

CS-310 Scalable Software Architectures

Lecture 7: Load Balancing

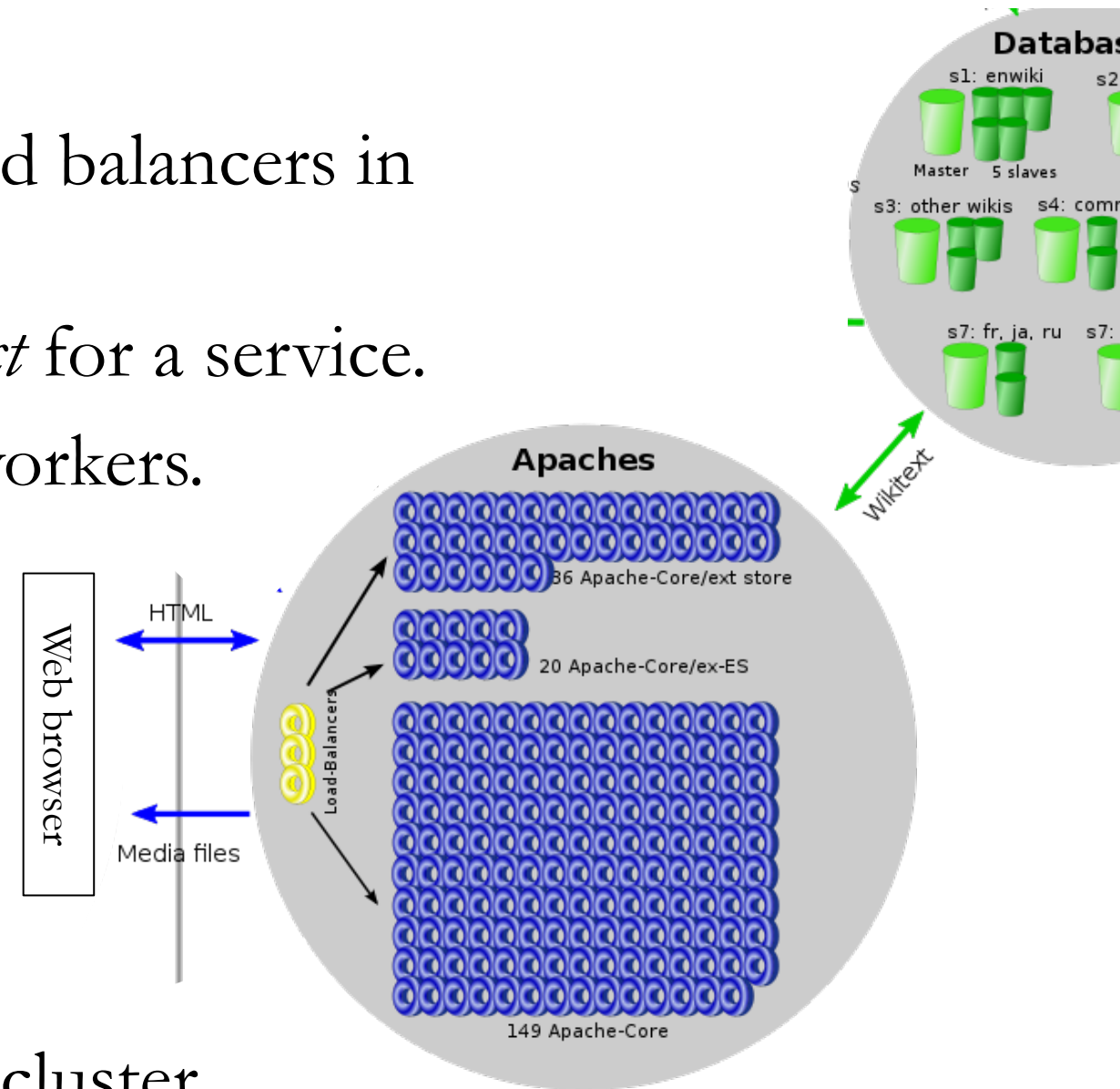
Steve Tarzia

Last Lecture: Microservices, etc.

- Introduced **microservices** as an alternative to **monolithic** design.
- Services are **black boxes**, exposing **network APIs**.
 - Decouples development of different parts of the system.
 - Network APIs define the format and meaning of requests and responses.
- JS **Single-page Applications** (SPAs) interact directly with services.
 - Moves UI concerns away from backend code.
- In a **cross-platform system design**, the same backend service/API can serve mobile, web, and desktop apps.

Load balancers

- In the diagram below, there are 3 load balancers in front of 200 MediaWiki servers.
- Load balancer is a *single point of contact* for a service.
- Requests are *proxied* to a cluster of workers.
- Load balancer does very little work:
 - Just forward request/response and remember the request source.
 - Load balancer can relay requests for 10s-100s of application servers.
- Makes one IP address appear like one huge machine, but it's actually a cluster.

