**System Design**

1. Distributed Systems

Different servers, complexity hidden from user (Abstraction)

Fallacies of Distributed systems

***Characteristics***

* No shared clock
* No shared memory
* Shared resources
* Concurrency and consistency

***Communication Issues***

* Client can’t find server
* Server Crash mid request
* Server response
* Client Chrashes

***Advantages***

* More reliable, fault tolerant
* Scalability
* Lower latency, increased performance
* Cost Effective

1. System Design Performance Metrics

* Scalability
* Ability of a system to grow and manage increased traffic
* Increased volume of data or requests
* Reliability
* Probability a system will fail during a period of time
* Slightly harder to define than hardware reliability
* Mean Time Between failure
* Availability
* Amount of time system is operational during a period of time
* Poorly designed software requiring downtime for updates is less available
* Availability % = (availability time/ total time) X 100%
* Efficiency
* How well it performs
* Manageability
* Speed and difficulty involved with maintaining system
* Observability, how hard to track bugs
* Difficulty of deploying updates
* Want to abstract away infrastructure so product engineers don’t have to worry about it

1. Numbers Programmers Should Know

* Latency Numbers
* Latency Key Takeaways
* Avoid Network calls whenever possible
* Replicate data across data centers for disaster recovery

As well as performance

* Use CDNs to reduce latency
* Keep frequently accessed data in memory if possible

Rather than seeking from disk, caching

* Quick Math for Capacity Estimates
* Data Conversion
* Common Data Types
* Time
* Traffic Estimates
* Estimate total number of requests app will receive
* Average Daily Active Users X averge reads/writers per user
* Memory
* Bandwidth
* Request per day x Request size
* Storage

1. Horizontal VS Vertical Scaling
2. Vertical Scaling

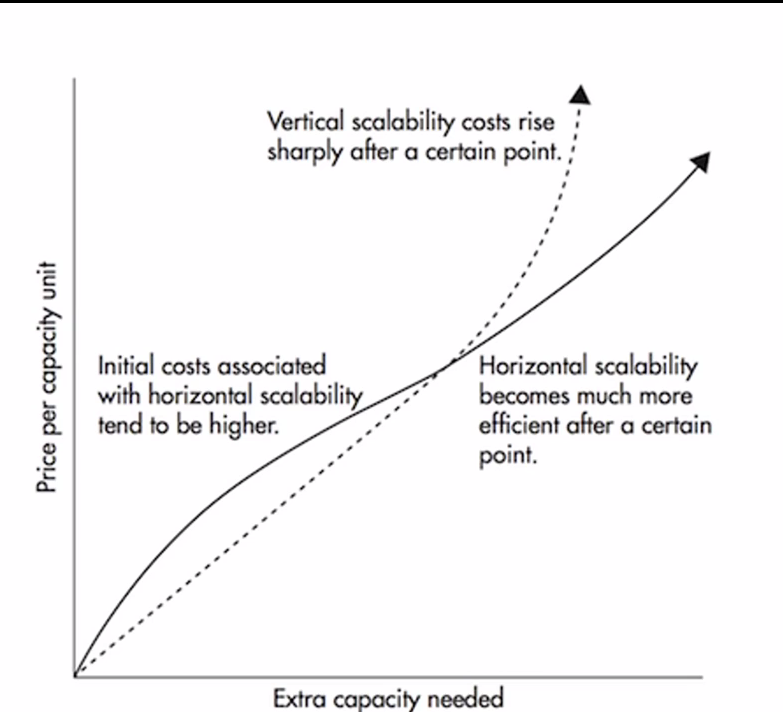
* Easiest Way to scale an application
* Diminishing returns, limits to scalability
* Single point of failure

1. Horizontal Scaling

* More Complexity up front, but more efficient long term
* Redundancy built in
* Need load balancer to distribute traffic
* Cloud providers make this easier

