

Scalability of web applications



Overview

- Scalability questions, what's important?
- High availability vs. load balancing
- Approaches to scaling
 - Performance tuning, horizontal, vertical scaling
- Multiple web servers
 - DNS based sharing, hardware/software load balancing
- State management
- Database scaling
 - Replication
 - Splitting things up

Scalability related questions

- Where is your **session state** being stored? Why?
- How are you generating **dynamic content**? Why?
- Are you regenerating things that could be **cached**?
- What is being stored in the **database**? Why?
- Could you **be lazier**?
 - Do you need exact answers? e.g. page 1/2063
 - Queue up work if it doesn't need to be done right now
 - e.g. Do the user really need that video thumbnail right now?
- What do you **care about**?
 - Time to market, money, user experience, uptime, power efficiency, bug density, ...

High availability / load balancing

- High availability

- Carrying on running despite failure of components
- May involve load-balancing, but not necessarily
 - e.g. Hot/warm standby that doesn't do anything until primary fails
- Also allow easier updating of components
 - e.g. Avoid maintenance windows in the middle of the night

- Load balancing

- Effectively combining resources from multiple systems
- Send request to somebody else if a certain system fails
- May provide high availability, but not necessarily
 - e.g. Adding a single-point of failure load balancing appliance

Availability 9s

Availability %	Downtime per year
90%, "one nine"	36.5 days
99%, "two nines"	3.65 days
99.9%, "three nines"	8.76 hours
99.99%, "four nines"	52.56 minutes
99.999%, "five nines", "carrier grade"	5.25 minutes
99.9999%, "six nines"	31.5 seconds

Approaches to scaling

- Make existing infrastructure go further
 - Classic **performance tuning** :
 - Find the bottleneck
 - Make faster if you can
 - Find the new bottleneck, repeat
 - How are you generating **dynamic content**? Why?
 - Where is your **session state** being stored? Why?
 - What is being stored in the **database**? Why?
 - Can you **be lazier**?
 - Do you need exact answers? e.g. page 1/2063
 - Add work to a queue if it doesn't need to be done right now
 - e.g. Do you need that video thumbnail before proceeding?

Approaches to scaling

- Vertical scaling (scale up)
 - Buy more memory, faster CPU, more CPUs, SSD disks
 - Quick fix, use existing software/network architecture
 - But there is a limit, plus price premium for high end kit



ABMX server, 1u
1 core @3.1 Ghz, 1GB memory, 80GB disk
\$397

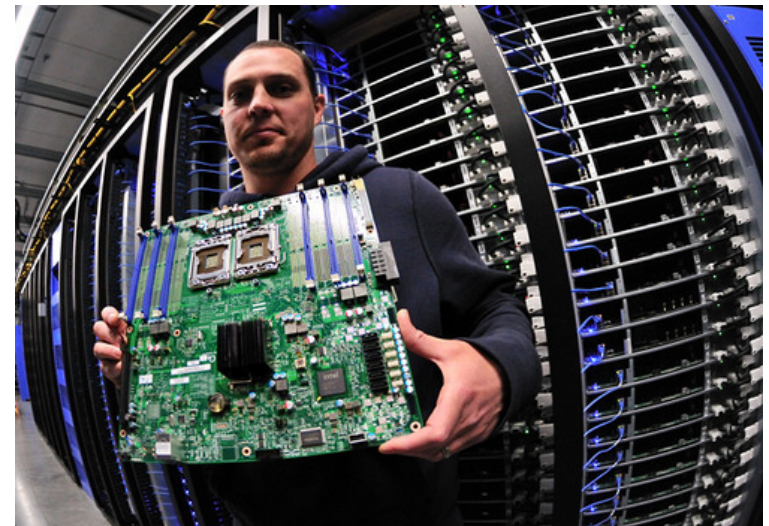
• • •



Oracle Exadata X2-8, 42u
160 cores @ 2.4Ghz, 4TB memory
14 storage servers, 168 cores, 336TB
1.5M database I/O ops/sec
\$1,650,000

