Managed Instance Groups and Load Balancing

Overview

Managed instance groups are a pool of similar machines which can be scaled automatically

Load balancing can be external or internal, global or regional

Basic components of HTTP(S) load balancing - target proxy, URL map, backend service and backends

Use cases and architecture diagrams for all the load balancing types HTTP(S), SSL proxy, TCP proxy, network and internal load balancing

Managed Instance Groups

Instance Groups

A group of machines which can be created and managed together to avoid individually controlling each instance in the project

2 Kinds of Instance Groups

Managed

Unmanaged

2 Kinds of Instance Groups

Managed

Unmanaged

Managed Instance Group

- Uses an **instance template** to create a group of **identical** instances
- Changes to the instance group changes all instances in the group

Instance Template

Defines the machine type, image, zone and other properties of an instance.

A way to save the instance configuration to use it later to create new instances or groups of instances

Instance Template

- **Global** resource not bound to a zone or a region
- Can **reference zonal resources** such as a persistent disk
    - In such cases can be used only within the zone

Managed Instance Group

- Can automatically scale the number of instances in the group
- Work with load balancing to distribute traffic across instances
- If an instance stops, crashes or is deleted the group automatically recreates the instance with the same template
- Can identify and recreate unhealthy instances in a group (autohealing)

2 Types of Managed Instance Groups

Managed

Zonal

Regional

Zonal vs. Regional MIG

- Prefer regional instance groups to zonal so application load can be spread across multiple zones
- This protects against failures within a single zone
- Choose zonal if you want lower latency and avoid cross-zone communication

Health Checks and Autohealing

- A MIG applies health checks to monitor the instances in the group
- If a service has failed on an instance, that instance is recreated (**autohealing**)
- Similar to health checks used in load balancing but the objective is different
  - LB health checks are used to determine where to send traffic
  - MIG health checks are used to recreate instances

Health Checks and Autohealing

- Typically configure health checks for both LB and MIGs
- The new instance is recreated based on the template that was used to originally create it (might be different from the default instance template)
- Disk data might be lost unless explicitly snapshotted

Configuring Health Checks

- **Check Interval:** The time to wait between attempts to check instance health
- **Timeout:** The length of time to wait for a response before declaring check attempt failed
- **Health Threshold:** How many consecutive “healthy” responses indicate that the VM is healthy
- **Unhealthy Threshold:** How many consecutive “failed” responses indicate VM is unhealthy