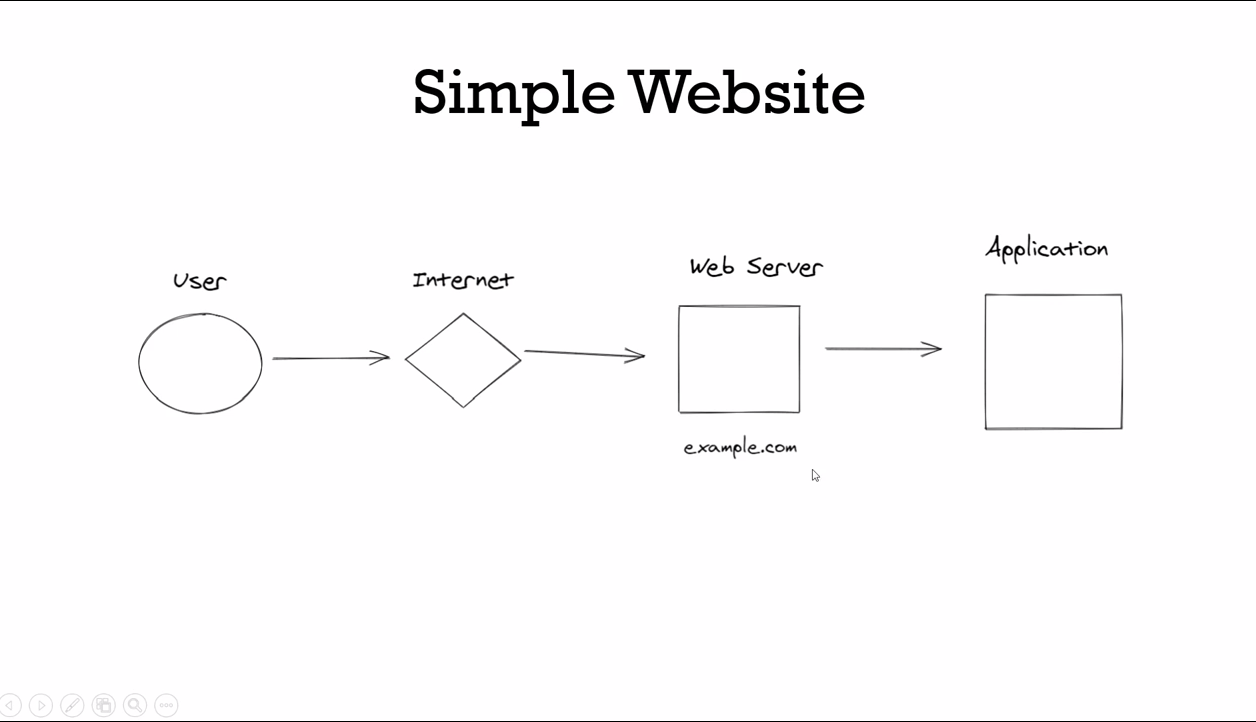
1. Tradeoffs(Vertical scaling)

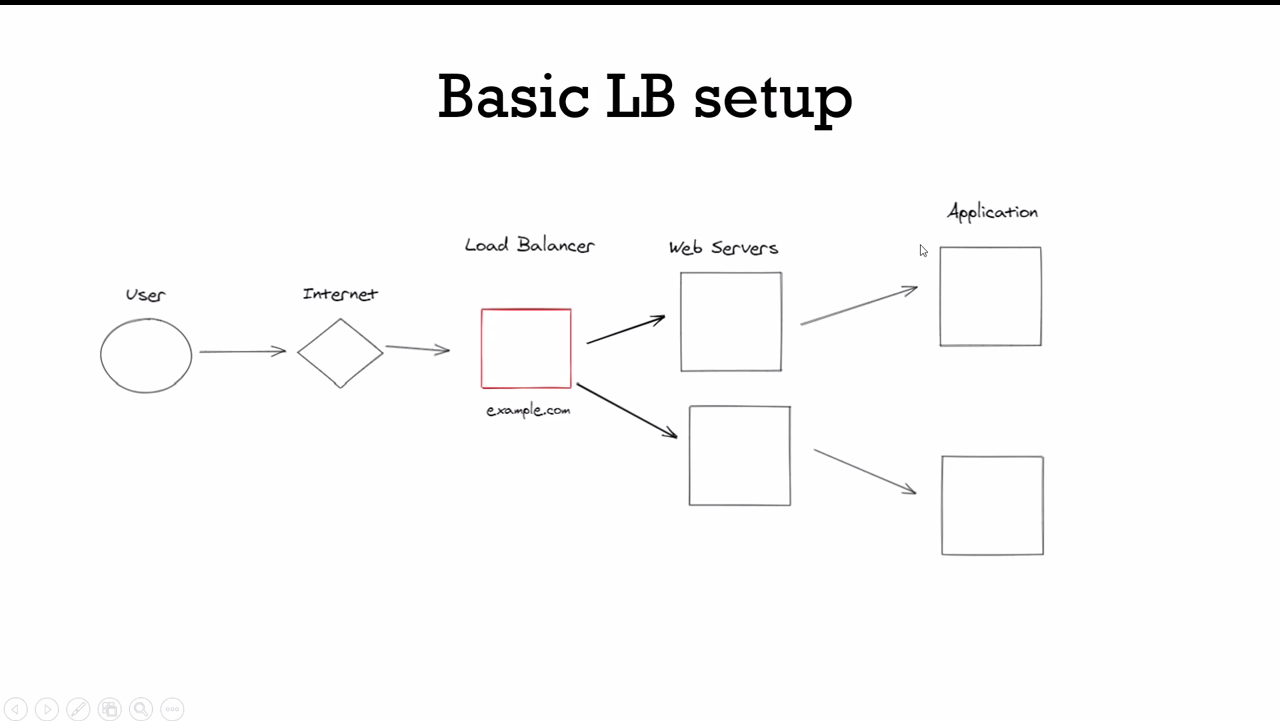
* Diminishing Returns
* Latency(Horizontal Scaling)



