Scale-Out Web Application Architectures

Ivan Pepelnjak (ip@ipSpace.net)

ipSpace.net AG

Scale Up or Scale Out?

Transactional databases

Web servers

https://www.evernote.com/pub/ioshints/scaleMatters

Roadmap

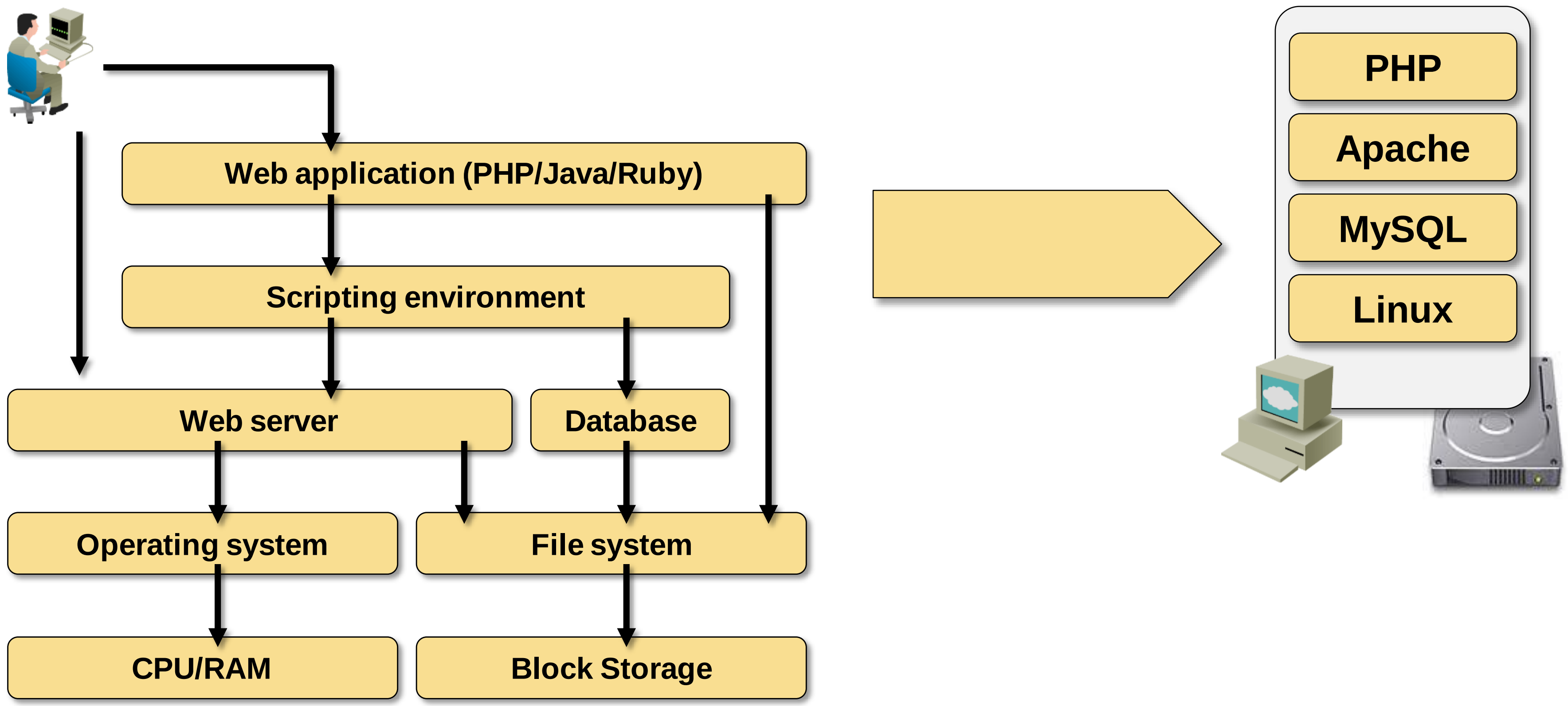
Single server solutions

- Simple single-server application stack
- Scripting offload server
- Standalone database server

Scale-out servers

Multiple data centers

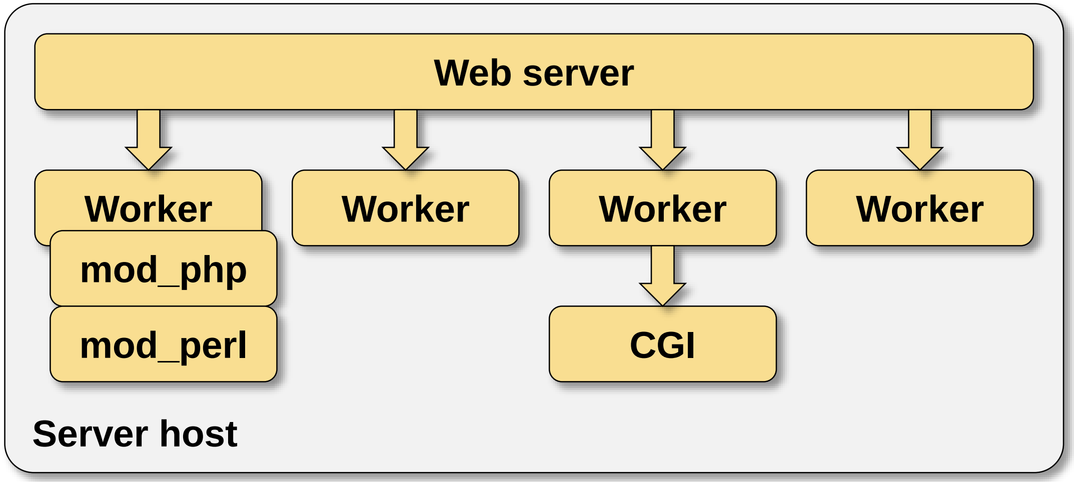
Typical Small Web Application: LAMP Stack



- Web site running on a single server (or VM)
- Local or virtual disk (hopefully with backup)
- Typical web hosting setup

Microsoft: PHP → ASP, Apache → IIS, MySQL → SQL Server, Linux → WinSrv

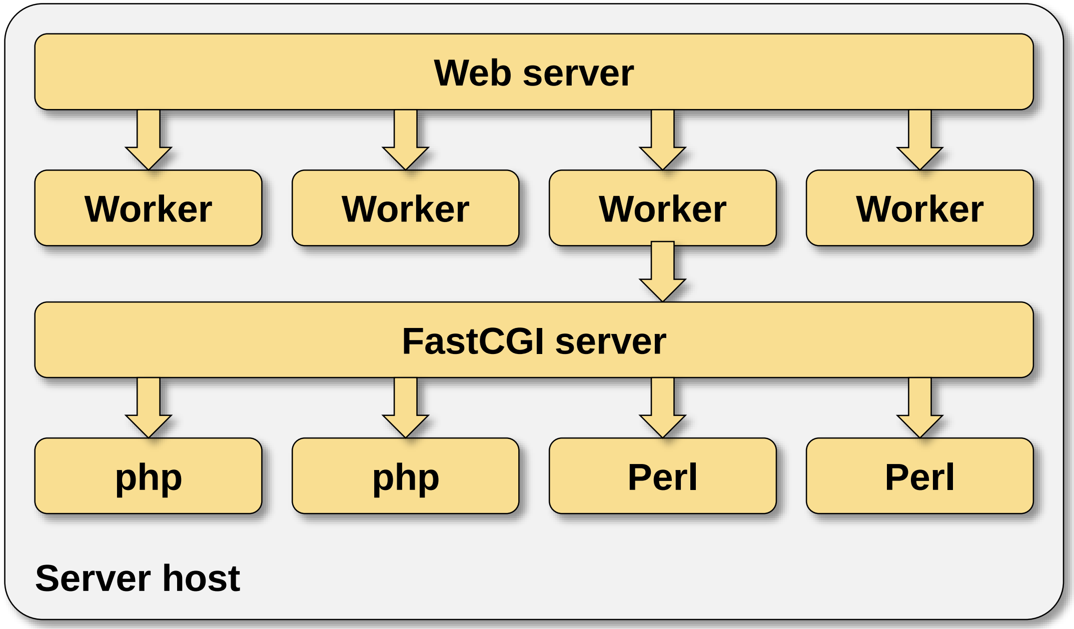
Increased Load → Add Worker Processes



- HTTP requests served by worker processes (process fork)
- All worker processes are identical (and large)
- Scripts processed in worker processes or external programs (CGI)
- Client request blocks a worker process (or a thread)
- Persistent session occupies a worker process for a long time

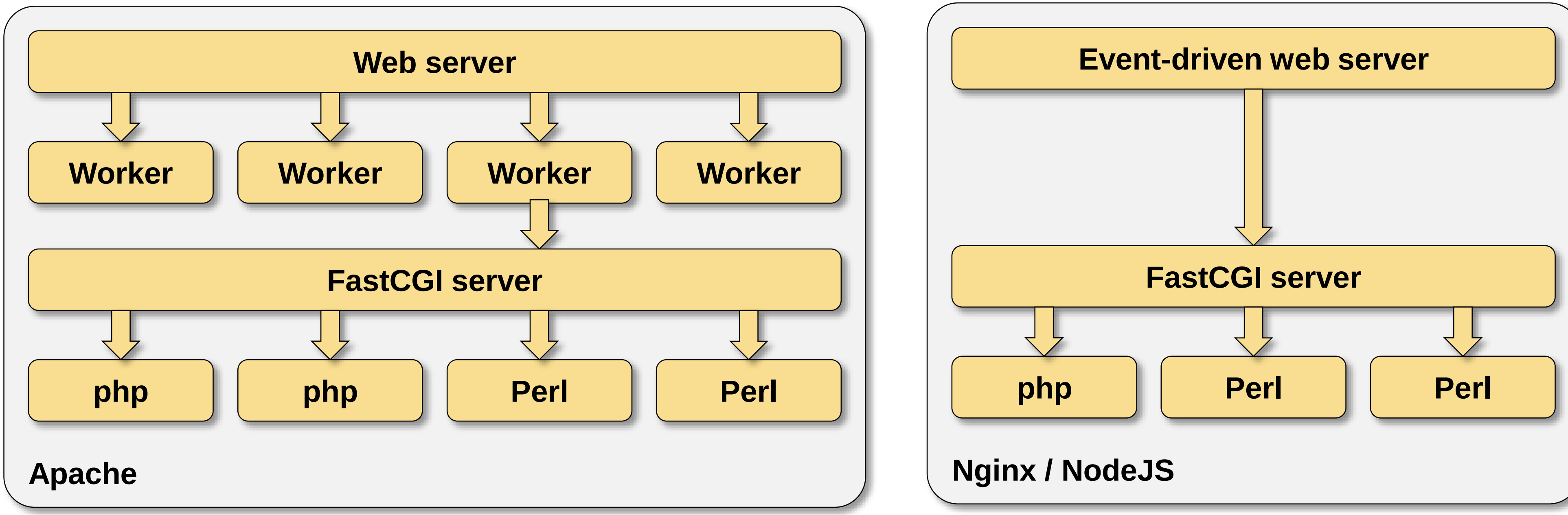
High-volume web sites hate persistent sessions