2 Kinds of Instance Groups

Managed

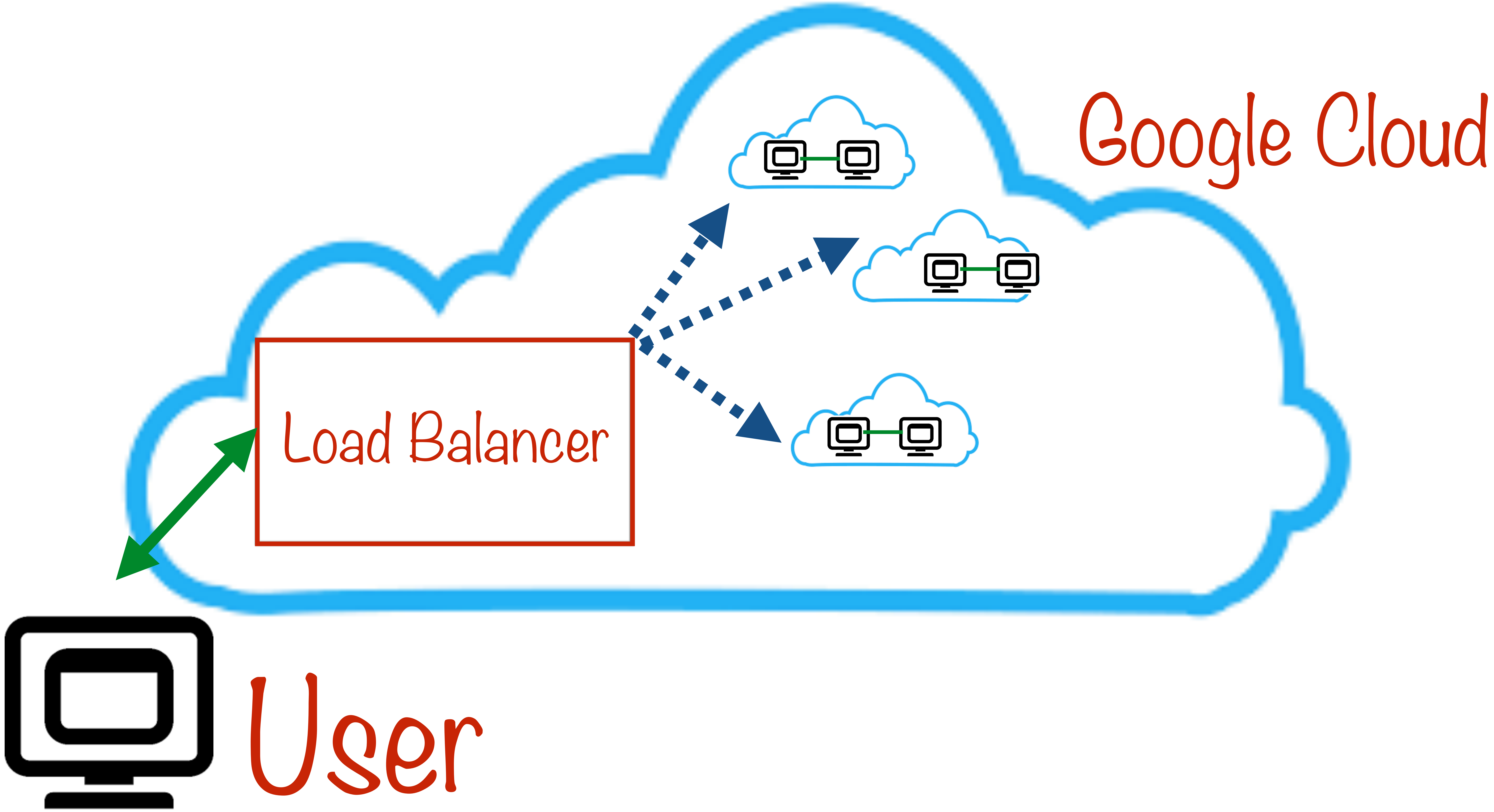
Unmanaged

Unmanaged Instance Groups

- Groups of dissimilar instances that you can add and remove from the group
- Do not offer autoscaling, rolling updates or instance templates
- Not recommended, used only when you need to apply **load balancing to pre-existing** configurations

Load Balancing

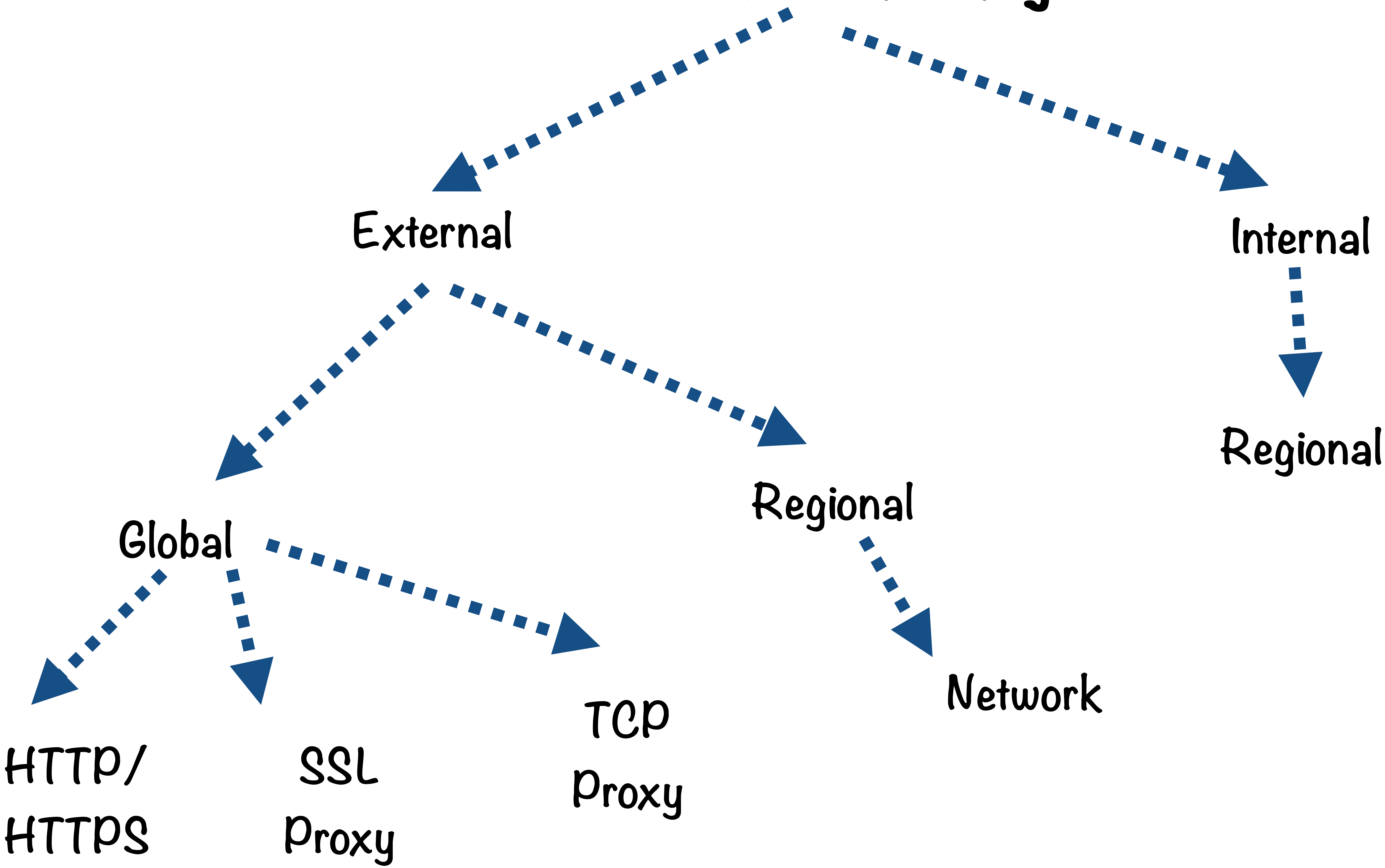
Load Balancing



Load Balancing

- Load balancing and autoscaling for groups of instances
- Scale your application to support heavy traffic
- Detect and remove unhealthy VMs, healthy VMs automatically re-added
- Route traffic to the closest VM
- Fully managed service, redundant and highly available