1. System Design Concepts
2. Load Balancers

* Balance incoming traffic to multiple servers
* Software or Hardware based
* Used to improve reliability and scalability of application
* Nginx, HAProxy, F5, Citrix





1. Load Balancer Routing Methods

* Round Robin
* Simplest type of routing
* Can result in uneven traffic
* Least Connections
* Routes based on number of client connections to server
* Useful for chat or stream applications
* Least Response Time
* Routes based on how quickly servers respond
* IP Hash
* Routes client to server based on IP
* Useful for stateful sessions

***L4 vs L7***

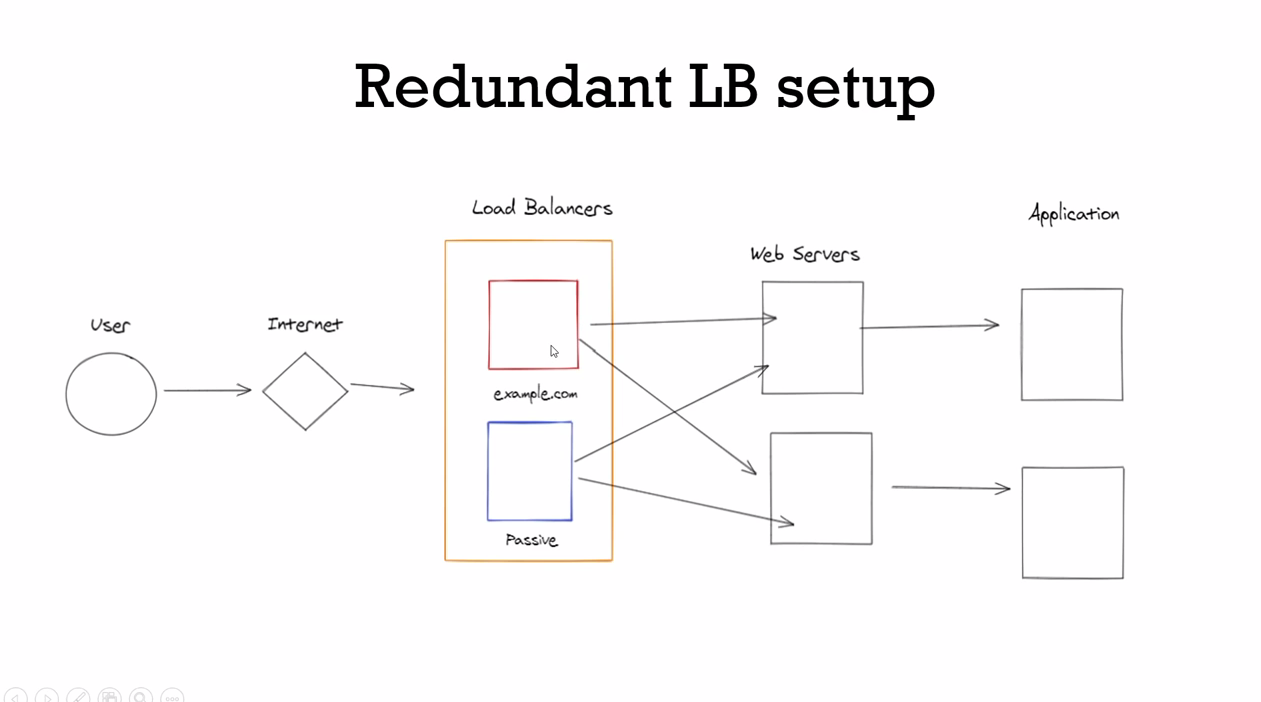
1. Layer 4

* Only has access to TCP and UDP data
* Faster
* Lack Information can lead to uneven traffic