Optimize Worker Processes: FastCGI



- Web server worker processes serve simple (static) requests
- Script processing offloaded to a different server
- Script output buffered in the worker process
- Client requests and persistent sessions no longer block script workers

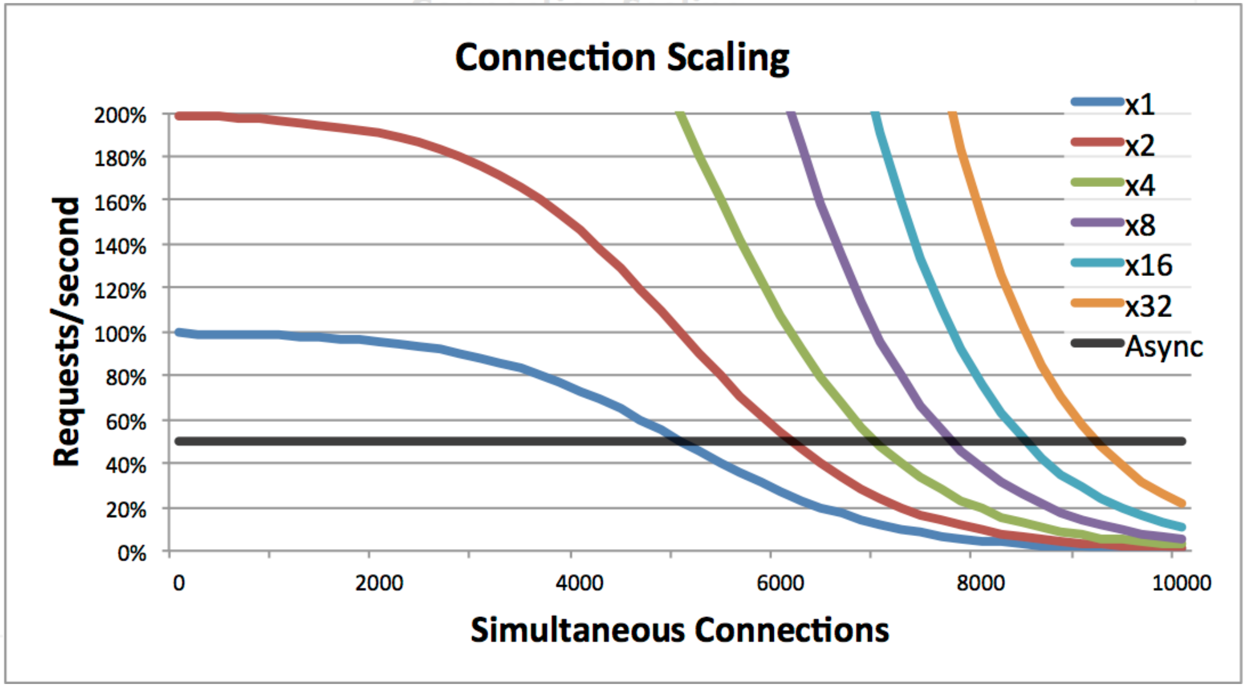
Optimize Web Server: Apache ➔ Nginx



- Worker-based web servers are connection-bound
- Throwing faster CPU @ worker-based web server won't increase the maximum number of connections (kernel locking limits concurrency)
- Event-driven web servers are bandwidth- not connection-bound ➔ Consistent behavior under heavy load

Note: IIS is very similar to Apache (no FastCGI support though)

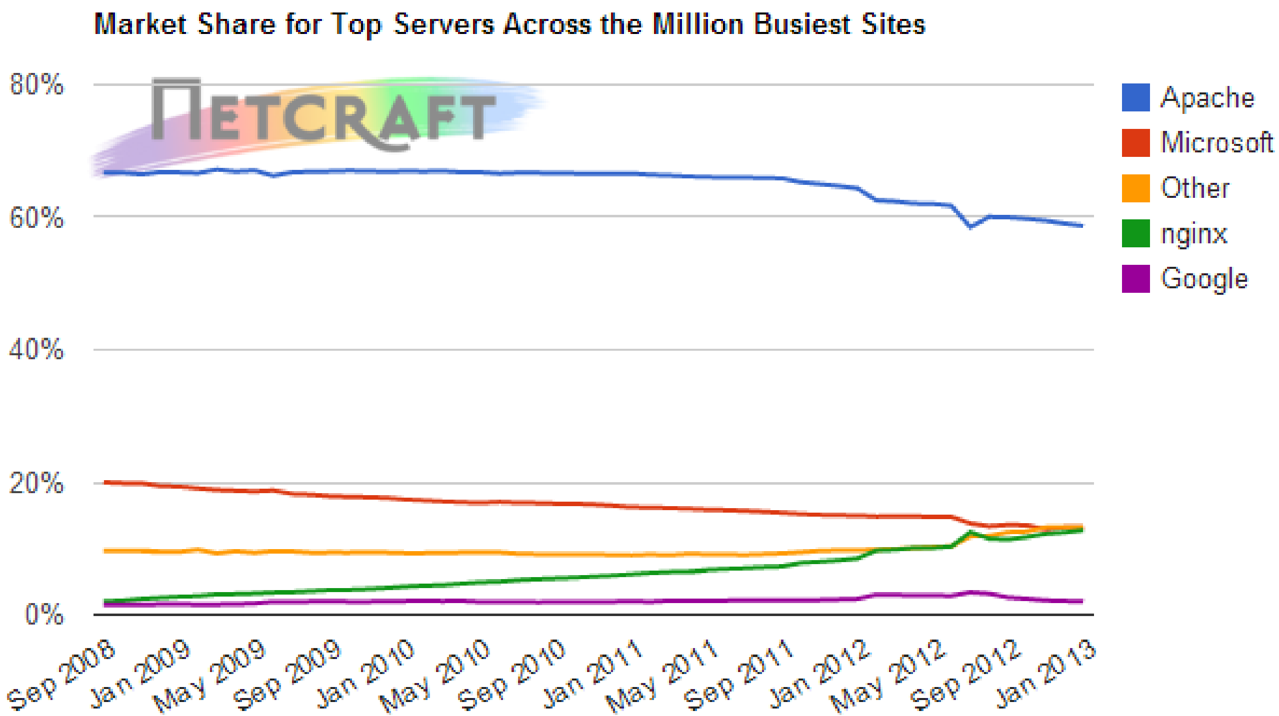
Apache Versus Nginx



- Apache has a problem with large number of concurrent connections
- Adding more CPU does not help much
- Nginx has consistent performance

Source: http://erratasec.blogspot.com/2012/10/scalability-is-systemic-anomaly.html

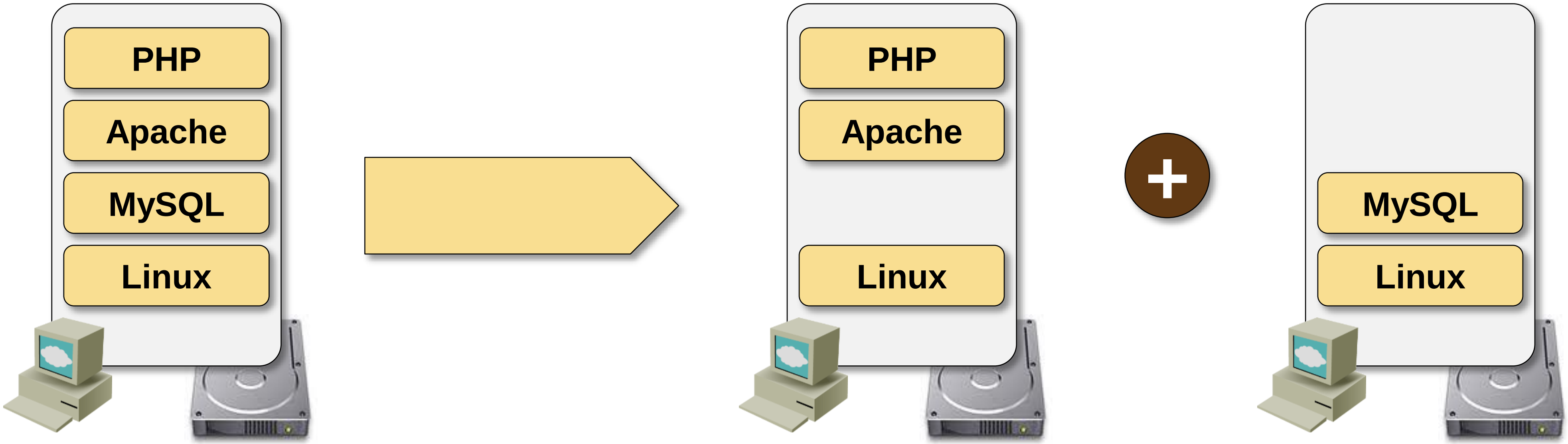
Result: Apache Is Losing Market Share



Developer	December 2012	Percent	January 2013	Percent	Change
Apache	586,594	59.04%	583,143	58.69%	-0.34
Microsoft	131,344	13.22%	131,830	13.27%	0.05
nginx	123,593	12.44%	126,909	12.77%	0.33
Google	20,700	2.08%	19,879	2.00%	-0.08