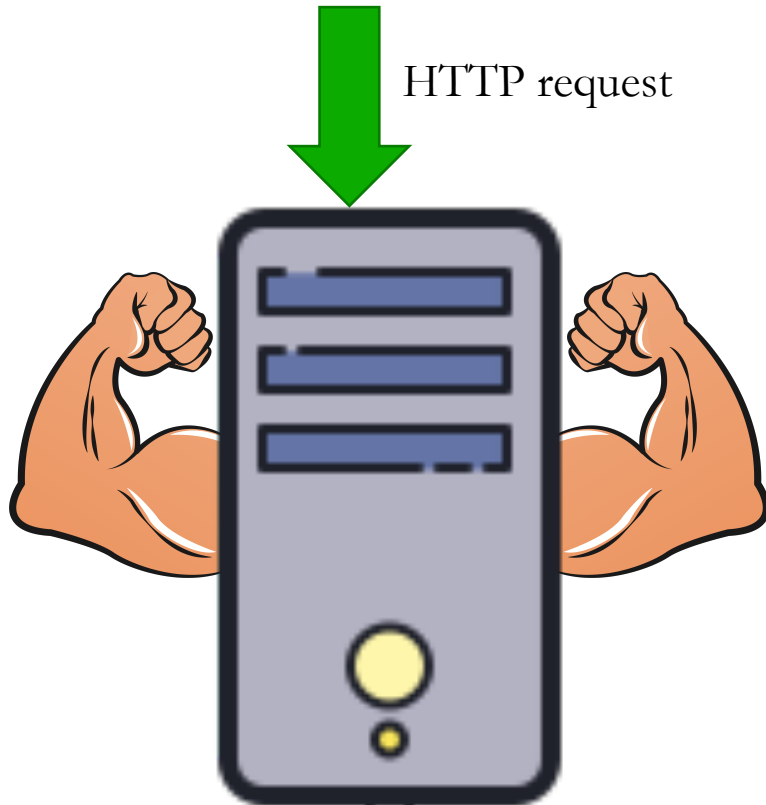


MediaWiki

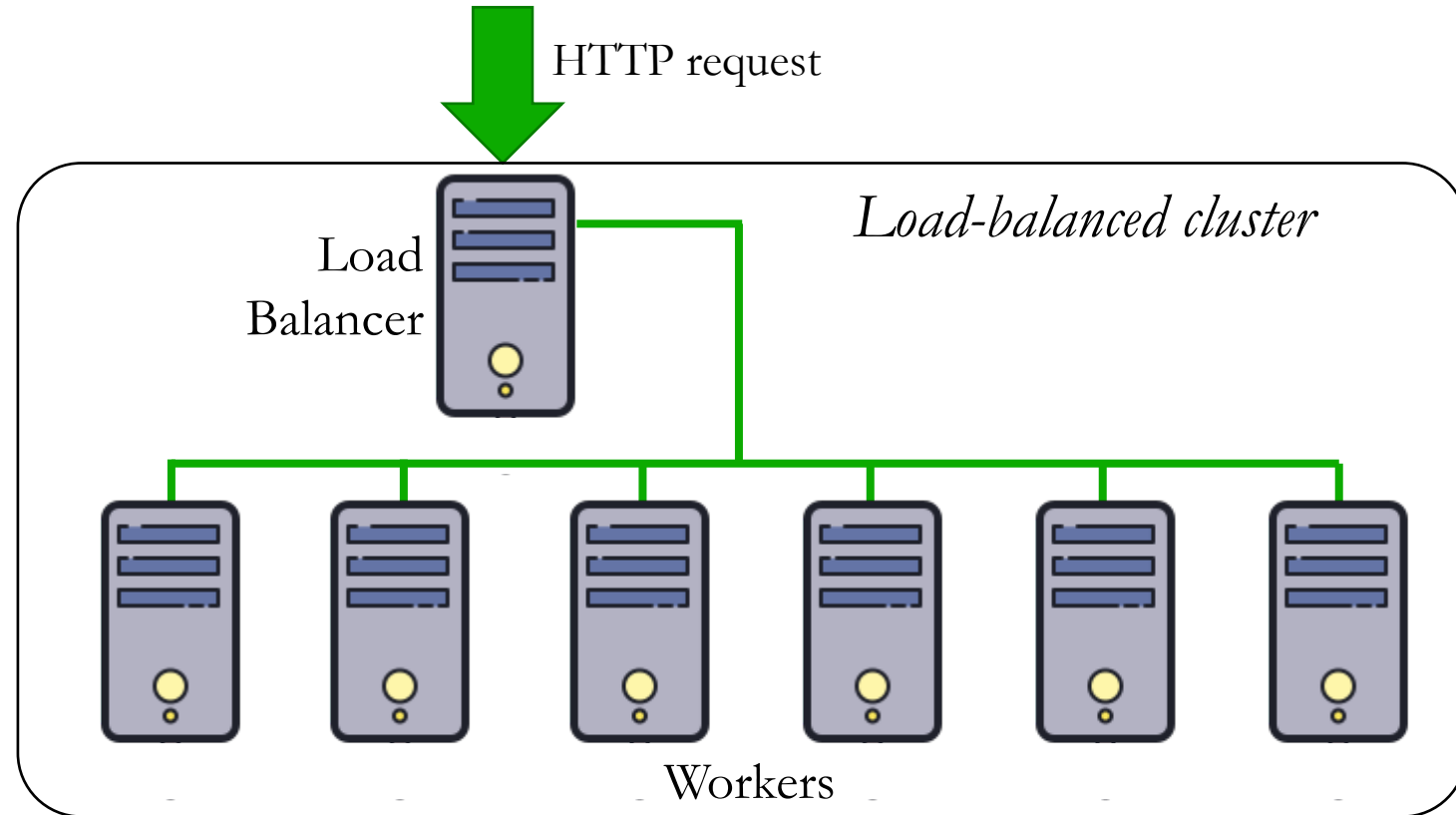
Basic idea of load balancer

- Make a cluster of servers look like one *superior* server.
- The load balancer provides the same interface as a single server.
(An HTTP server operating on a single IP address)