1. Layer 7

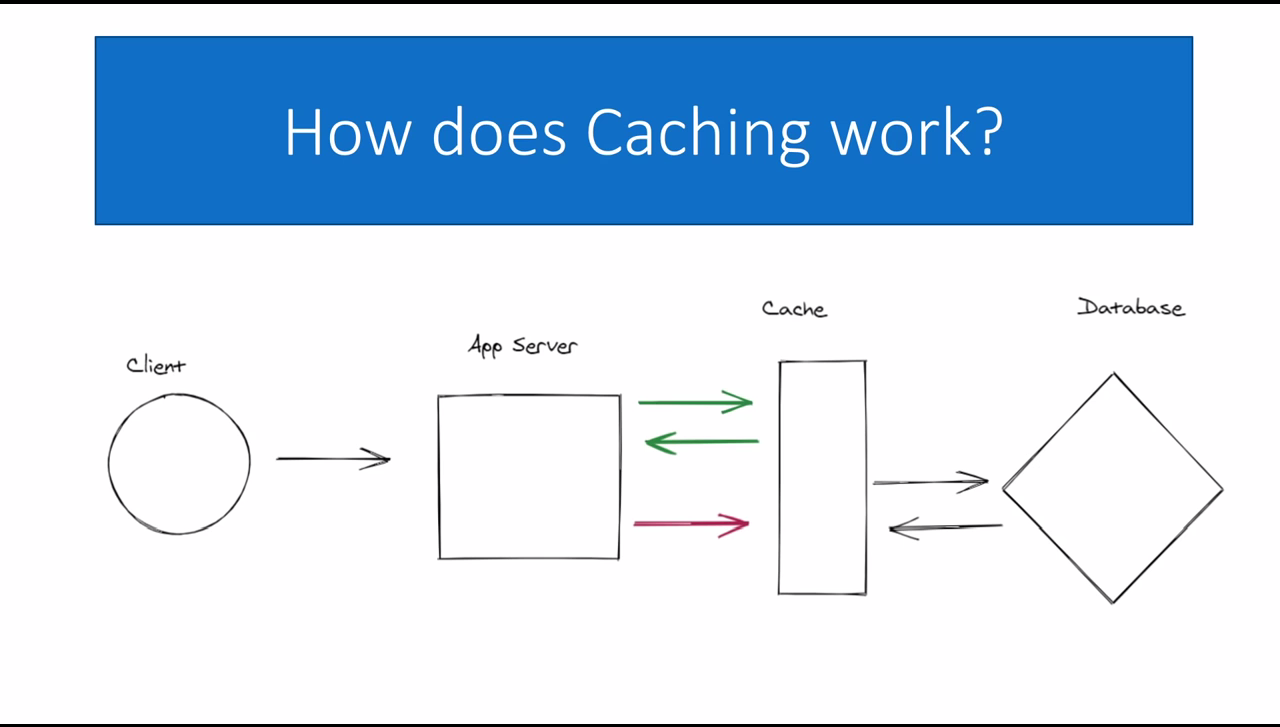
* Full access to HTTP protocol and data
* SSL termination
* Check authentication
* Smarter routing options