Source: http://news.netcraft.com/archives/2013/01/07/january-2013-web-server-survey-2.html

Beyond Single Server: Decouple Database Server



- Replace database on web server with a dedicated database server
- Prerequisite for any scale-out application architecture
- Better use of resources
- Multiple web servers can access the same data

Roadmap

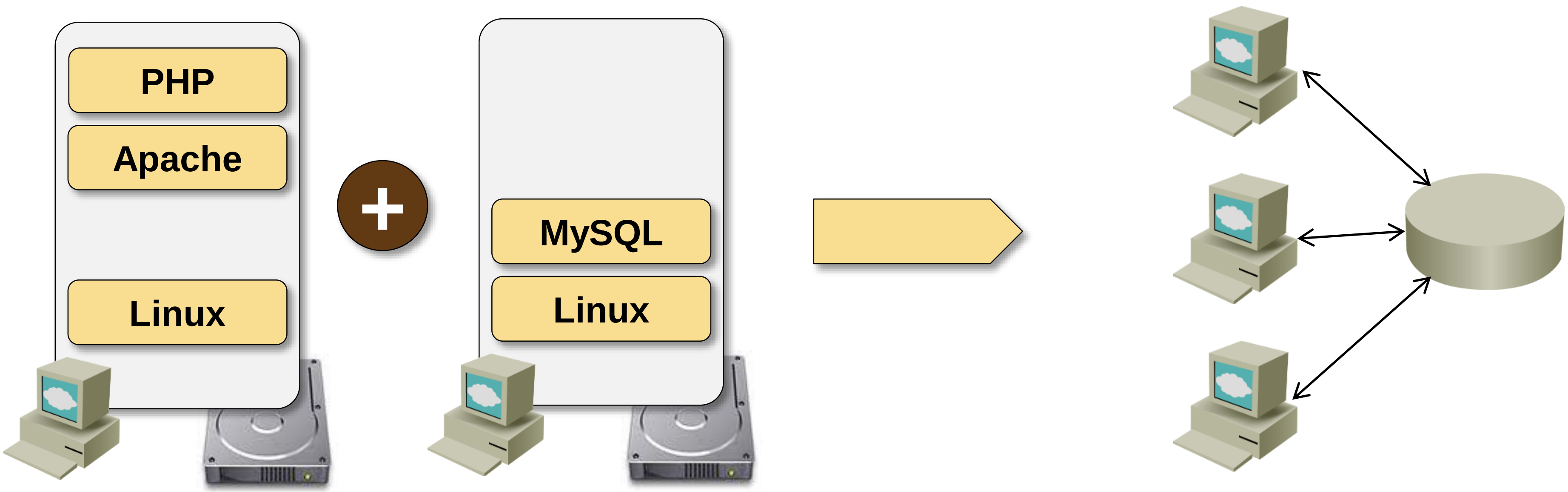
Single server solutions

Scale-out servers

- **Application-level load balancing**
- Network-based load balancing mechanisms
- Session state management
- Scale-out database servers

Multiple data centers

Further Scale Out: Multiple Web or App Servers



A farm of web servers to spread the load

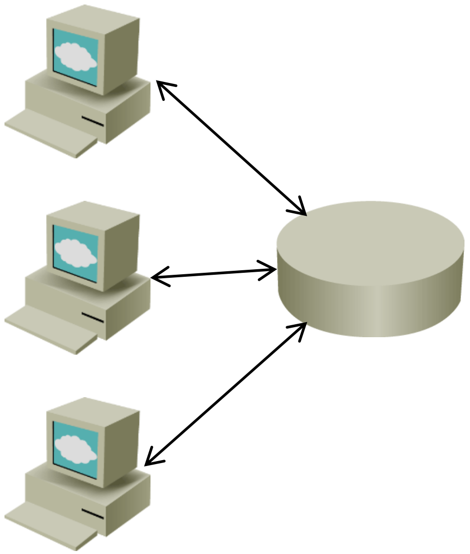
Challenges:

- All servers must appear as the same host name ➔ load balancing
- Application code and configuration files must be synchronized across web servers ➔ single virtual disk image or distributed file system
- Session state management

Load Balancing Architectures

Servers directly connected to the outside network

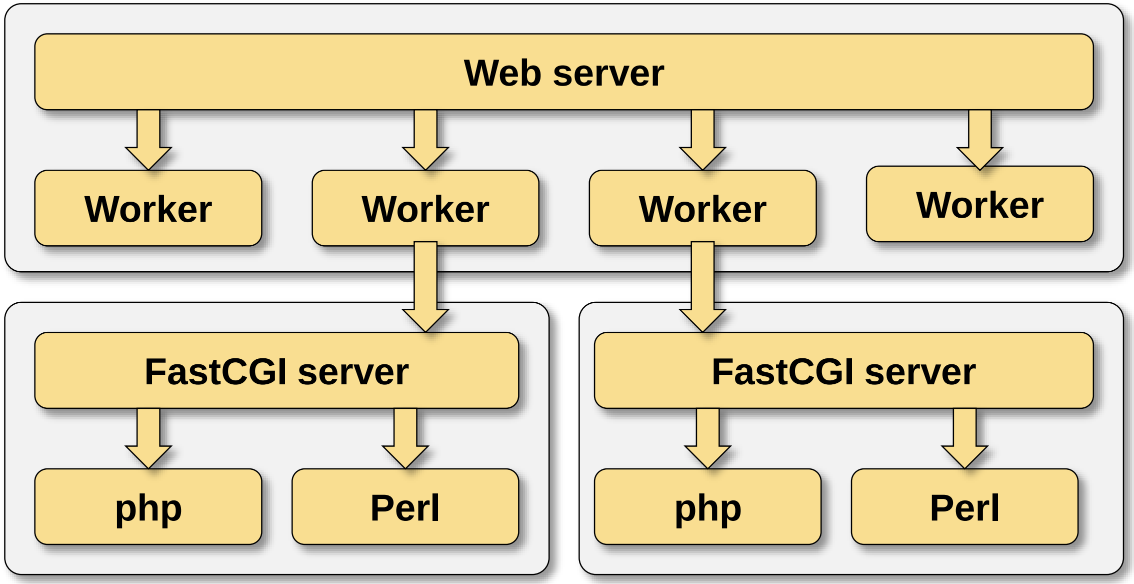
- Multiple independent servers
- Multiple outside IP addresses
- DNS-based load balancing



Load balancing appliance or server

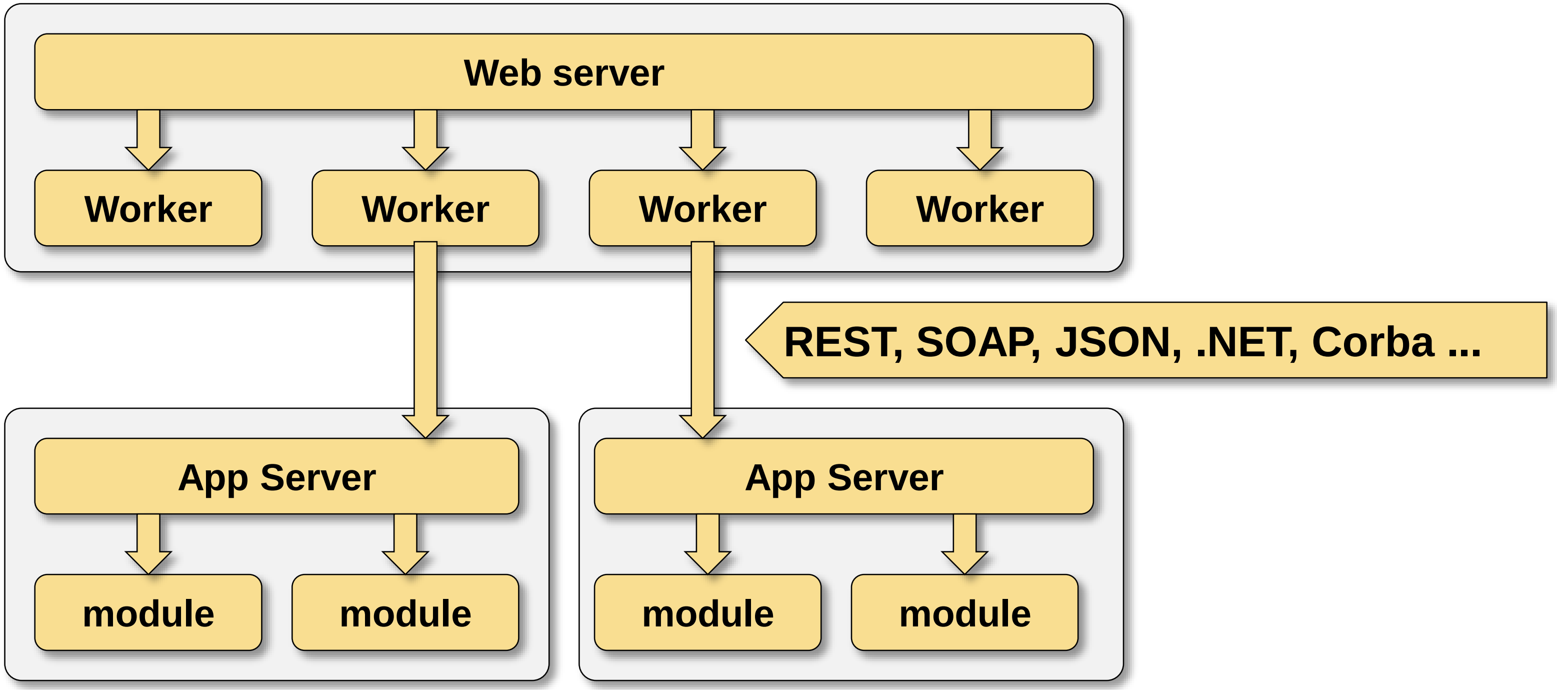
- **Single web server, multiple CGI or app servers**
- **Single caching server, multiple web servers**
- TCP or HTTP load balancing