Approaches to scaling

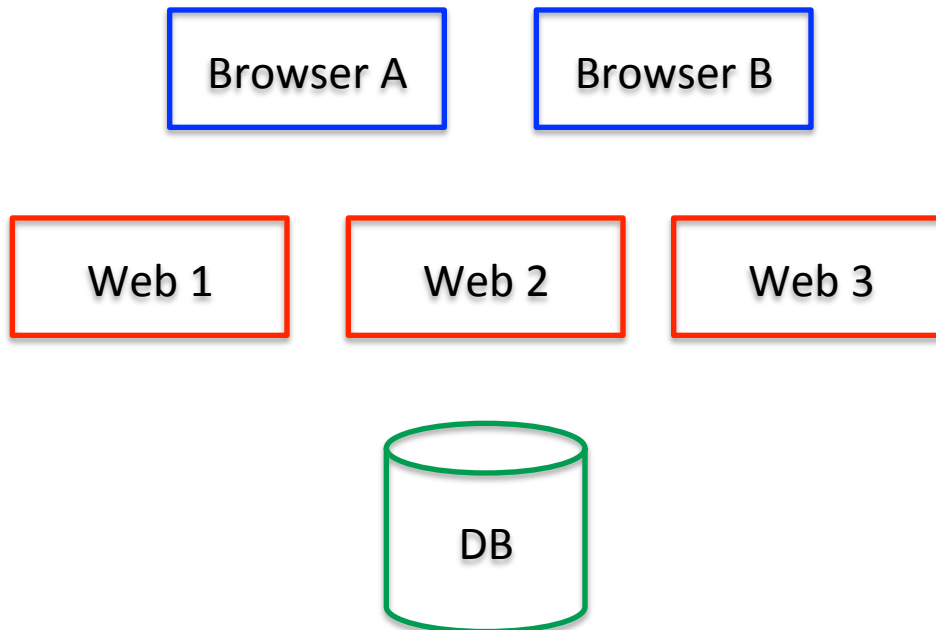
- Horizontal scaling (scale out)
 - Buy more servers
 - Well understood for many parts
 - Application servers (e.g. web servers)
 - But may require software and/or network changes
 - Not so easy for other parts
 - Databases



<http://www.flickr.com/photos/intelfreepress/6722296265/>

One web site: many servers

- How does the user arrive at a particular server?
 - Does the session need to "stick" to same web server?
 - Very important depending on how you manage your state
 - What happens if web server crashes?
 - User would prefer geographically close server



Round robin DNS

- Round robin DNS

- Multiple IP addresses assigned to a single domain name
- Client's networking stack chooses which to connect to

Browser A

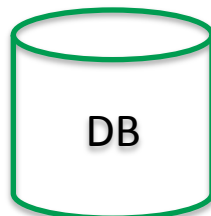
Browser B

```
Queries
  cnn.com: type A, class IN
    Name: cnn.com
    Type: A (Host address)
    Class: IN (0x0001)
Answers
  + cnn.com: type A, class IN, addr 157.166.226.25
  + cnn.com: type A, class IN, addr 157.166.226.26
  + cnn.com: type A, class IN, addr 157.166.255.18
```