Load Balancing



External

Internal

Global

Regional

Regional

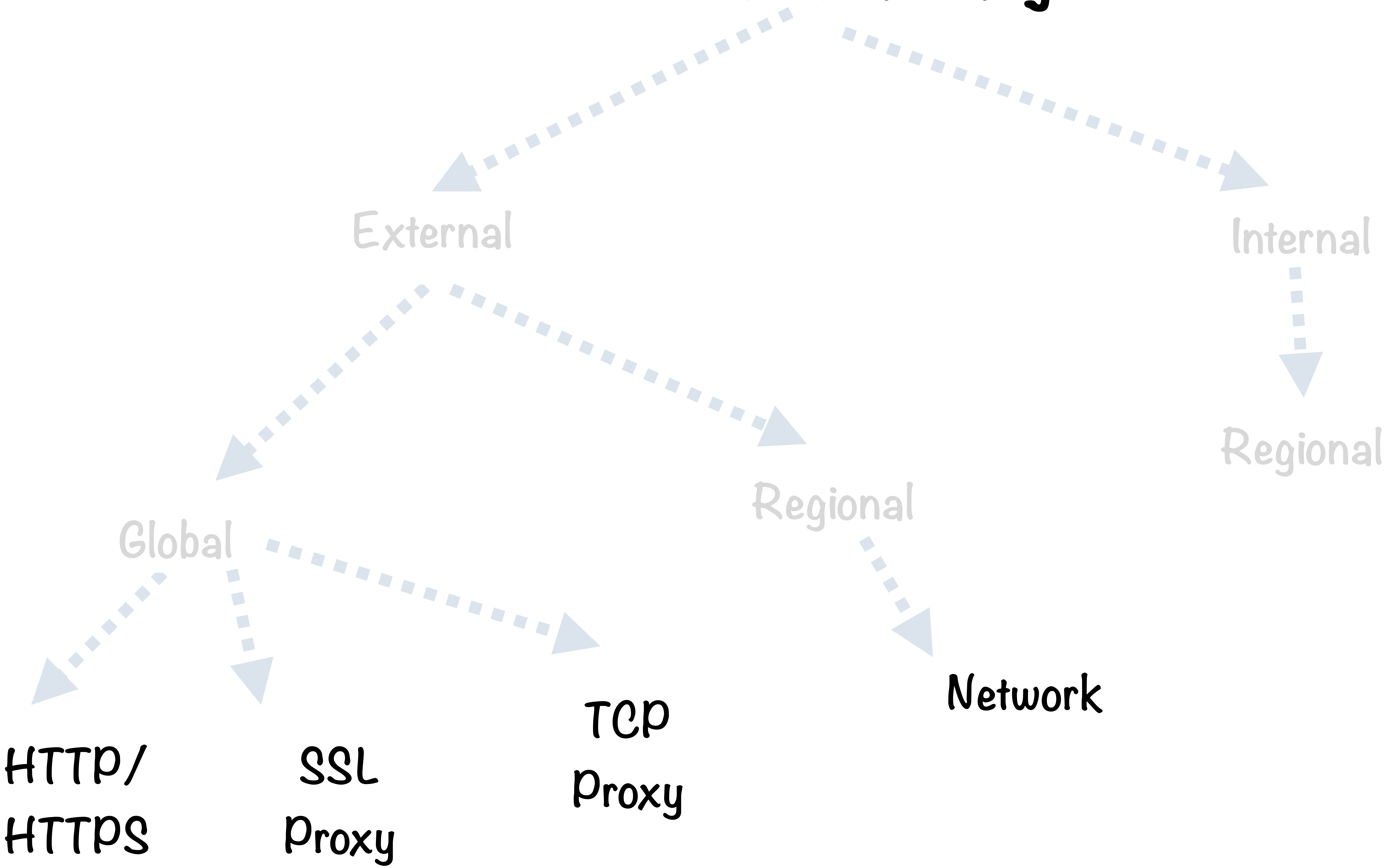
HTTP/ HTTPS

SSL Proxy

TCP Proxy

Network

Load Balancing



External

Internal

Global

Regional

Regional

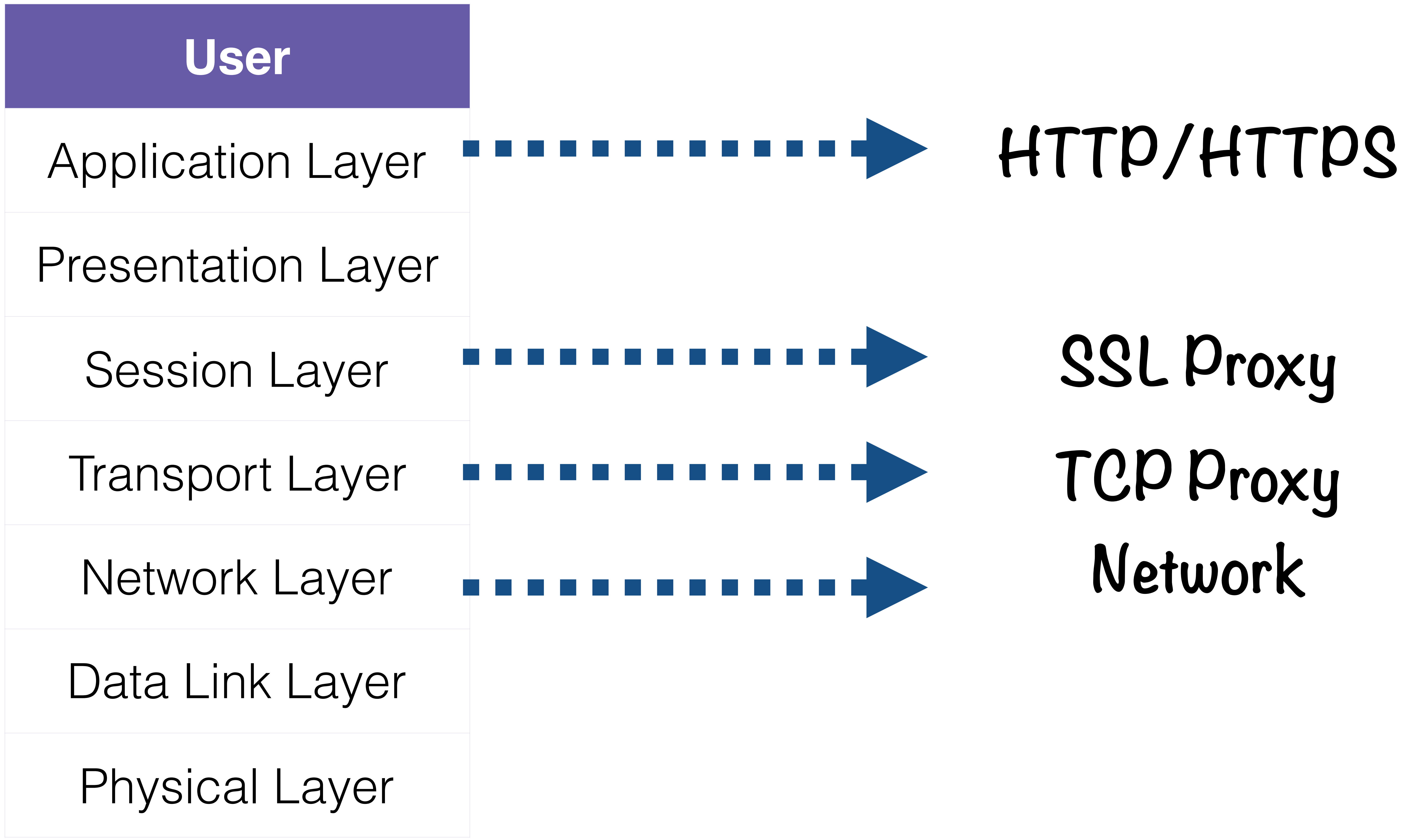
HTTP/
HTTPS

SSL
Proxy

TCP
Proxy

Network

OSI Network Stack



User

Application Layer → HTTP/HTTPS

Presentation Layer

Session Layer → SSL Proxy

Transport Layer → TCP Proxy

Network Layer → Network

Data Link Layer

Physical Layer

OSI Network Stack

User	
Application Layer	HTTP/HTTPS
Presentation Layer	
Session Layer	SSL Proxy
Transport Layer	TCP Proxy
Network Layer	Network
Data Link Layer	
Physical Layer	