Load Balancing with FastCGI Offload



- FastCGI works over TCP → you can separate web and app servers
- FastCGI server selection based on URL path → per-application servers
- FastCGI server selection based on suffix → language-specific servers
- Multiple FastCGI servers (nginx, lighthttpd) → application-level load balancing

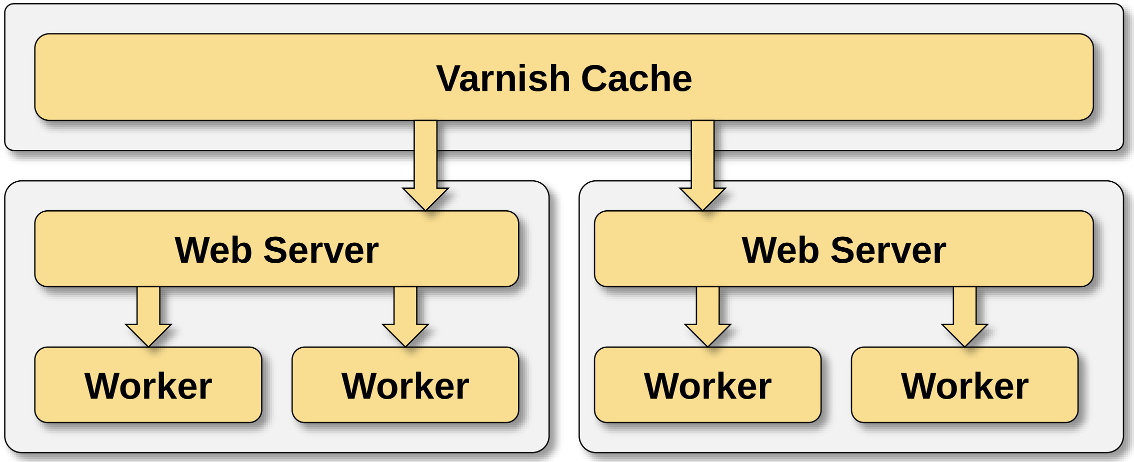
Load Balancing with Application Servers



- Architecturally similar to FastCGI offload
- FastCGI script receives headers from original HTTP request
- App server receives HTTP request from web worker process ➔ a layer of isolation

Network-based load balancer might be needed between web and app servers

Load Balancing with Reverse Proxy



- Reverse proxy (front-end cache) can use multiple physical servers for a single HTTP hostname

Challenges:

- Load balancing mechanism
- Session state persistence (sticky sessions)
- Original client IP address is lost
- SSL/TLS client certificate might be lost

Roadmap

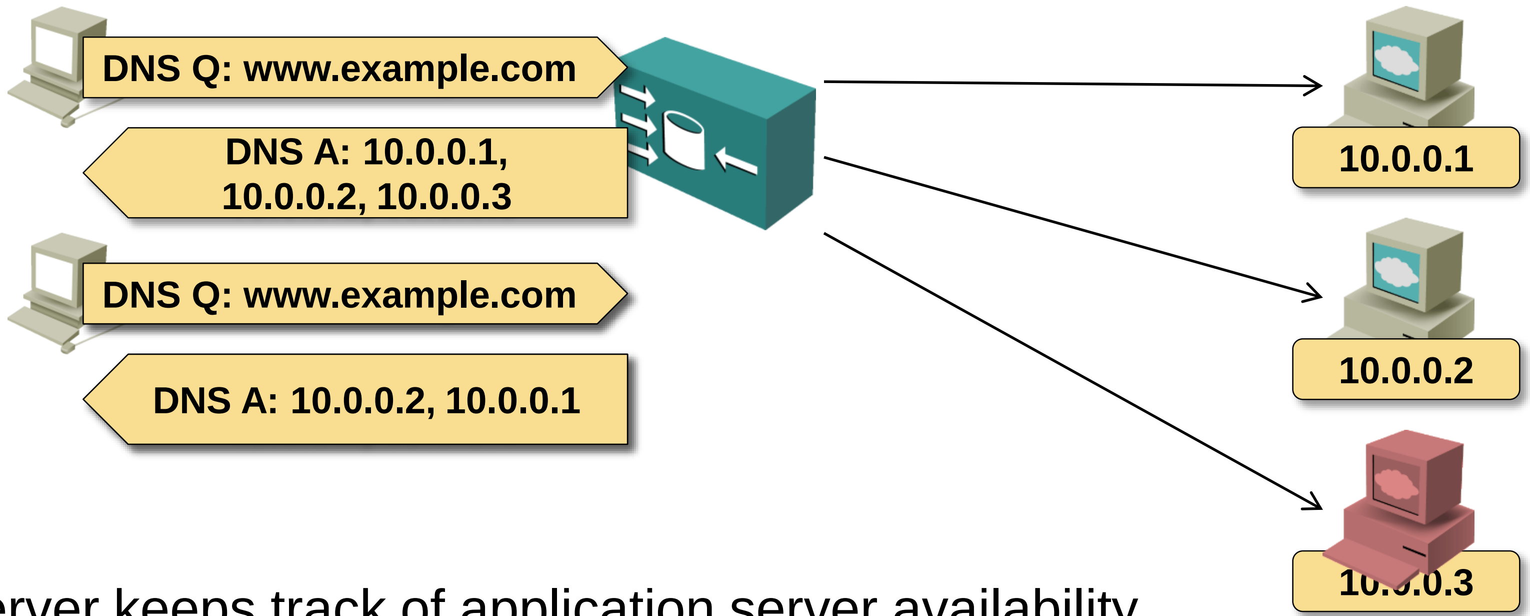
Single server solutions

Scale-out web or app servers

- Application-level load balancing
- **Network-based load balancing mechanisms**
- Session state management
- Scale-out database servers