Web 1
157.166.226.25

Web 2
157.166.226.26

Web 3
157.166.255.18



Round robin DNS

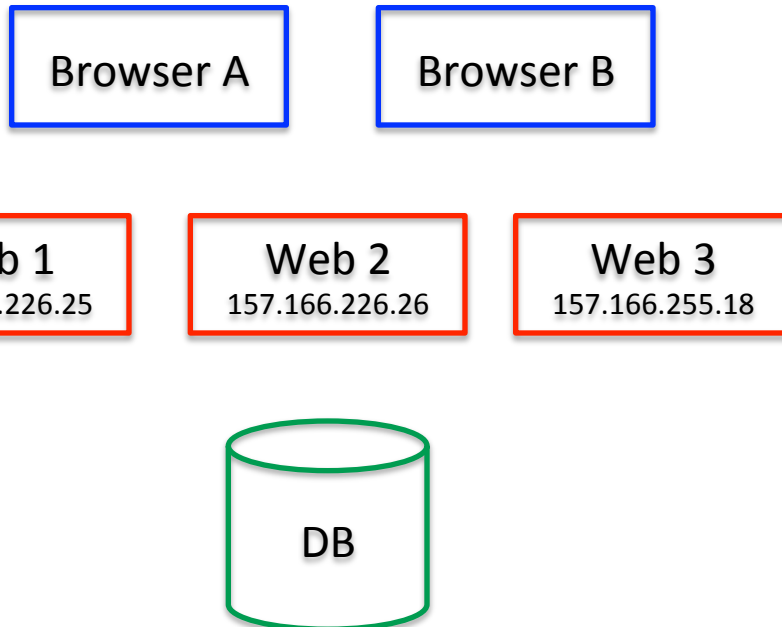
- Round robin DNS

- Simple and cheap to implement

- No specialized hardware, using existing DNS infrastructure

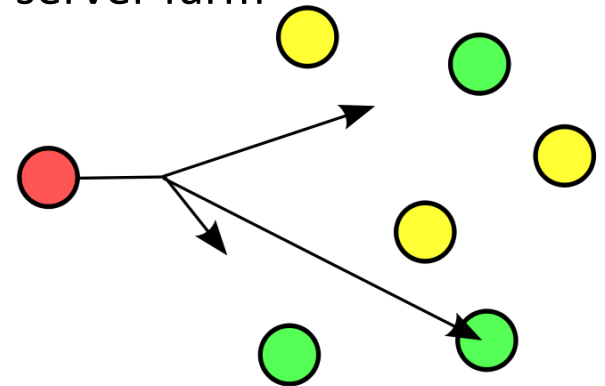
- But DNS has not visibility into actual load on servers

- Or whether a server's hard drive has crashed



Anycast + DNS

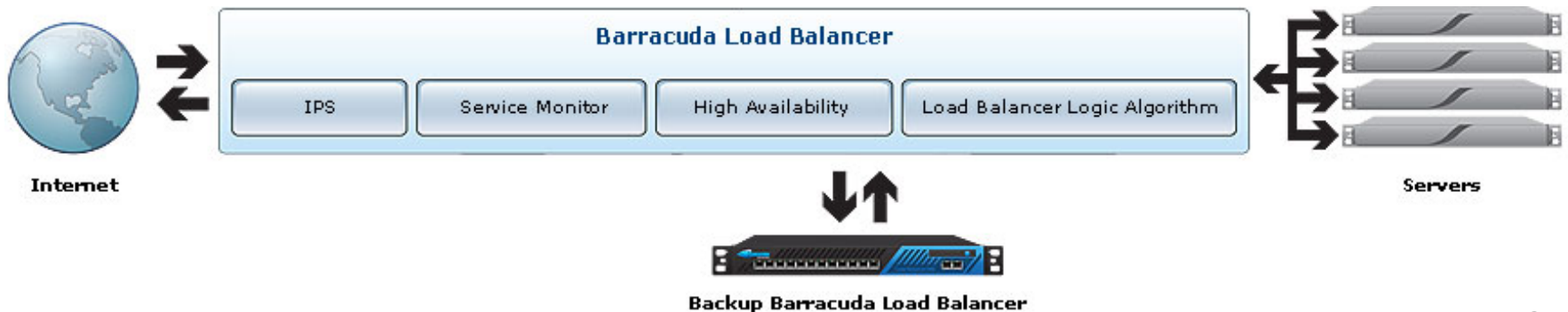
- Getting users to a closer server
- Multiple clusters
 - Each cluster has different pool of IP addresses
 - Place a DNS server next to each cluster
 - Each has same IP address via IP Anycast
 - DNS gives out IP addresses of servers in its cluster
 - Anycast routes client to "closest" DNS server
 - That DNS servers routes client to "closest" server farm



Load balancers

- Load balancers

- Hardware or software (e.g. mod_proxy_balancer, Varnish)
- Like a NAT device in reverse
 - People hit a single public IP to get to multiple private IP addresses
- Introduces a new single point of failure
 - But can introduce a backup balancer
 - Monitor each other via a heartbeat
- How to distribute load?