Rule-of-thumb: Load balancer in the highest layer possible

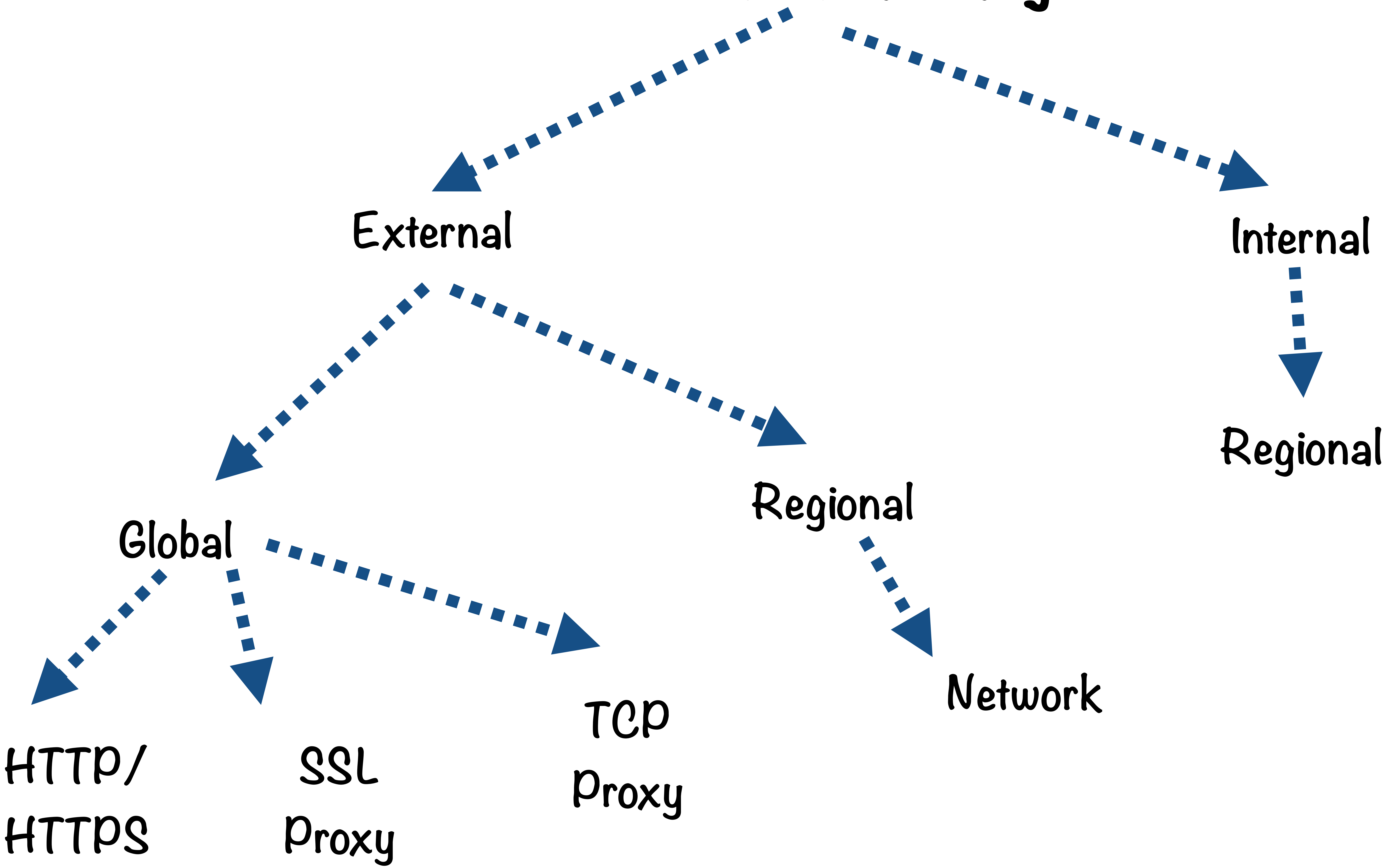
Health Checks

- HTTP, HTTPS health checks: Highest fidelity check because they verify that the web server is up and serving traffic, not just that the instance is healthy.

- SSL health checks: Configure the SSL health checks if your traffic is not HTTPS but is encrypted via SSL(TLS)

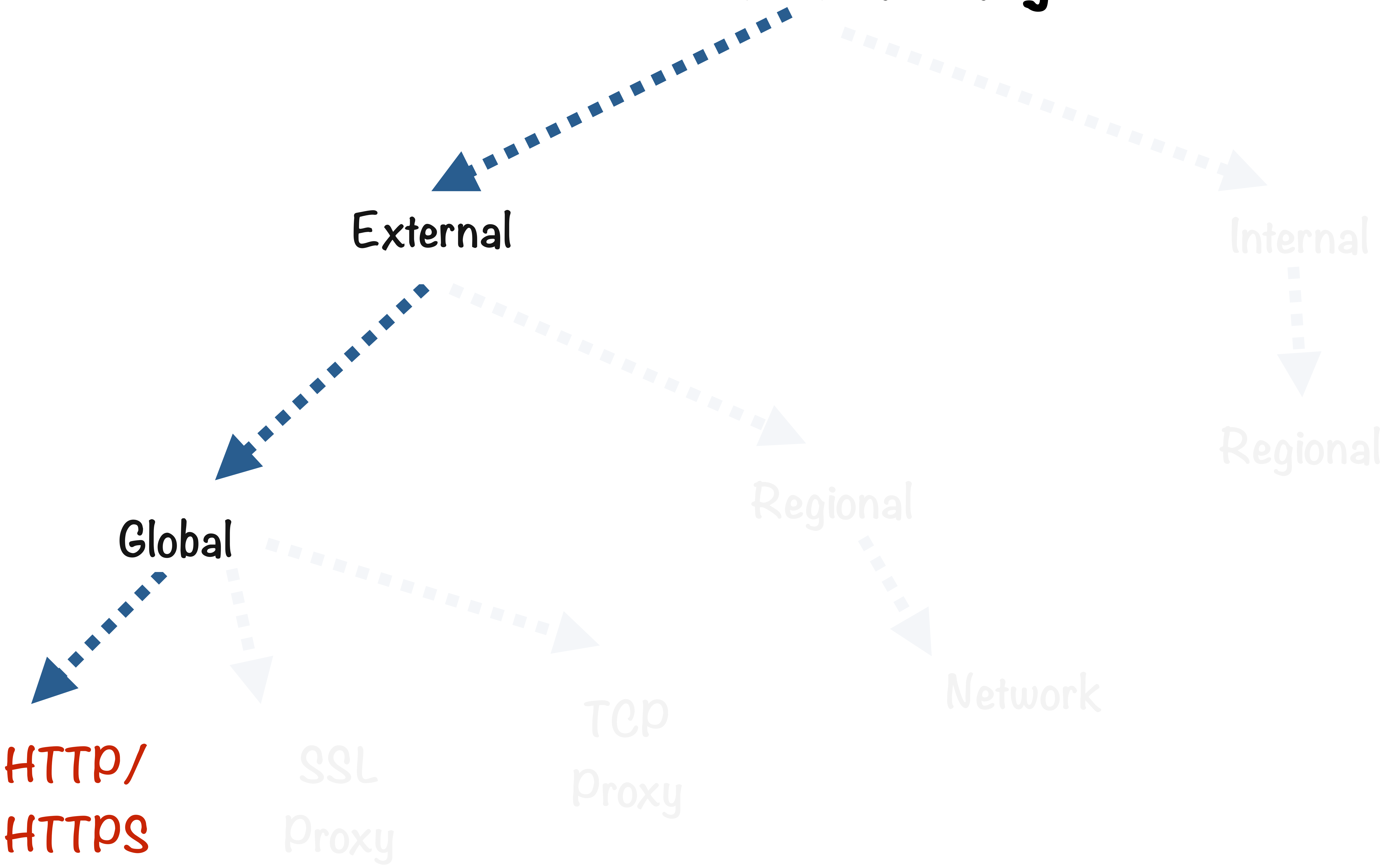
- TCP health checks: For all TCP traffic that is not HTTP(S) or SSL(TLS), you can configure a TCP health check