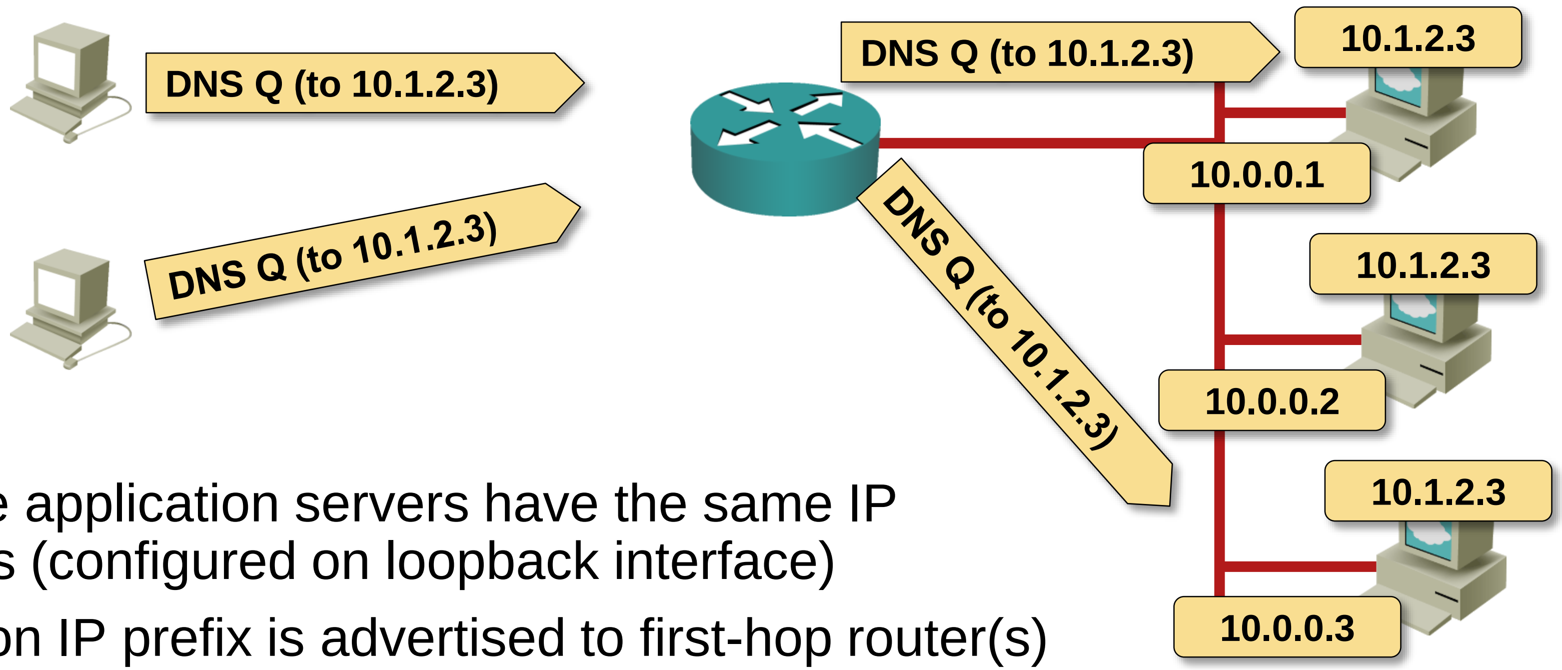
Multiple data centers

DNS-based Local Load Balancing



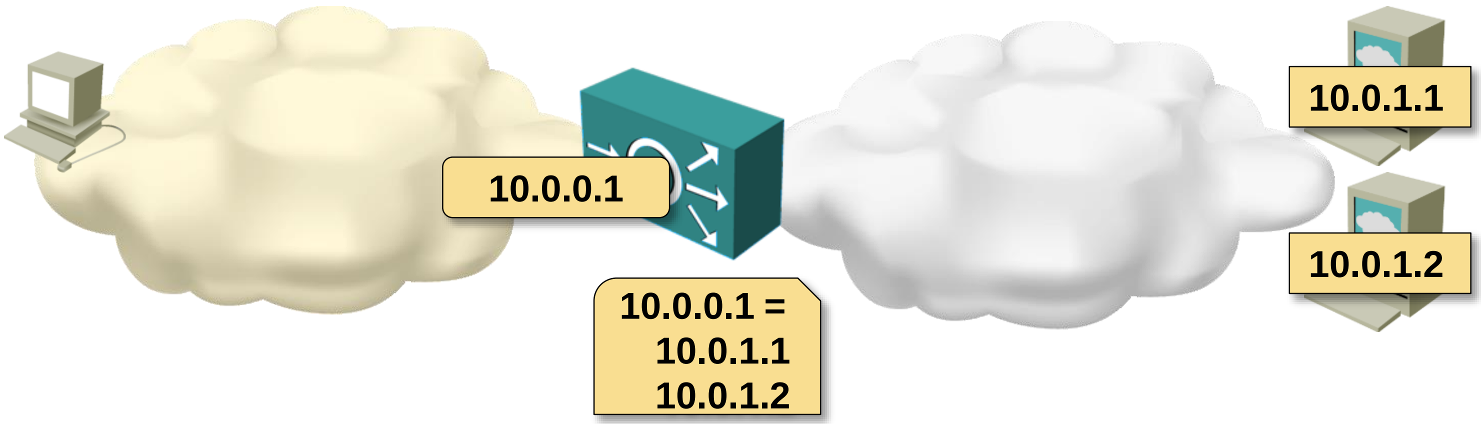
- DNS server keeps track of application server availability
- Random list of addresses of all available servers is sent in DNS responses
- Low TTL times used to remove unavailable servers from the list
- Works reasonably well for non-critical applications that rely on DNS
- Web browsers don't work well due to DNS pinning ➔ use in combination with high-availability features (IP address sharing)

Local Anycast Load Balancing



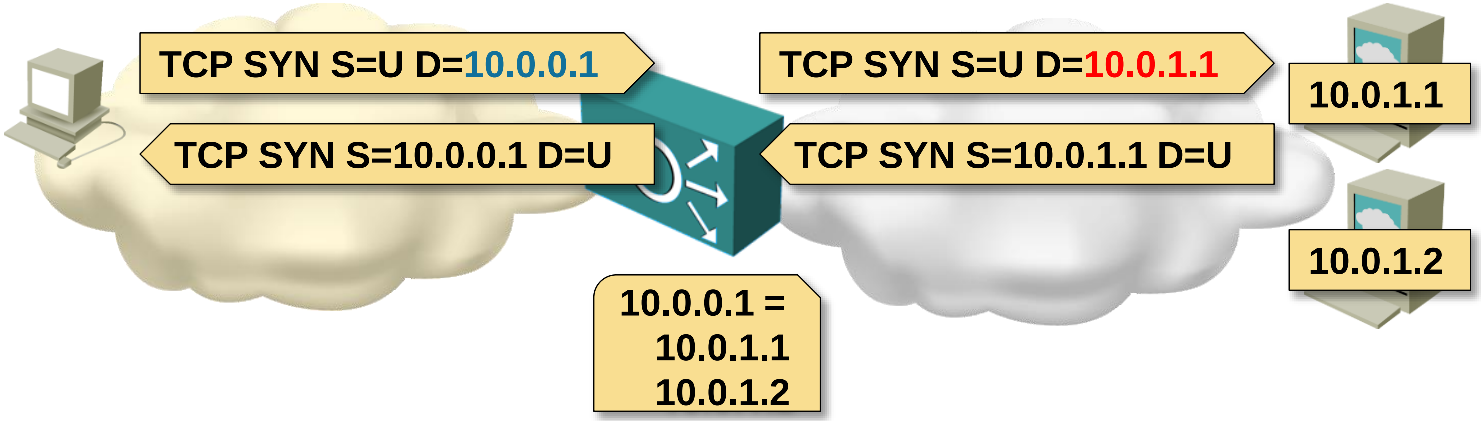
- Multiple application servers have the same IP address (configured on loopback interface)
- Common IP prefix is advertised to first-hop router(s)
  ➔ routing protocol running on servers
  ➔ static routes on first-hop routers
- 5-tuple load balancing available in most routers spreads the load
- Every change in server availability changes the load balancing tables
  ➔ useful only for UDP traffic
  ➔ heavily used in high-volume DNS environments

Load Balancers – Principles



- Every service has one or more virtual IP addresses (and/or ports)
- Service is associated with a pool of servers
- Load balancer constantly checks the servers' health and responsiveness
- Clients connect to the virtual IP address, load balancer maps the request to the *best* server in the pool

Load Balancers – Operations



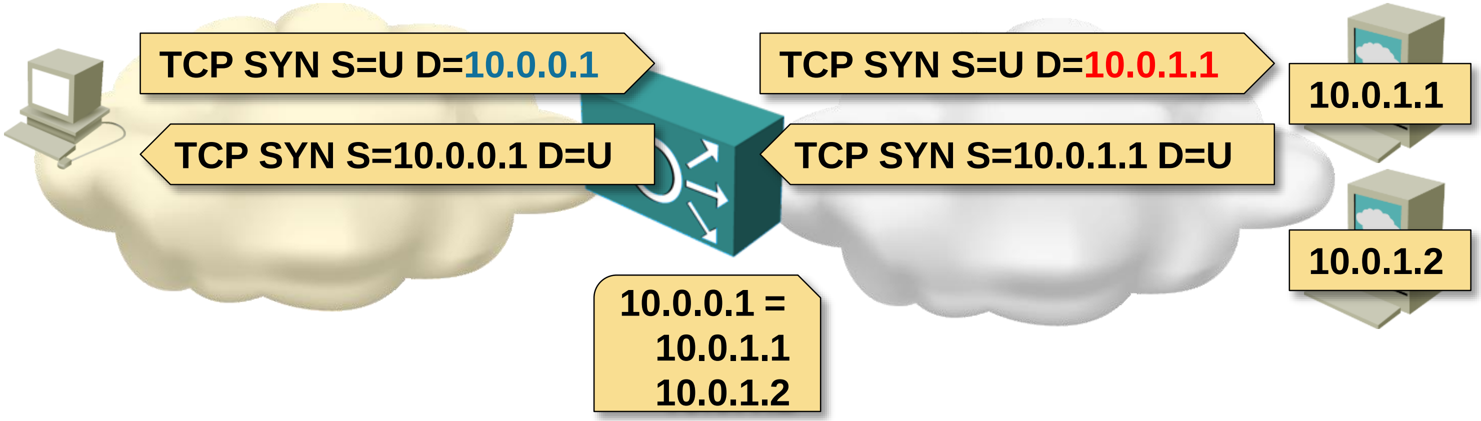
Control plane

- Monitor the health of *inside* servers (from ping to application-level requests)
- Track the server load (number of sessions or responsiveness)

Data plane

- Select the “best” *inside* server for a new session (incl. *stickiness*)
- Use NAT and/or two TCP sessions
- Optional: adjust/rewrite the content

Load Balancers – Transparent Mode

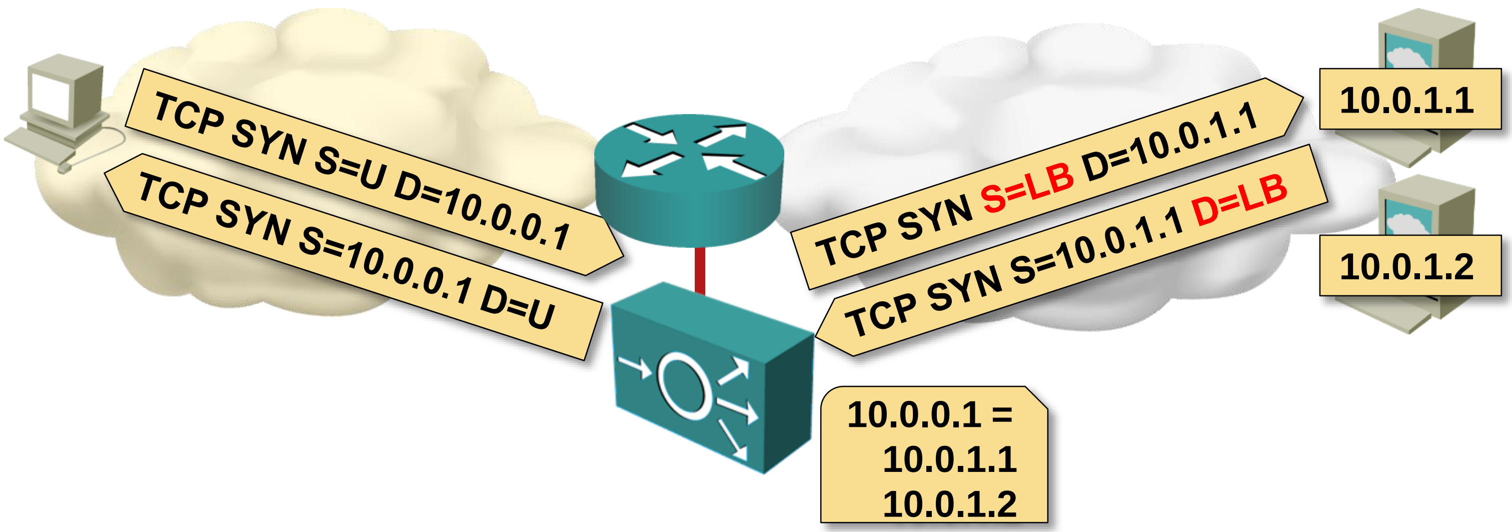


Load balancer is transparent to the clients and servers

Destination-only NAT:

- Virtual server IP address is replaced with real IP address of selected server
- Client IP address is not changed
  ➔ logging, address-based access control or geolocation work
- Reverse traffic must flow through the load balancer
  ➔ load balancer must be in the data path
- NAT is required for IPv4 (SLB44) and IPv6 (SLB66) load balancing

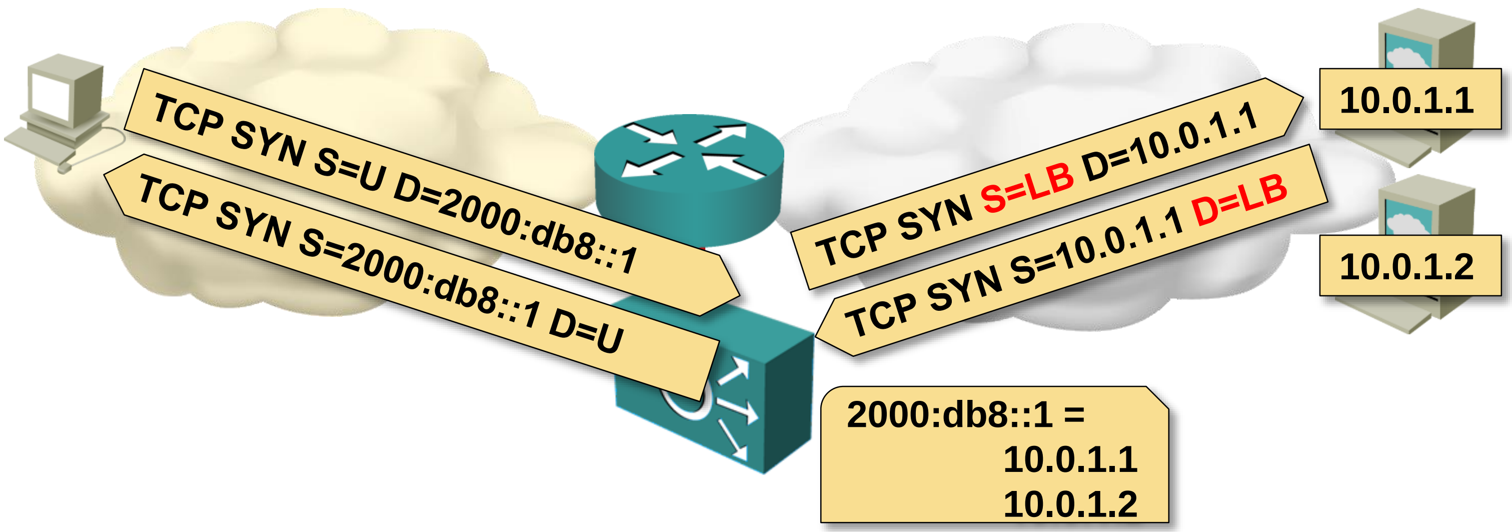
Load Balancers – One Arm Mode



Use when the load balancer is not in the forwarding path

- Source (client) and destination (server) IP addresses are translated
- A pool of inside addresses is assigned to the load balancer
- Client address+port is translated into an address+port assigned to LB pool
- Client IP address is no longer available to the server
  ➔ Use X-Forwarded-For HTTP header
  ➔ Might require SSL offload

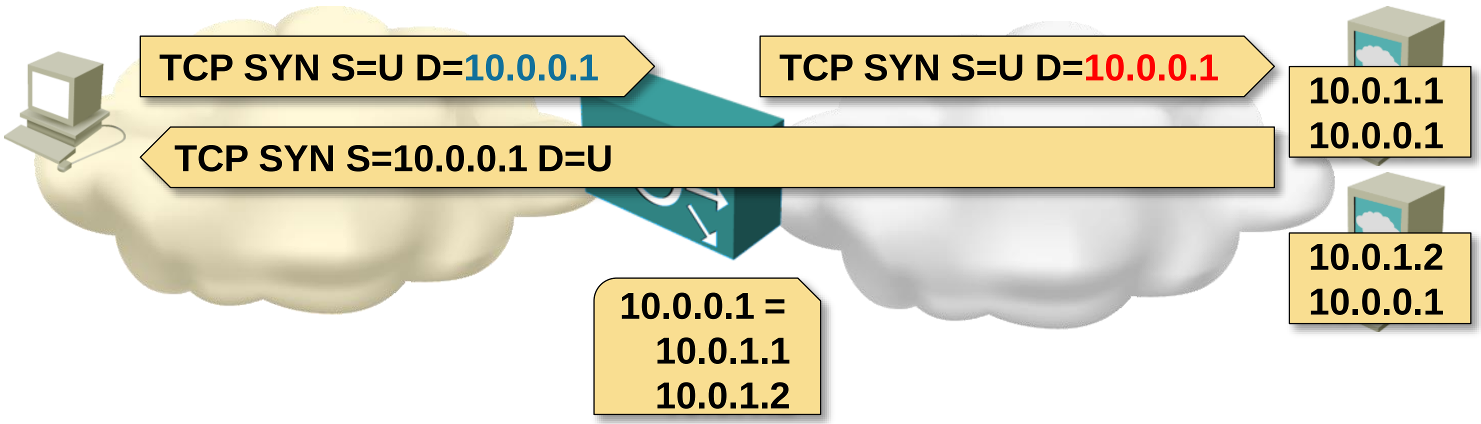
Load Balancers – Protocol Translation (SLB64)



Make IPv4 content available to IPv6 clients

- Virtual IP address = IPv6 address
- Server pool = IPv4 or IPv6 addresses
- Source and destination addresses must be in the same address family
  ➔ Source NAT is mandatory

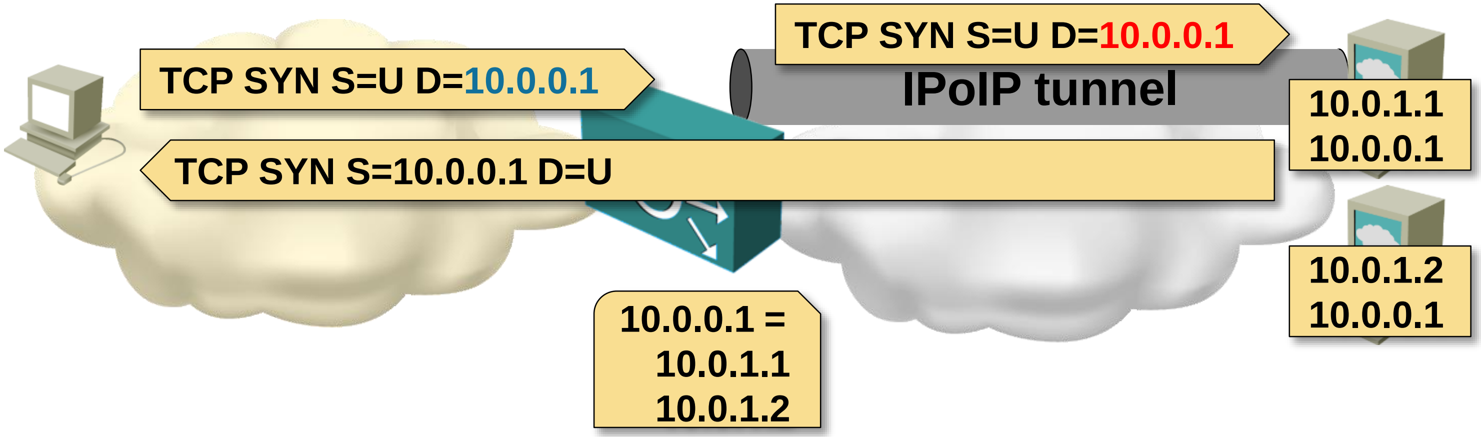
Direct Server Return



- Same IP address configured on all hosts (loopback interfaces)
- LAN IP address used for ARP (host MAC address resolution)
- Load balancer rewrites MAC header only
- Unmodified IP packet sent to selected server
- Server sends a reply packet directly to the client
- Requires L2 connectivity between load balancer and servers

Sample product: Linux Virtual Server (LVS)

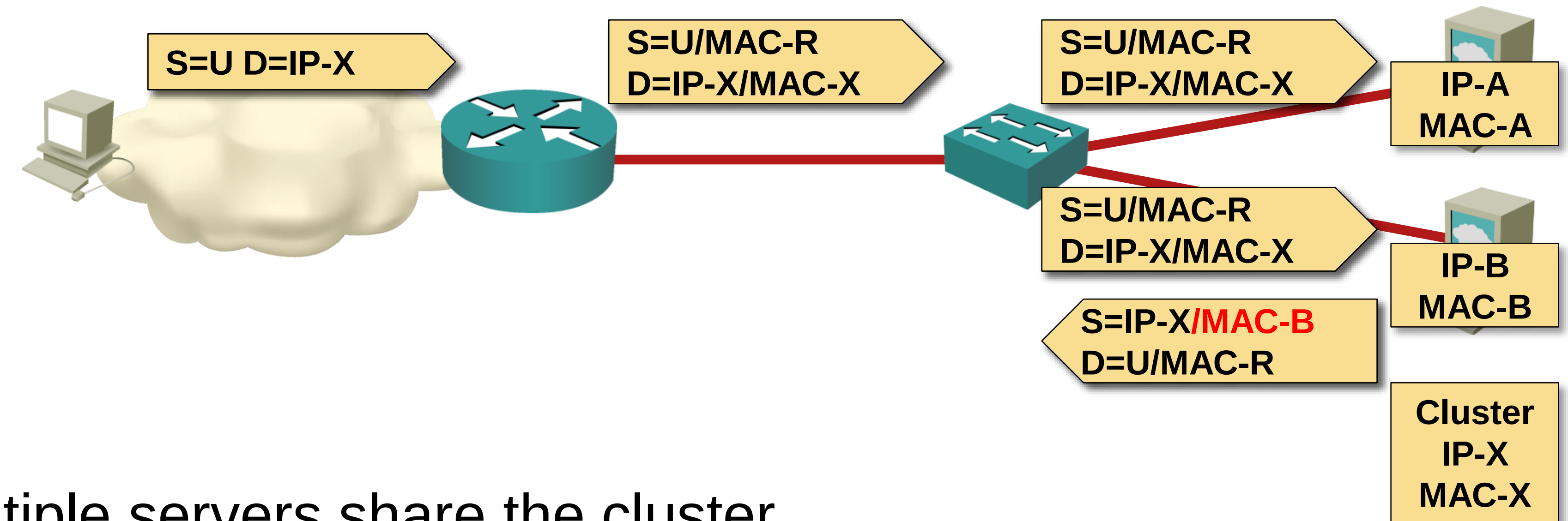
Direct Server Return with IP Tunnel



- Same IP address configured on all hosts (loopback interfaces)
- IP tunnels between load balancer and server(s)
- Load balancer encapsulates client IP packets
- Server sends a reply packet directly to the client
- Works with L3 connectivity between load balancer and servers