Client thinks it's dealing with this:



But it's just an illusion. The reality is:

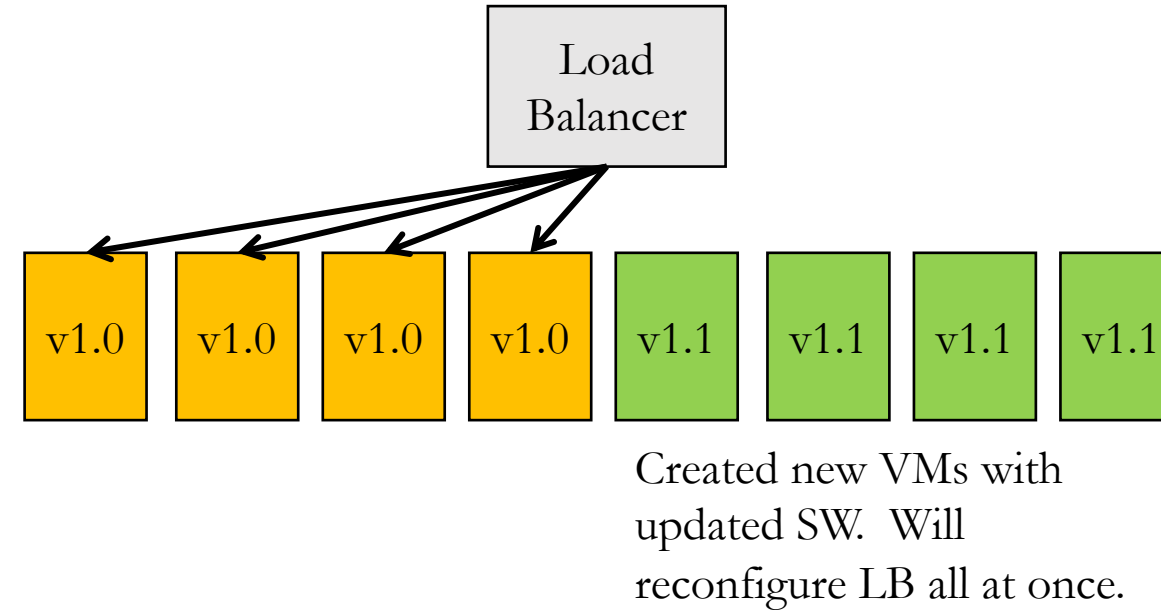
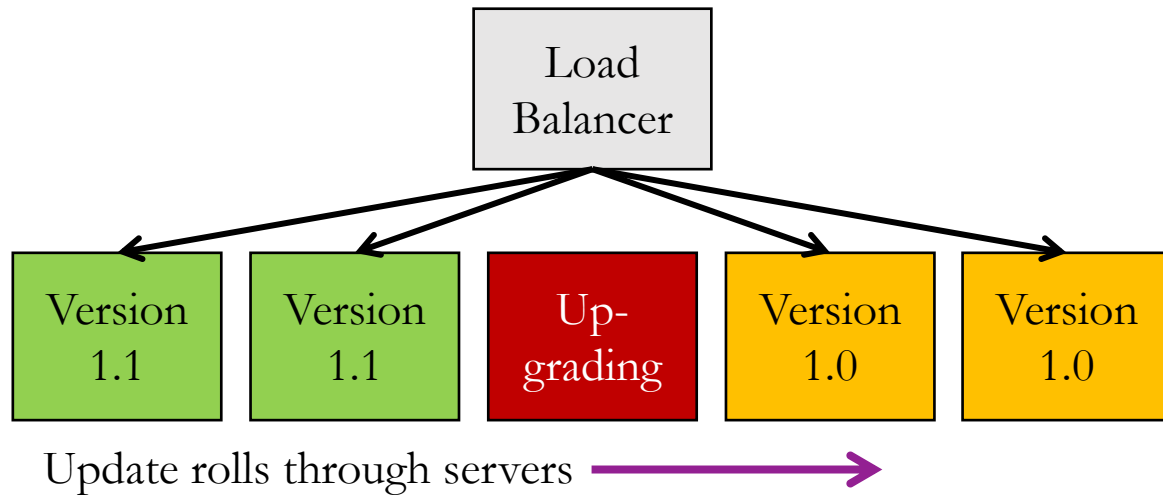


Additional benefits of a load balancer

Pros and cons of rolling vs. synchronized updates?