Load balancer, some features

- Session persistence
 - Getting user back to same server (e.g. via cookie/client IP)
- Asymmetric load
 - Some servers can take more load than others
- SSL offload
 - Load balancer terminates the SSL connection
- HTTP compression
 - Reduce bandwidth using gzip compression on traffic
- Caching content
- Intrusion prevention, DDoS protection

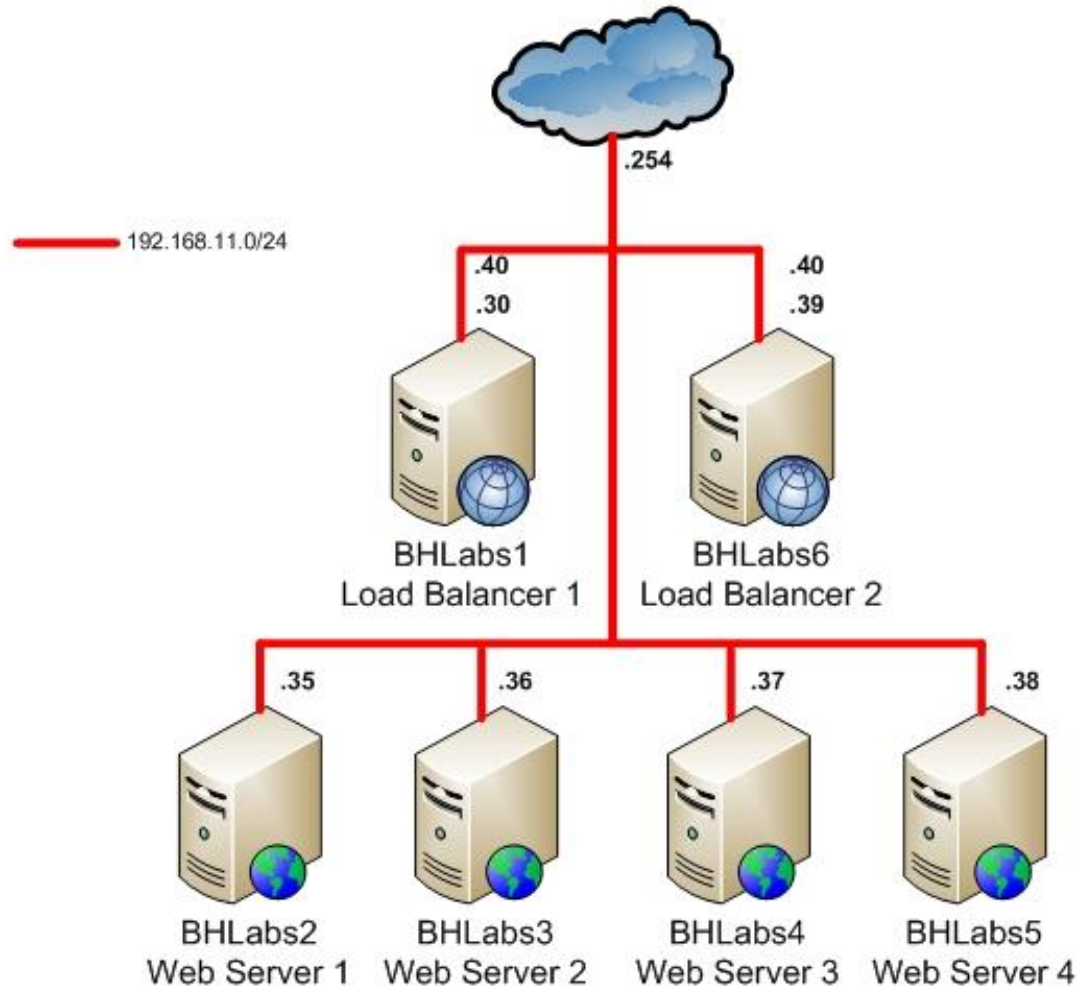
Software load balancer

- Apache server running mod_proxy_balancer
 - One server answers user requests
 - Distributes to two or more other servers

```
<Proxy balancer://mycluster>
BalancerMember http://192.168.1.50:80
BalancerMember http://192.168.1.51:80
</Proxy>
ProxyPass /test balancer://mycluster
```

```
Header add Set-Cookie "ROUTEID=.%
{BALANCER_WORKER_ROUTE}e; path=/"
env=BALANCER_ROUTE_CHANGED
<Proxy balancer://mycluster>
BalancerMember http://192.168.1.50:80 route=1
BalancerMember http://192.168.1.51:80 route=2
ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass /test balancer://mycluster
```