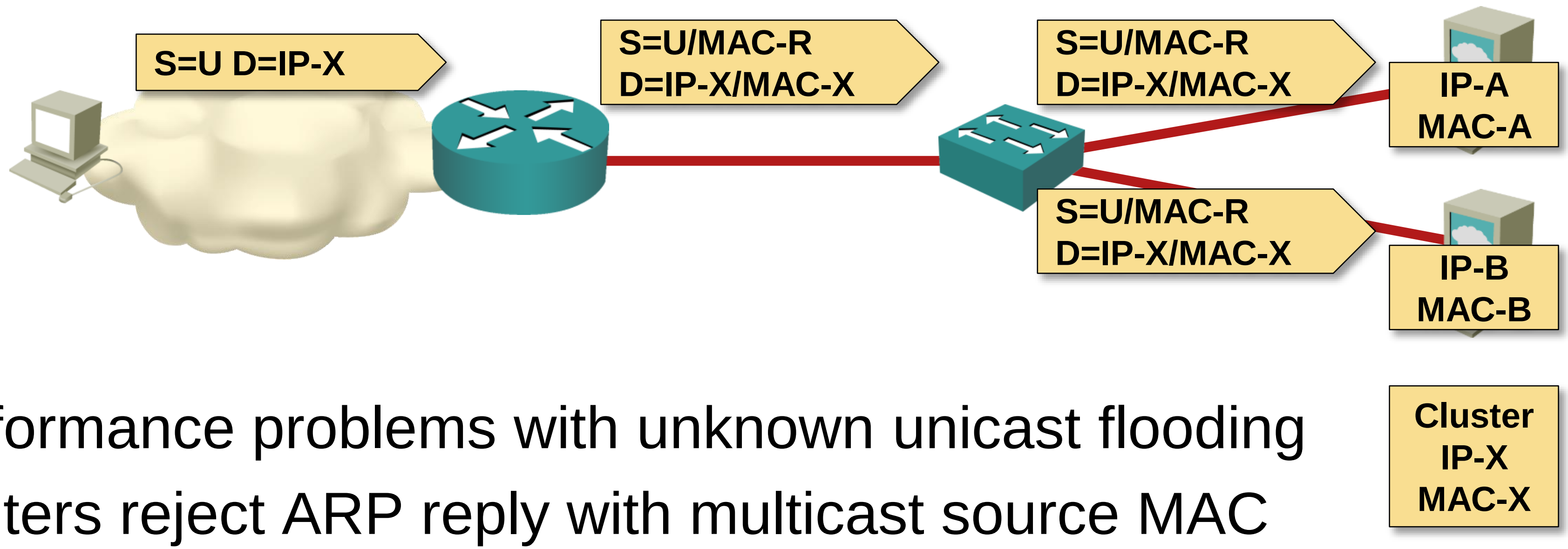
Sample product: Linux Virtual Server (LVS)

Server-Based Network Load Balancers (Microsoft NLB)



- Multiple servers share the cluster IP address
- Bridging tricks are used to send the traffic to all servers
- One of the servers replies to the packet

Microsoft NLB Caveats

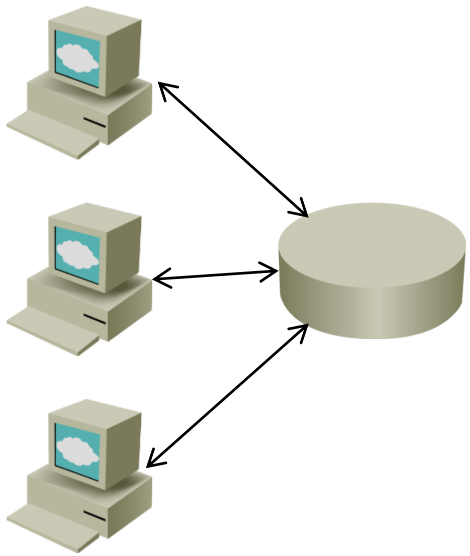


- Performance problems with unknown unicast flooding
- Routers reject ARP reply with multicast source MAC
  → Solve with static ARP
- All servers have to process every incoming packet
  → Unnecessary CPU load
- Every incoming packet is flooded to all the servers
  → Wasted bandwidth

Roadmap

- **Single server solutions**
- **Scale-out web or app servers**
  - Application-level load balancing
  - Network-based load balancing mechanisms
  - **Session state management**
  - Scale-out database servers
- **Multiple data centers**

Web Session Management



Some facts first:

- HTTP requests are stateless
- Almost all scripting environments support *sessions* – state persistence across HTTP requests
- Session ID in cookie or URL
- Session data in memory or on disk

Session management in scale-out architecture:

- Load balancer with persistent (sticky) sessions
  ➔ Requests from the client are always sent to the same server
  ➔ Based on client IP address or session cookie
  ➔ Explosion of state on load balancer
- Session data stored in database or key-value store
- Typical solution: *memcached*

Roadmap

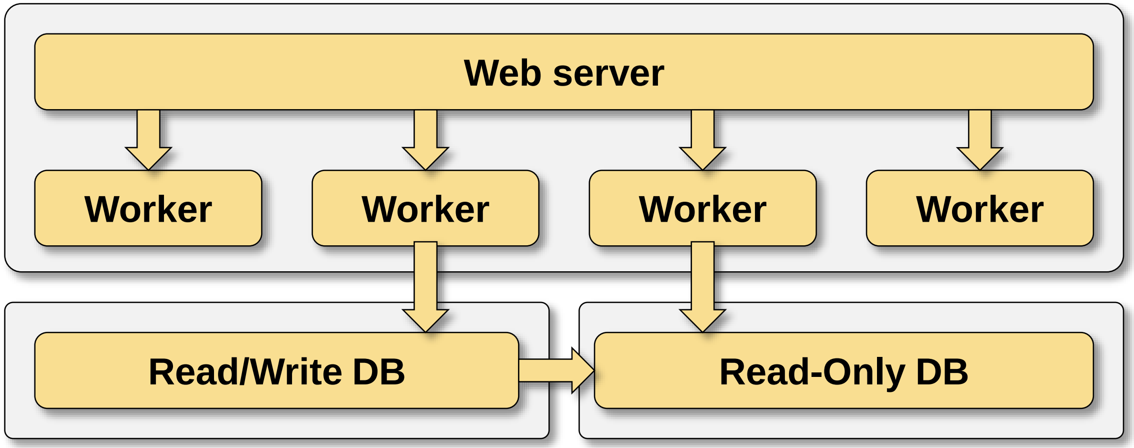
Single server solutions

Scale-out web or app servers

- Application-level load balancing
- Network-based load balancing mechanisms
- Session state management
- **Scale-out database servers**