- Individual servers can be replaced without affecting overall service.
 - Deploy “rolling” app updates.
 - Or deploy a synchronized update



- Proxy can monitor **health** of servers
 - Periodically send a “health check” request. A simple GET API call.
 - If the request fails, then the server must be crashed.
 - Stop relaying new requests to that server.

Two types of *local* load balancers

Network Address Translation

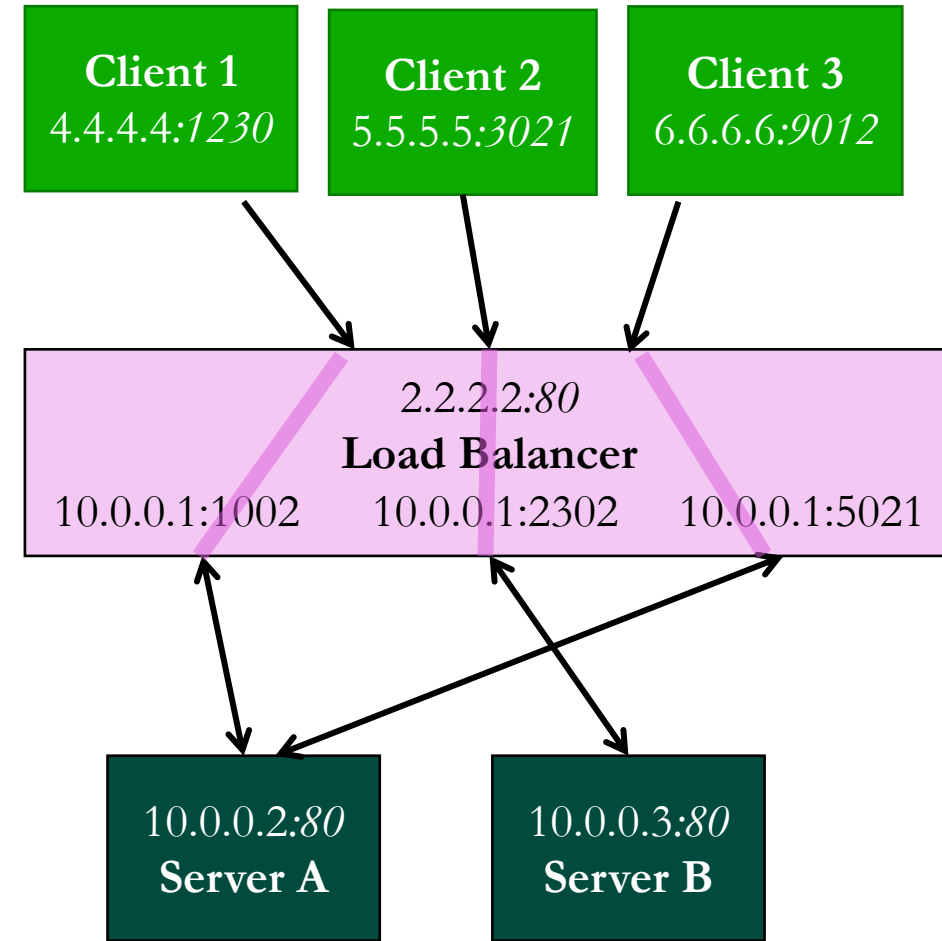
- Works at the **TCP/IP** layer.
 - Called a Layer-4 load balancer
- Forwards packets one-by-one, but remembers which server was assigned to each client.
- Is compatible with any type of service, not just HTTP.