Load Balancing



- External
  - Global
    - HTTP/ HTTPS
    - SSL Proxy
    - TCP Proxy
  - Regional
    - Network
- Internal
  - Regional

Load Balancing



External

Internal

Global

Regional

Regional

HTTP/
HTTPS

SSL
Proxy

TCP
Proxy

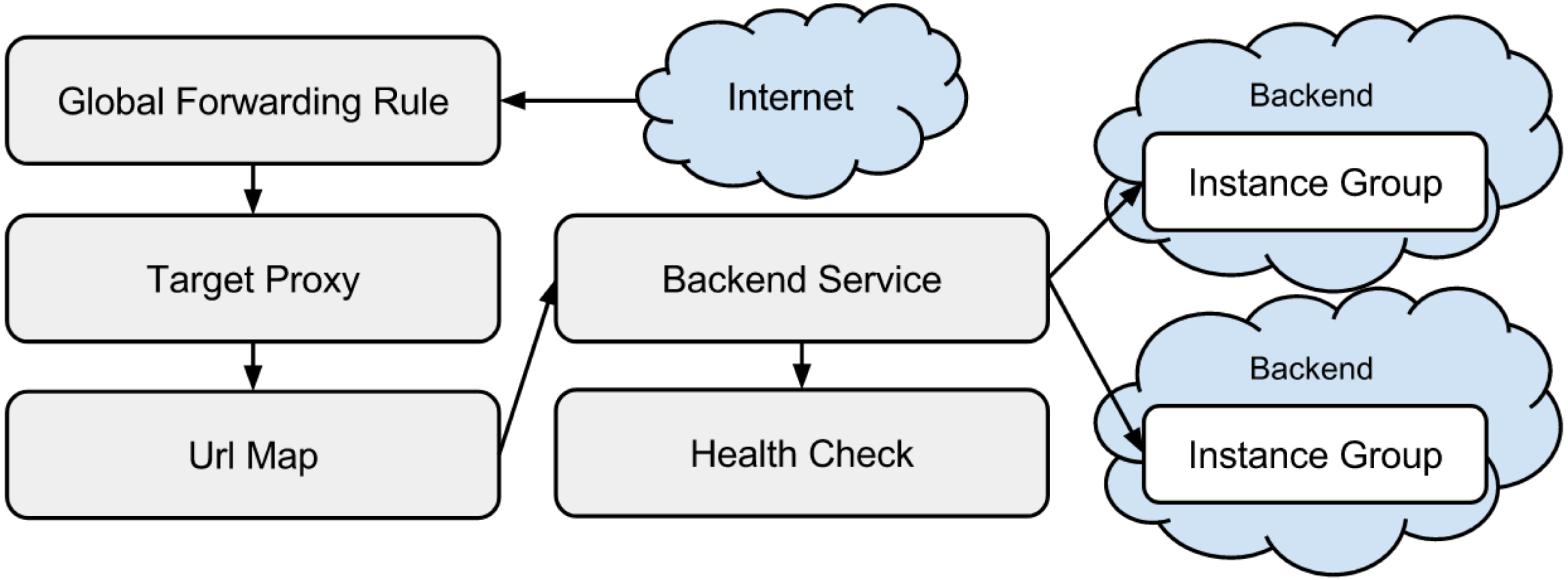
Network

OSI Network Stack

User	
Application Layer	HTTP/HTTPS
Presentation Layer	
Session Layer	SSL Proxy
Transport Layer	TCP Proxy
Network Layer	Network
Data Link Layer	
Physical Layer	

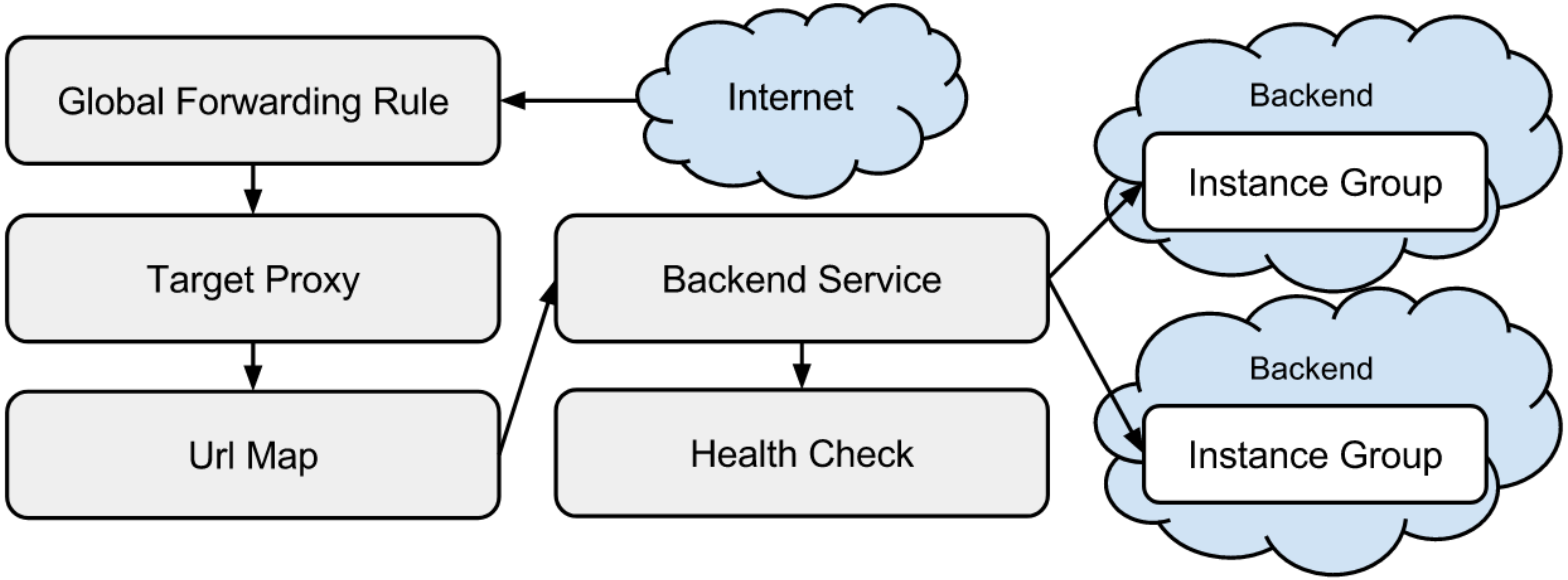
HTTP(S) load balancing is the "smartest"