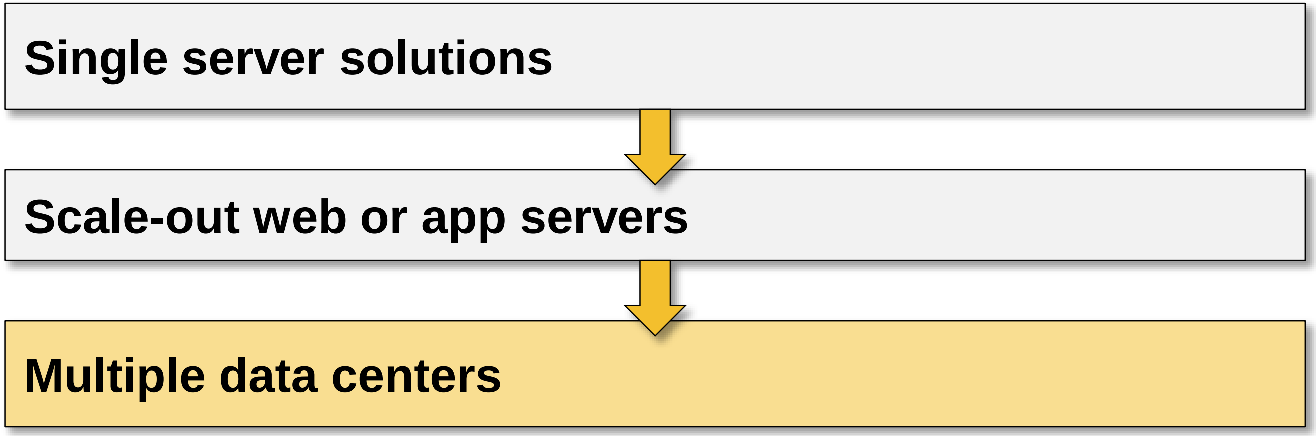
Multiple data centers

Web Applications: Database Load Balancing



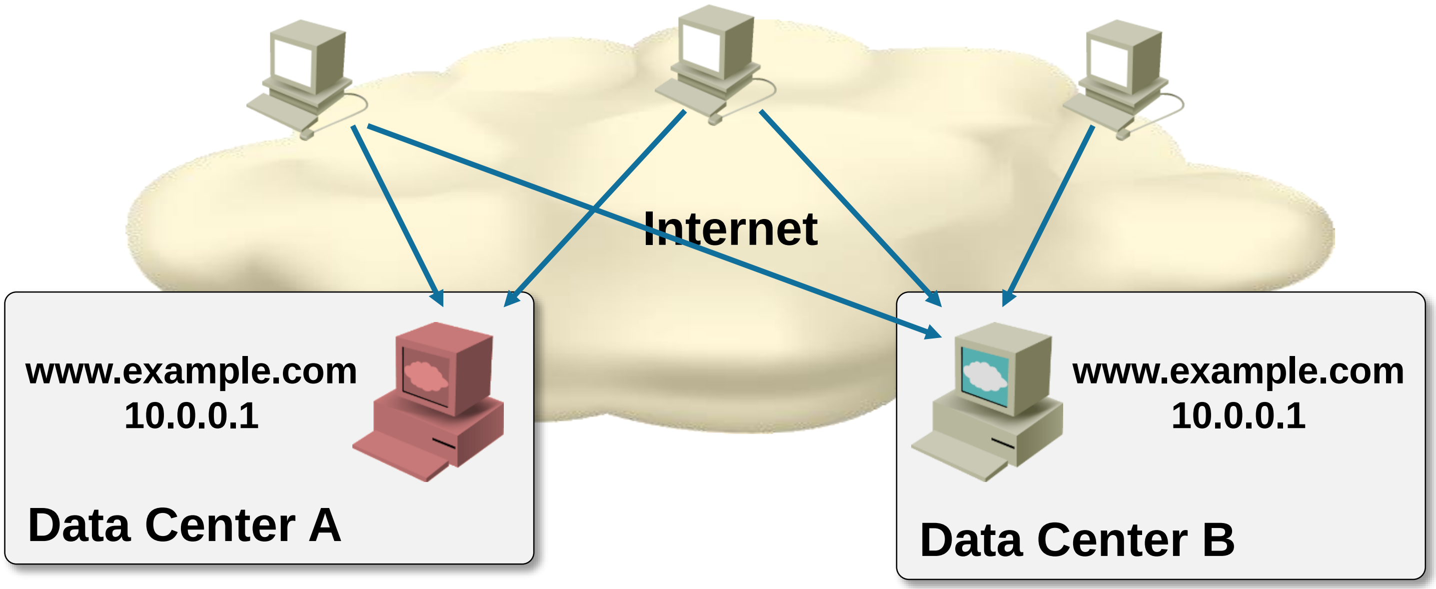
- Single R/W database replica and multiple R/O replicas
- Asynchronous replication (eventual consistency)
- Multiple database connections
- Most scripts access R/O replica(s)
- Solve per-user consistency issues with cookies

Roadmap



Single server solutions
Scale-out web or app servers
Multiple data centers

Global Anycast

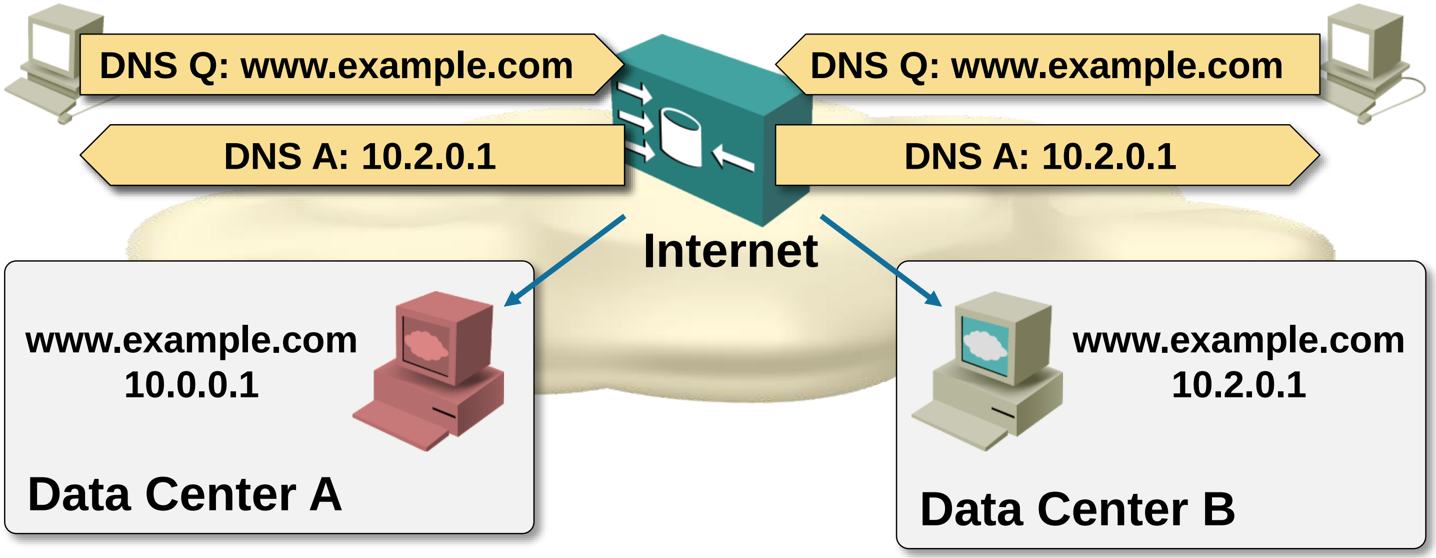


Same IP address is advertised from multiple data centers

Caveats

- Depends exclusively on Internet routing
- Perfect solution for UDP-based services (DNS)
- Quality of TCP-based services depends on network stability and routing distance between data centers

Global DNS-based Load Balancing



DNS responses vary based on user's location, server load and server availability

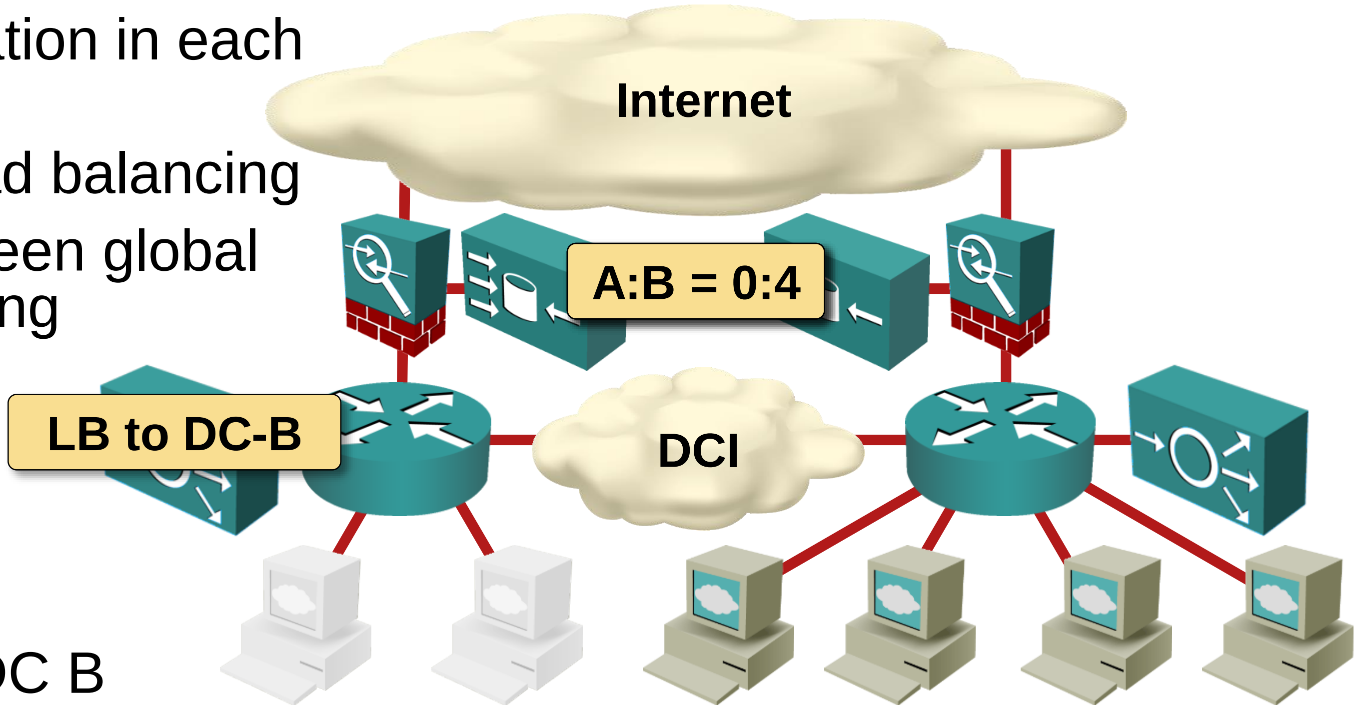
Caveats

- Geolocation based on recursive DNS server's location (not client's)
- Clients usually (but not always) pick the first IP address in the DNS response
- DNS pinning in browsers limits the usability of this solution

Disaster Avoidance With Load Balancing

Prerequisites

- Public VIP per application in each data center
- DNS-based global load balancing
- Synchronization between global and local load balancing



Process

- Graceful shutdown of servers in DC A
- Start new servers in DC B
- Load balancers shift load toward DC B
- No Layer-2 DCI or vMotion required

Conclusions

Look Before You Jump

- Design application with scalability in mind
- Test a sample scale-out architecture (and failure handling)
- Deploy scale-out architecture when needed
- Investigate bottlenecks and fix application before deploying complex scale-out solutions

Questions?

Paperwork issues

- **Follow-up email**
- **Please fill in the evaluation form (waiting in your browser)**
- **Recording available within 24 hours**
- **PDF materials always available for download**
- **Discount for future webinars – use wlp10 discount code**
- **Upgrade to yearly subscription**
- **Please spread the word!**

Send them to ip@ipSpace.net or @ioshints