Reverse Proxy

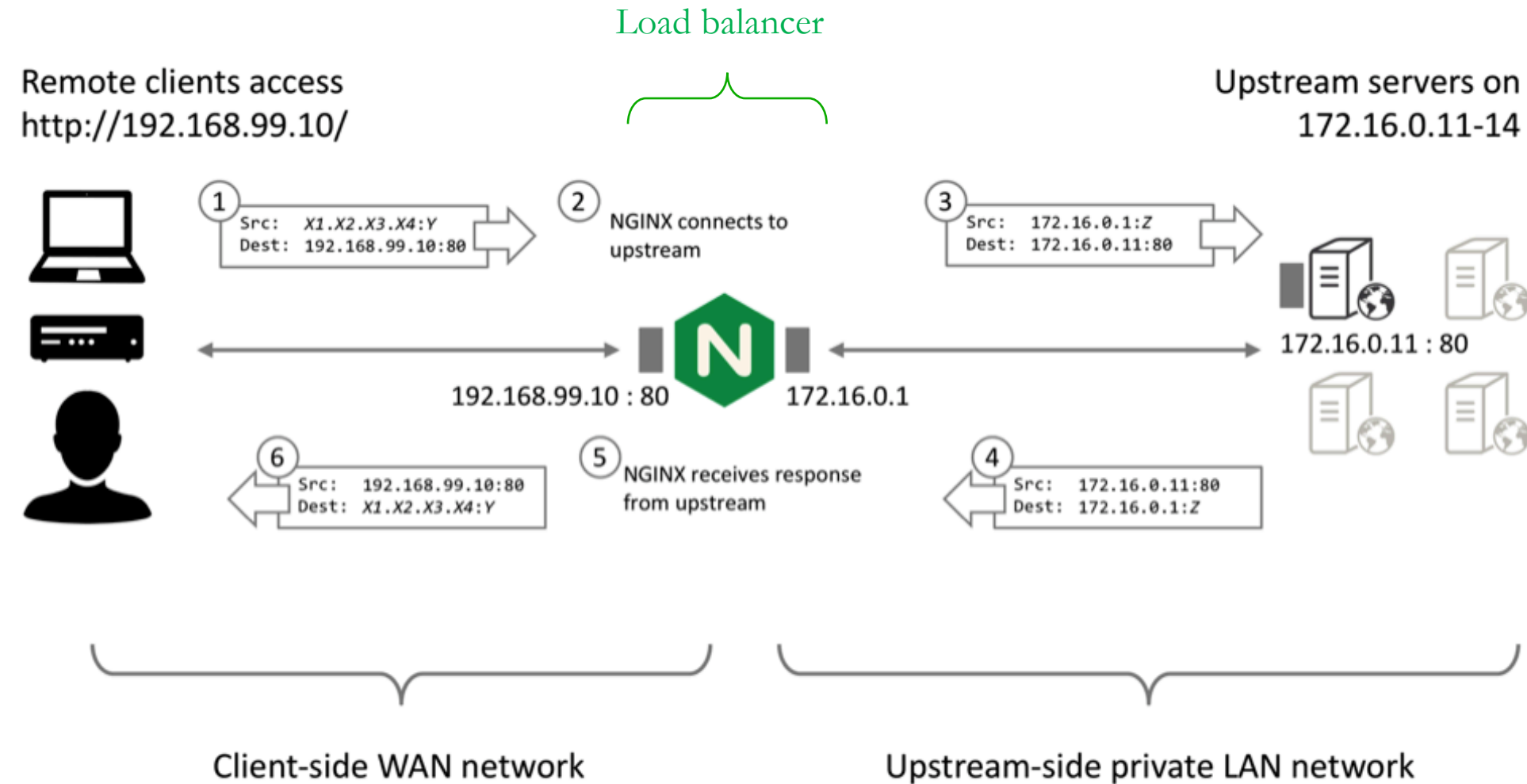
- Works at the **HTTP** layer.
 - Called an application-layer LB.
- Stores full requests/response before forwarding.
- Eg., Nginx (or maybe Squid)
- Reverse Proxy can also do:
 - SSL termination.
 - Caching.
 - Compression.

NAT Load Balancer

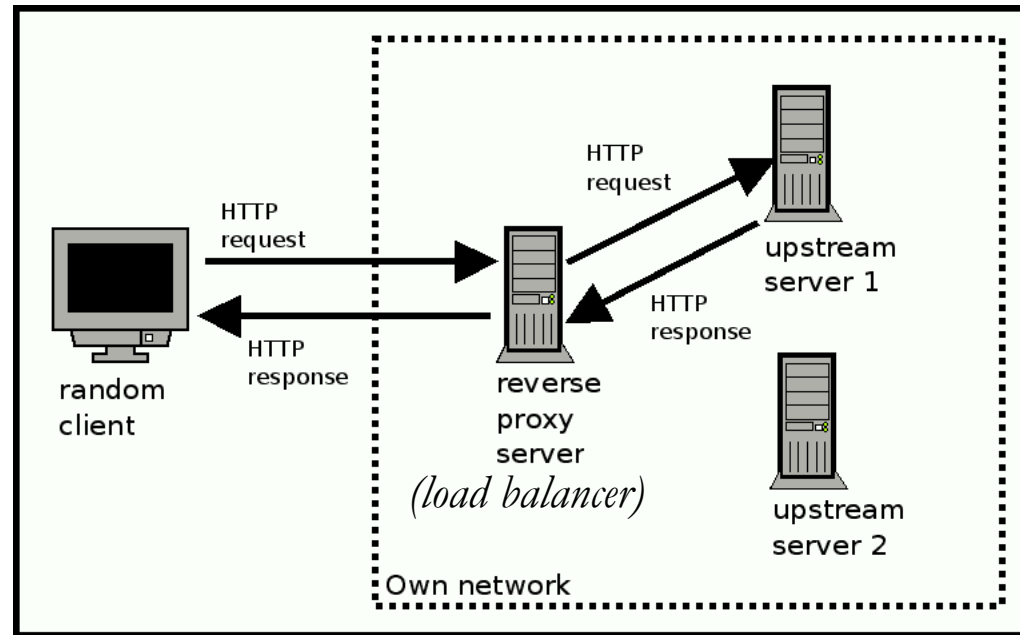
- For details, take CS-340.
- A type of NAT device that relays requests to multiple equivalent servers.
- NAT LB changes IP addresses and ports of packets in both directions.
- Load balancer maintains IP address and port mappings, like a traditional NAT.
- Simpler and more efficient than reverse proxy because it need not implement TCP, HTTP, or store full request/responses.



Nginx can be used as a **Reverse Proxy Load Balancer**



Additional benefits of a Reverse Proxy LB



- TLS/SSL certificates can be stored just on the proxy.
 - Internal communication may be unencrypted.
- Proxy might also cache responses, but this limits its scalability.
 - Eg., **Squid** in the Wikipedia architecture.