Redundant load balancers



<http://www.centric-it.com/2009/04/29/how-to-quickly-setup-a-load-balanced-high-availability-apache-cluster/>

State management

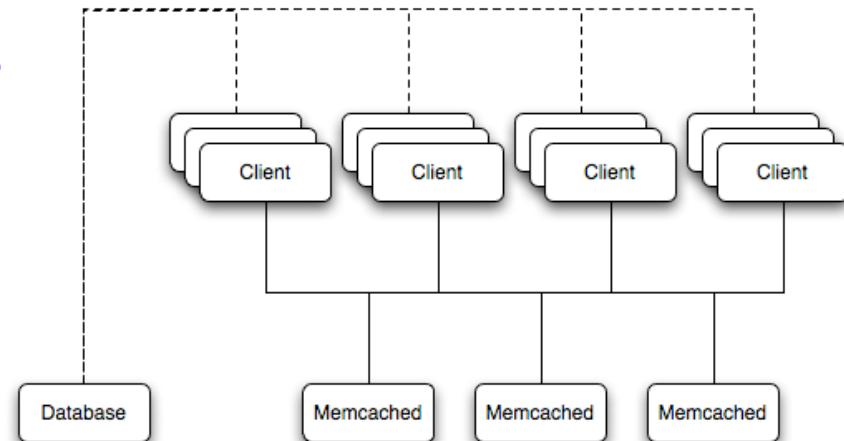
- HTTP is stateless, but user interactions are stateful
- Store session state somewhere:
 - Local to web server
 - Centralized across servers
 - Stored in the client
 - Some combination
 - Centralized but cached at closer level(s)

Local sessions

- **Stored on disk**
 - PHP temp file somewhere
- **Stored in memory**
 - Faster
 - PHP:
 - Compile with `--with-mm`
 - `session.save_handler=mm` in `php.ini`
- **Problems:**
 - User can't move between servers
 - User session won't survive server failure

Centralized sessions

- User can move freely between servers
 - But always need to pull info from central store
- Web servers can crash
 - User gets routed to another web server
- Shared file system
- Store in the database
- Store in an in-memory cache
 - Memcached



No sessions

- Put all information in the cookie
- Browser "nodes" scale with your users
- Concerns:
 - User may delete cookie
 - User may modify cookie
 - Limits on amount of data
 - Local to the browser, user may use multiple browsers

