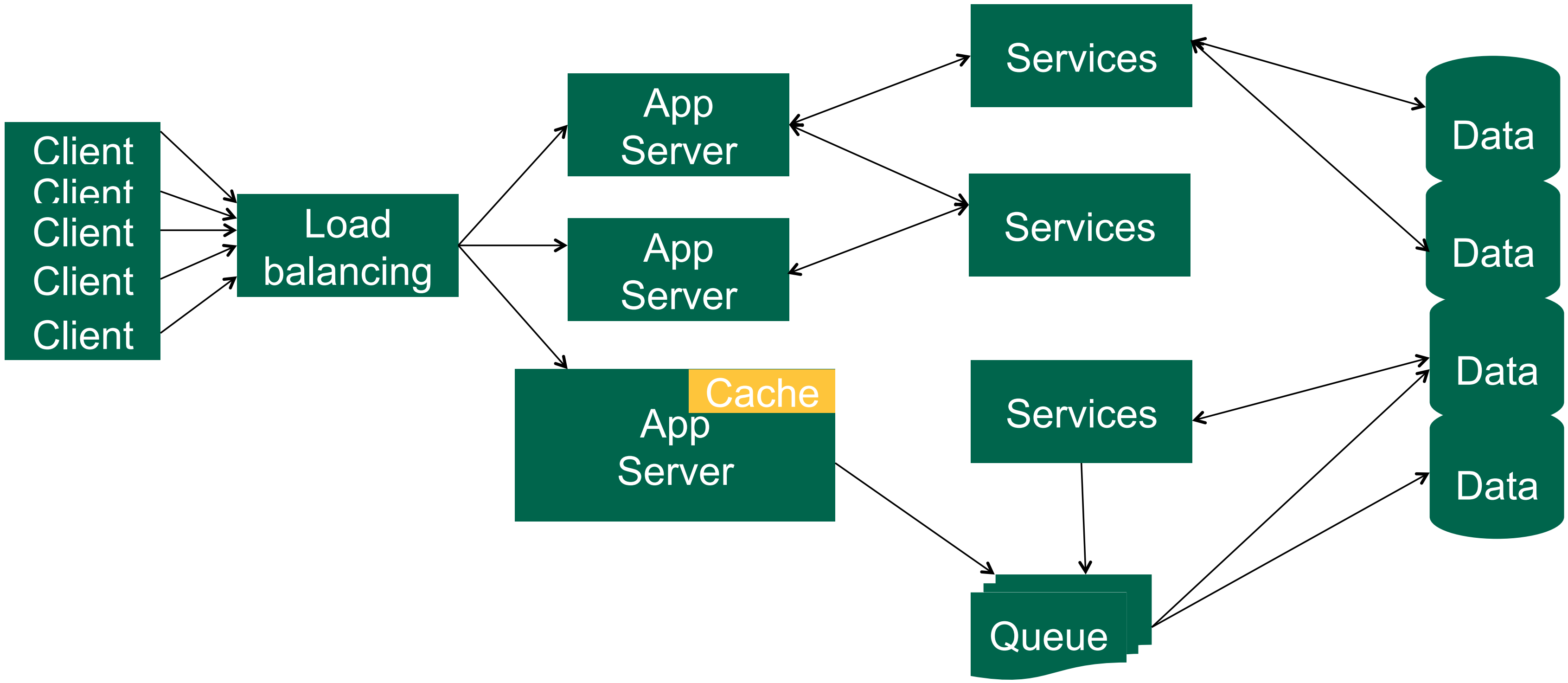
Principles

- **Reliability**
  - Fault tolerance
  - Consistency
    - Eventual consistency
    - Strong consistency
- **Scalability**
  - Increasing resources, supporting more users
- **Manageability**
  - System updates, failures
- **Cost**

CAP theorem – Pick two

- **Consistency**
  - All nodes see the same data at the same time
- **Availability**
  - A guarantee that every request receives a response about whether it was successful or failed
- **Partition tolerance**
  - The system continues to operate despite arbitrary message loss or failure of part of the system

Pieces of the puzzle



Design decisions

- **Load balancing algorithms**
  - DNS round robin
  - Random
  - CPU/Memory utilization
- **Service-oriented Architecture (SOA)**
  - Replication versus partitioning of functionality
- **Caching**
  - Policies for populating cache and evicting data
  - Memcached
  - Redis

Design decisions

- **Data**
  - Replicated versus partitioned
- Asynchronous processing
  - Log requests for “big data” processing, transformation, etc
  - Kafka