Comparison of **Local** load balancing options

	NAT	Reverse Proxy
Routing done by:	IP address/port translation	HTTP proxy
Scale	~1–10M requests/s	~100k–1M requests/s
Services supported	Any	Only HTTP
Additional features	None	SSL termination. Caching.

- Expensive "hardware" load balancers implement NAT.
- Reverse proxies are the cheap, open-source option.
- Cloud-based LBs can do either.



Local load balancer limitations

- Load balancer machine is a **single point of failure**.
- Can only handle \sim 1M requests/sec.
- Resides in one data center, thus:
 - It's not near all your customers.
 - The data center is also a single point of failure.
- Huge services need more than just local load balancers.
- Can clients find a service replica without contacting a central bottleneck?
- We have a distributed **service discovery** problem.

Domain Name Service (DNS)

- DNS is a distributed directory that maps hostnames to IP addresses.
 - [mail.stevetarzia.com](#) → 54.245.121.172
- DNS uses a distributed, hierarchical, caching architecture for scalability.
- On campus, my machine sends queries to NU's DNS server.
- This local DNS resolver has cached copies of recent (*common*) answers.
 - [gmail.com](#), [northwestern.edu](#), [facebook.com](#), etc...
- Local DNS resolver asks up the hierarchy if answer is not in its cache.
 - Need to ask the *nameserver* for “[stevetarzia.com](#)” for IP address of “[mail](#)”.
 - To get IP address of that nameserver, would ask the “com” nameserver.
 - IP address of “[com](#)” TLD nameserver is almost certainly cached, but if not then query one of the few hard-coded root nameservers.
- For more details, take CS-340 Intro. to Computer Networking.
- AWS outage on October 22nd, 2019 was due to a DDoS attack on AWS DNS.

Round-robin DNS for load balancing

- DNS allows multiple answers to be given for a query (multiple IP addresses per domain name).
 - Client can then randomly choose one of the IP addresses.
 - Even better, DNS server can store multiple answers, but give different responses to different users (either randomly, or cyclically).
- Remember that DNS is a cached, distributed system.
 - Northwestern's DNS resolver may have been told google.com = 172.217.9.78.
 - UChicago's DNS resolver may have been told google.com = 172.217.9.80.
 - Comcast Chicago's DNS resolver was told google.com = 172.217.7.83.
 - Different answers are cached and relayed to all the users on the 3 networks.
- Each of those three IP addresses are different reverse-proxying load balancers sitting in front of hundreds of app servers.
- Is there a limit to scaling by DNS?



There's no limit, at least on the frontend!

Geographic load balancing with DNS

- More than just balancing load, DNS can also connect user to the **closest** replica of a service.
- Clever DNS server examines **IP address of requester** and resolves to the server that it thinks is closest to the client (IP address *geolocation*).
- In other words, the IP address answers are not just different, but **customized** for the particular client.

👁👁 *(command line demo with nslookup)*

Content Delivery Network (CDN)

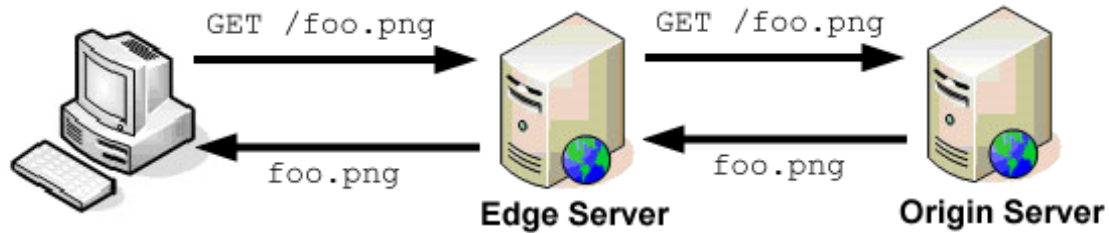
- Globally distributed web servers that *cache* responses for local clients.
- A CDN is just a distributed caching HTTP reverse proxy that uses DNS (*and other techniques*) to geographically load-balance.
- Eg., Akamai, Cloudflare, Cloudfront, Fastly