Database scaling

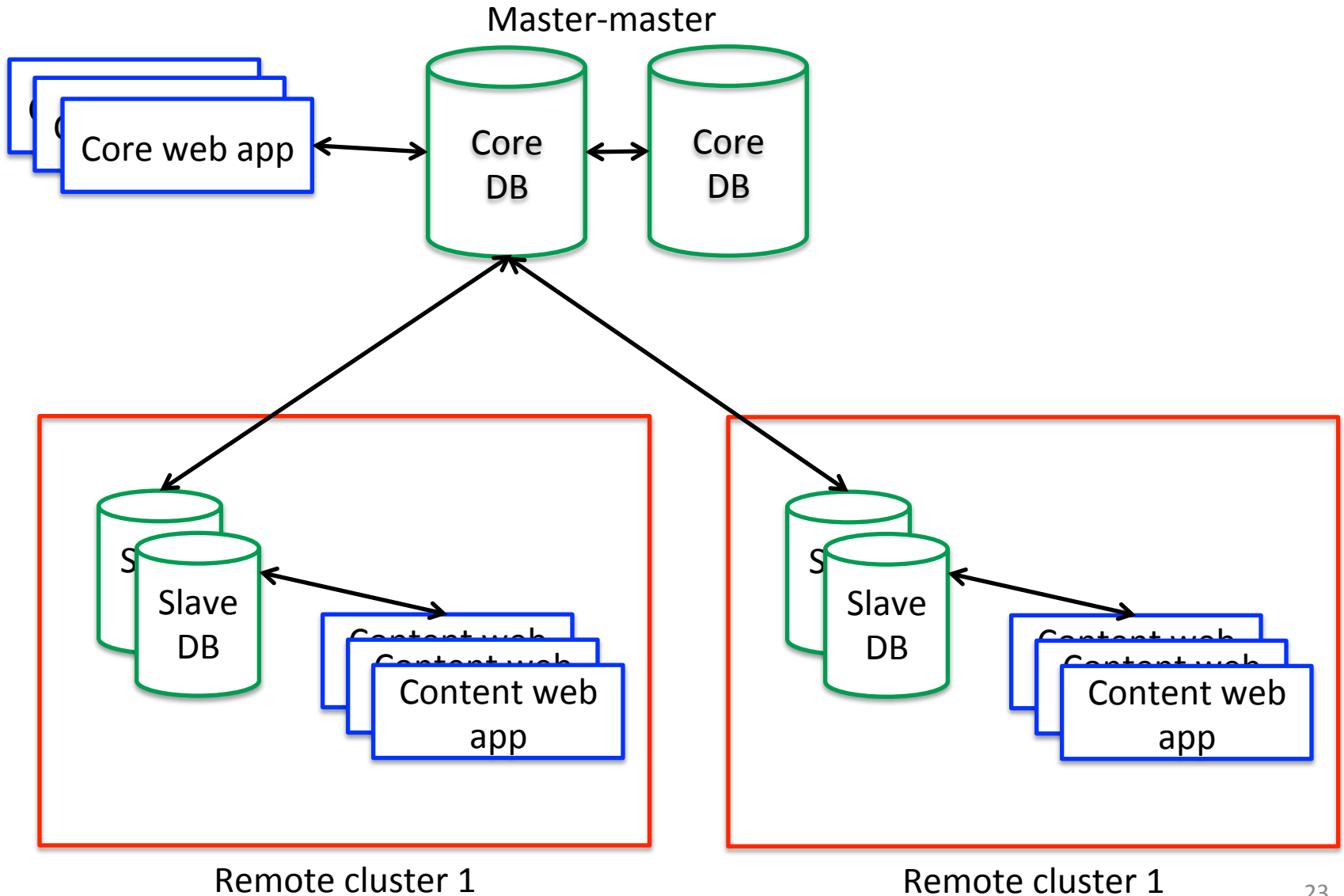
- **Scaling databases is hard**
 - Distribute among many servers and maintain performance
 - DB must obey ACID principles:
 - Atomicity - transactions are all or none
 - Consistency - transactions go from one valid state to another
 - Isolation - no transaction can interfere with another one
 - Durability - on failure, information must be accurate up to the last committed transaction
 - ACID isn't too hard/expensive on a single machine:
 - Shared memory, interthread/interprocess synchronization, shared file system
 - Facilities are fast and reliable
 - Distribute over a LAN or WAN, performance problems!

Database replication

- **Multimaster replication**
 - The "holy grail" of distributed databases
 - Group of DBs, updates can occur on any DB
 - Doing this without loosening ACID, very expensive
 - Two-phase commit between all the nodes
- **Master-master replication**
 - For achieving high-availability, not scalability
 - Two servers with a low latency connection
- **Master-slave replication**
 - Modification only occur on master
 - Changes propagated to slaves



Database example



Other database options

- **Horizontal partitioning**
 - Separate rows onto separate tables
 - Spreads read/writes, improve cache locality
- **Vertical partitioning**
 - Split rows into multiple tables with fewer columns
 - Lets queries scan less data
 - Unless you end up needing to do a join across tables
- **Sharding**
 - Separate rows onto separate databases
 - e.g. All customers west of the Mississippi
 - Must determine which shard customer belongs to
 - Queries/transactions involving multiple shards

Summary

- **Scaling web sites**
 - High availability != load balancing
 - Scale vertically
 - Scale horizontally
 - More application servers
 - Balanced via DNS/hardware/software
 - Session management becomes harder
 - Database is usually the big problem
- **Possible paper topic:**
 - Web farms at extreme scale
 - "The Datacenter as a Computer", Google