1. Redundant LB setup



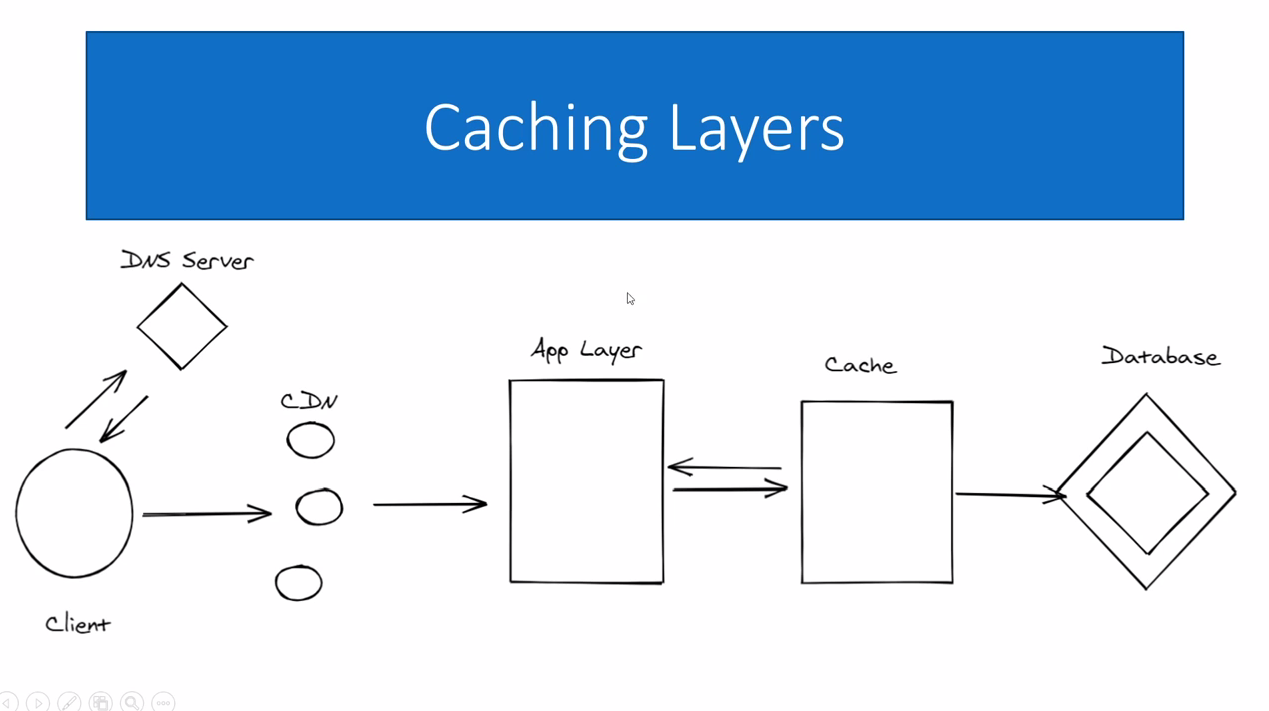
1. Caching

* Improve Performance of application
* Save money

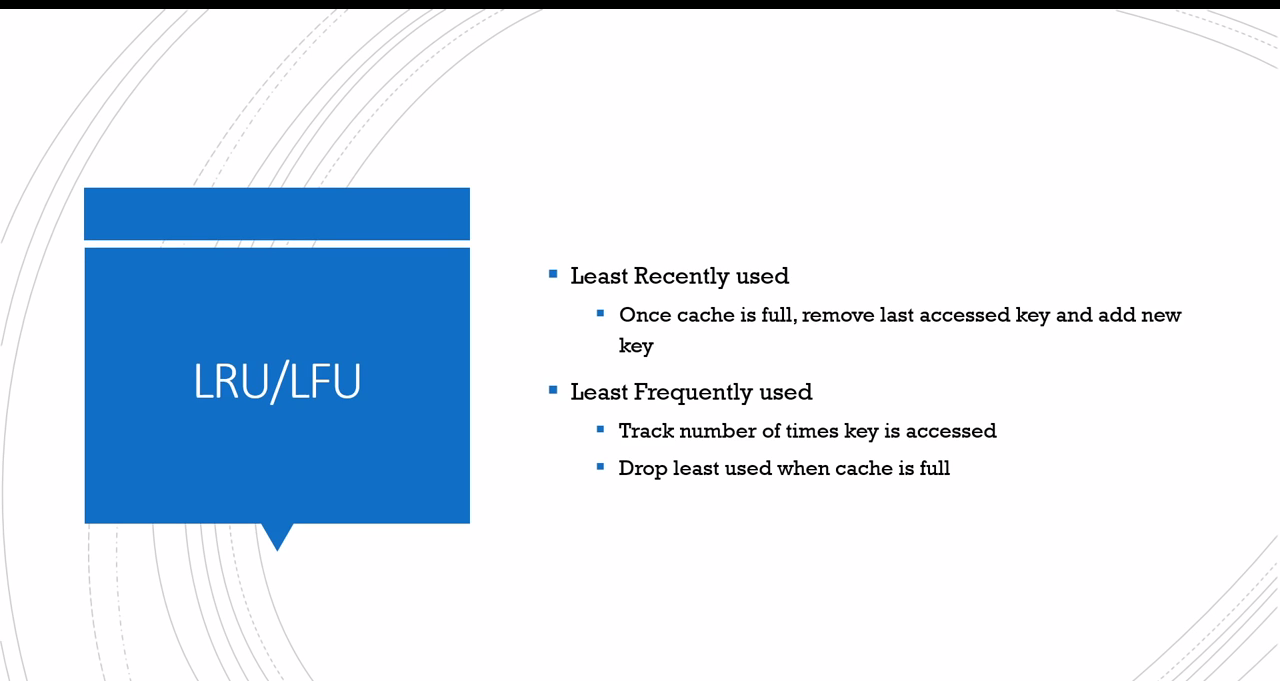


1. Caching Layers

* DNS
* CDN
* Application
* Database



1. Distributed Cache System
2. Cache Eviction
3. TTL (Time to Live)
4. LRU/LFU



1. Cache Strategies



1. Database Scaling
2. Key information – most web are majority reads, around 95%+
3. Scaling Techniques

* Indexes
* DE normalization
* Connection pooling
* Caching
* Vertical scaling

1. Replication and Partitioning