Distributed
“edge” servers

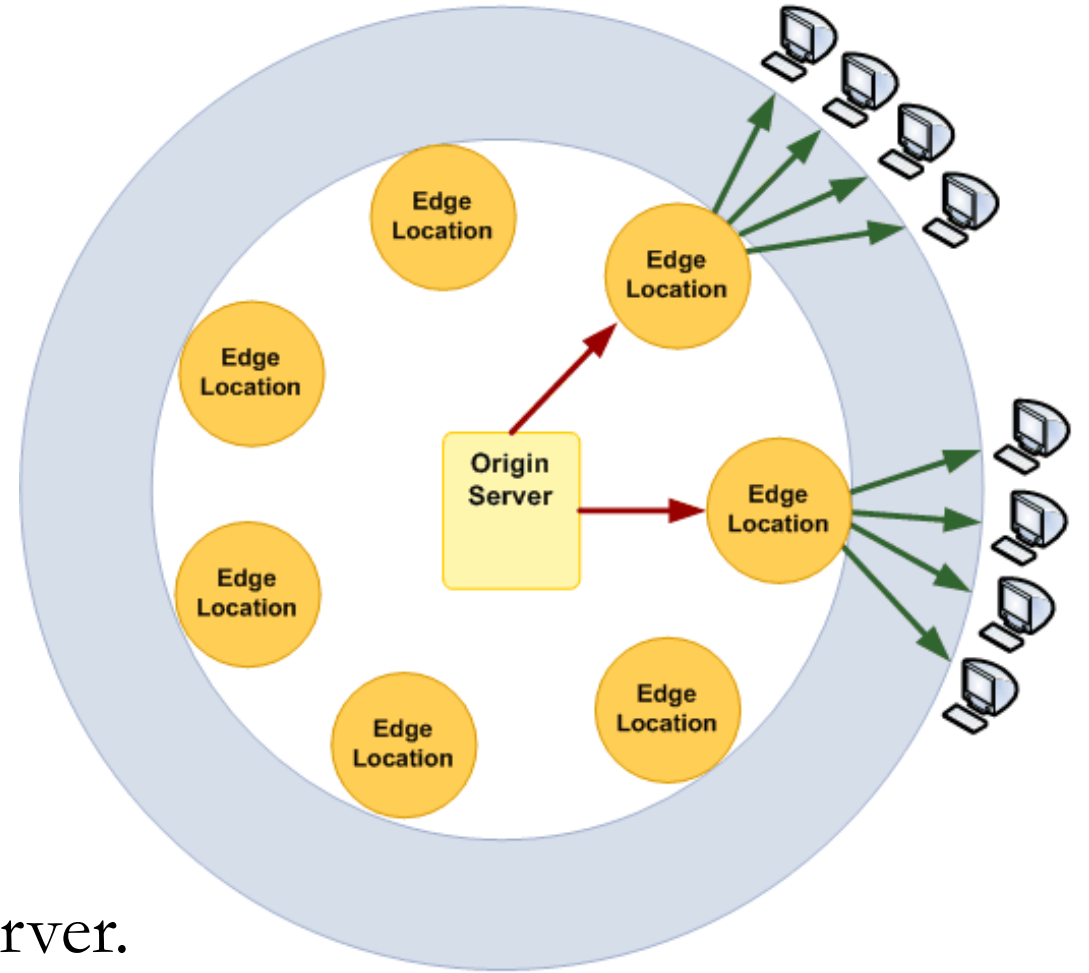
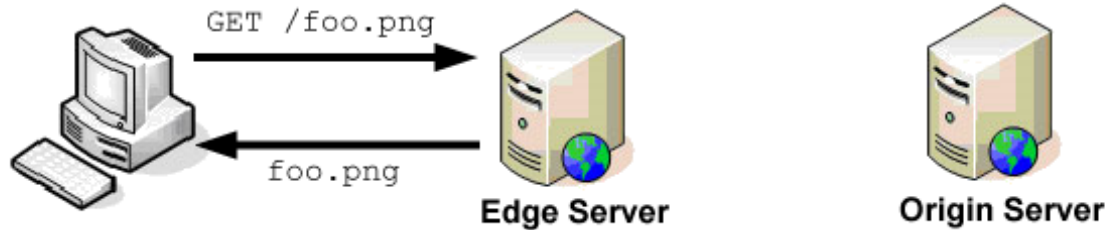


CDN uses HTTP caching proxies

First Request

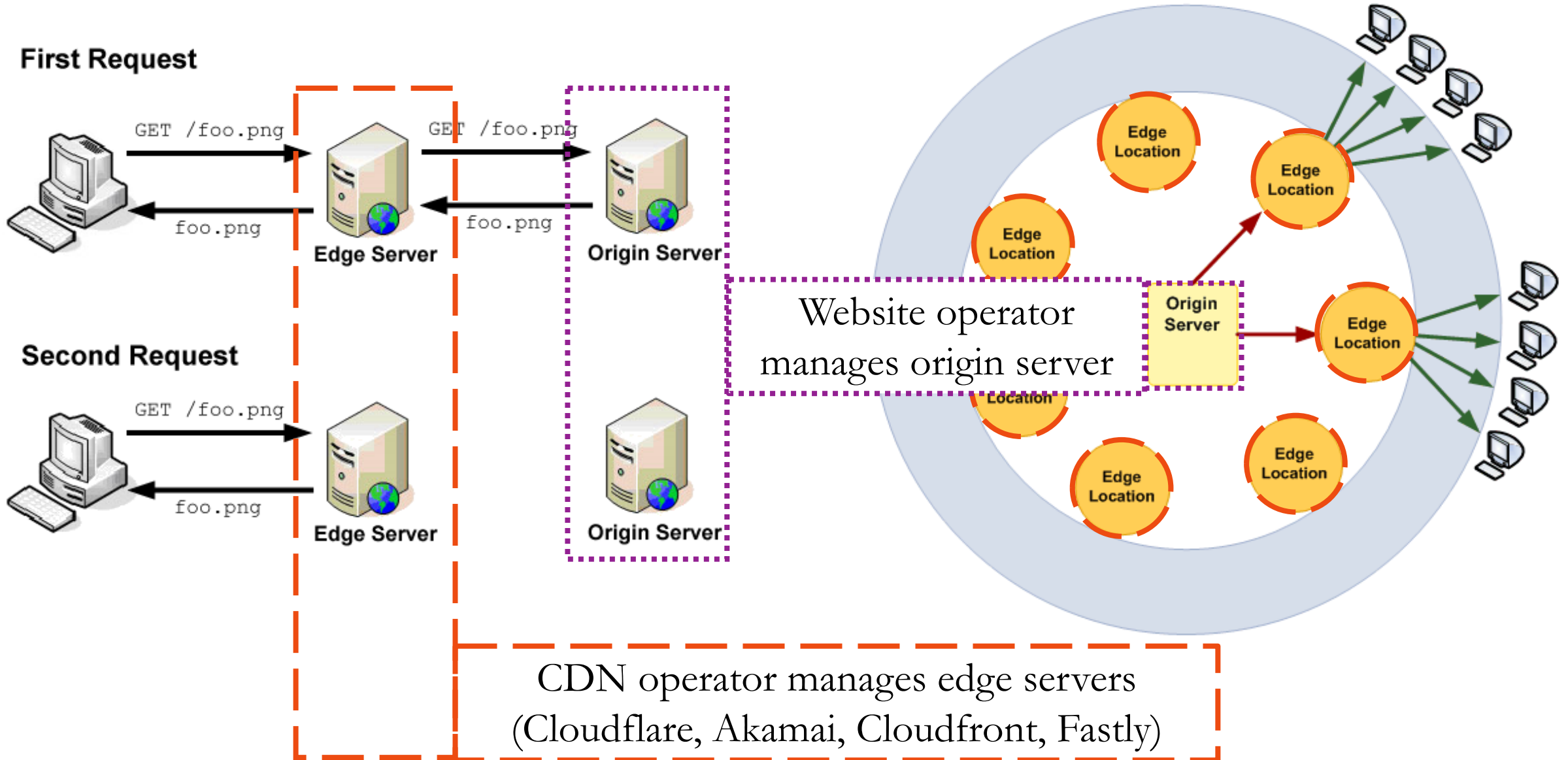


Second Request



- **Origin Server** is the original, central web server.
(Sets *cache-control* HTTP headers in responses).
- Edge Servers are **caching proxies**.
Ask origin server if don't have a cached response.

CDNs are *add-on* services




Cache-Control headers in HTTP responses

Command line demo:

- <https://www.google.com> does not allow caching:
 - cache-control: private, max-age=0
- https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png allow caching for **one year**
 - cache-control: private, max-age=31536000
- <https://www.northwestern.edu> also does not allow caching:
 - Cache-Control: max-age=0
- <https://common.northwestern.edu/v8/css/images/northwestern.svg> allows caching for one day:
 - Cache-Control: max-age=86400

Geographic load balancing with IP Anycast

- 8.8.8.8 is the **one IP address** for Google's huge public DNS service.
- Handles >[400 billion](#) DNS requests per day! *Anyone here use it?*  5 million QPS!
- Cannot rely on DNS for load balancing, because it **is** the DNS server!
- IP Anycast load balancing is implemented with BGP. (*Details in CS-340*)
- Basic idea is that many of Google's routers (around the world) all advertise to their neighbors that they can reach 8.8.8.8 in just one hop.
- Thus, traffic destined for 8.8.8.8 is sent to whichever of these Google routers are closest to the customer.
- Technically, this violates the principle that an IP address is a *particular* destination, but for DNS this doesn't matter because it's UDP and stateless.

Comparison of **Global** load balancing options

	DNS	IP anycast
Routing done by:	Domain Name Service	BGP
Maximum scale limited by:	Internet itself.	Internet itself.
How quickly can changes be made?	DNS TTL (minutes to hours)	BGP convergence (~minute)
Easy of deployment:	Requires advanced DNS software/config.	Must operate your own <i>Autonomous System (ISP)</i>

- For global load balancing, DNS is the most common choice.
- Tech giants (Google, Facebook, Amazon, MS, etc.) can use IP anycast.

Comparison of load balancing Levels

	Local	Global
Routing done by:	NAT or HTTP Proxy	DNS or BGP
Maximum scale limited by:	Speed of one machine.	Internet itself.
How quickly can changes be made?	Milliseconds	minutes to hours
Easy of deployment:	Simple, using off-the-shelf software/HW.	Requires advanced DNS software/config.

Most large services load balance at two levels:

- **Local** – provides continuous operation and scale within a data center
 - Includes *health checks & rolling updates*.
- **DNS** – for global scalability and low latency (*send users to nearby data center*).

Recap: Load balancers

We have 2/3 of the *end-to-end* view of a basic scalable architecture!

(for *services*, at least)

- *Frontend*: Client connects to “the service” via a **load balancer**.
 - Really, the client is being directed to one of many copies of the service.
 - Global LBs (DNS and IP anycast) have no central bottlenecks.
 - Local LBs (Reverse Proxy or NAT) provide mid-level scaling and continuous operation (*health checks & rolling updates*).
- *Services*: Implemented by thousands of clones.
 - If the code is **stateless** then any worker can equally handle any request.
- *Data Storage??*
 - The next big topic!

Today's topic