HTTP/HTTPS Load Balancing



A global, external load balancing service offered on the GCP

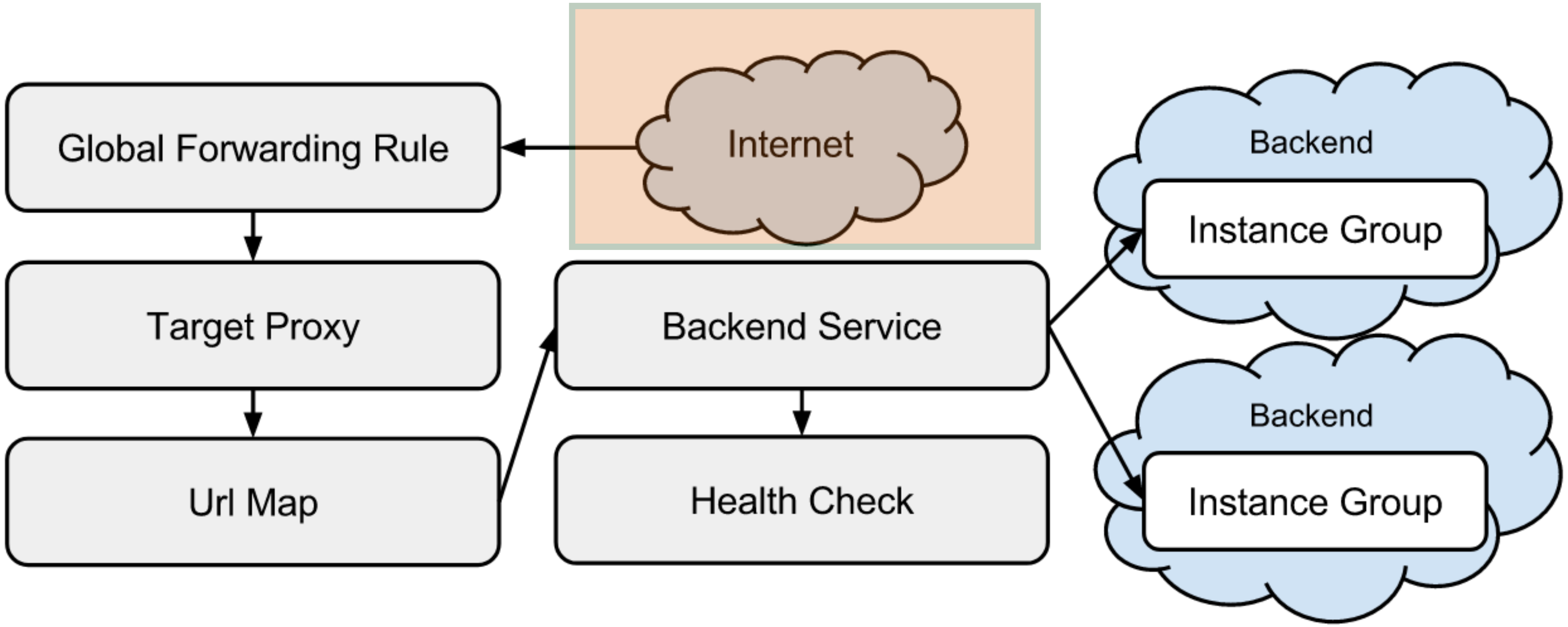
HTTP/HTTPS Load Balancing



Distributes HTTP(S) traffic among groups of instances based on:

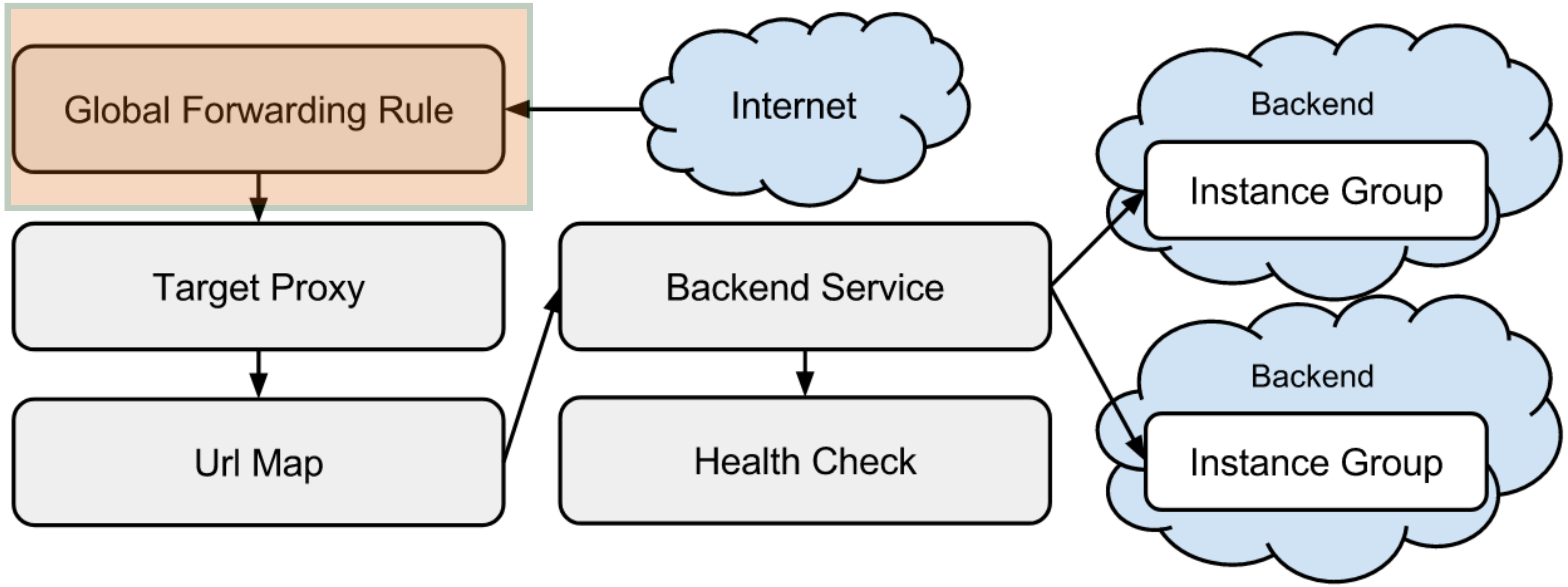
- proximity to the user
- requested URL
- or both.

HTTP/HTTPS Load Balancing



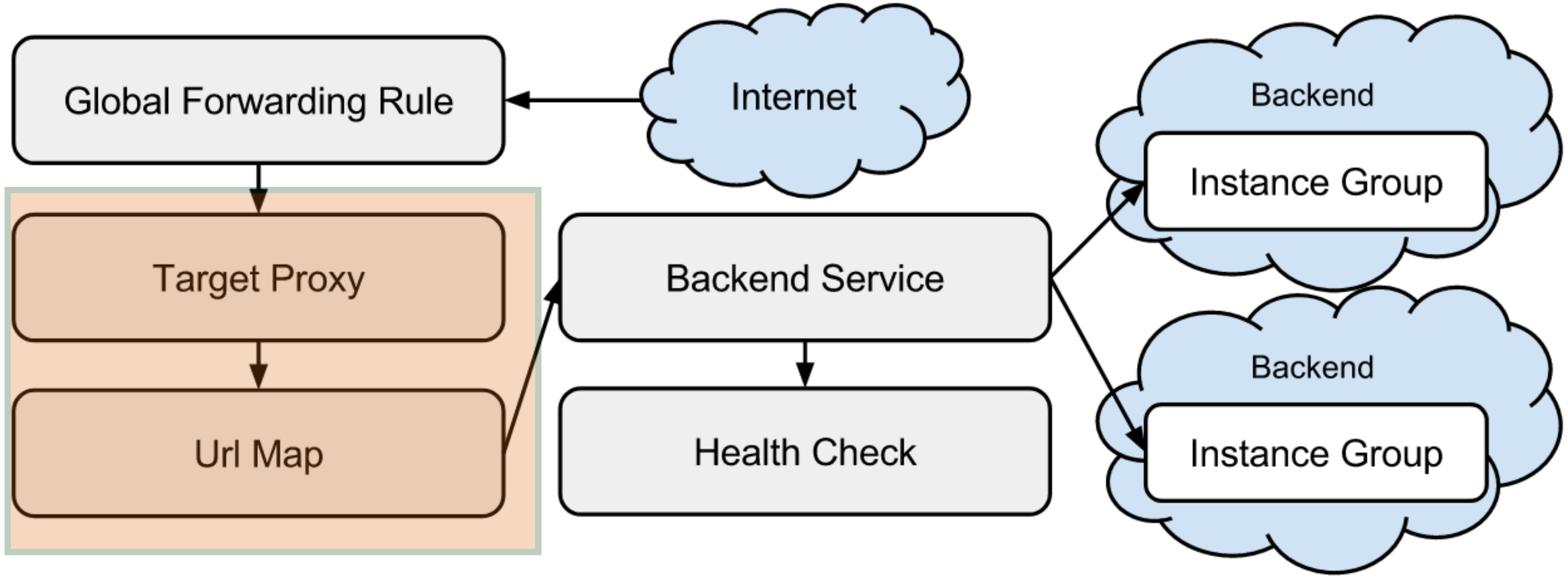
Traffic from the internet is sent to a global forwarding rule - this rule determines which proxy the traffic should be directed to

HTTP/HTTPS Load Balancing



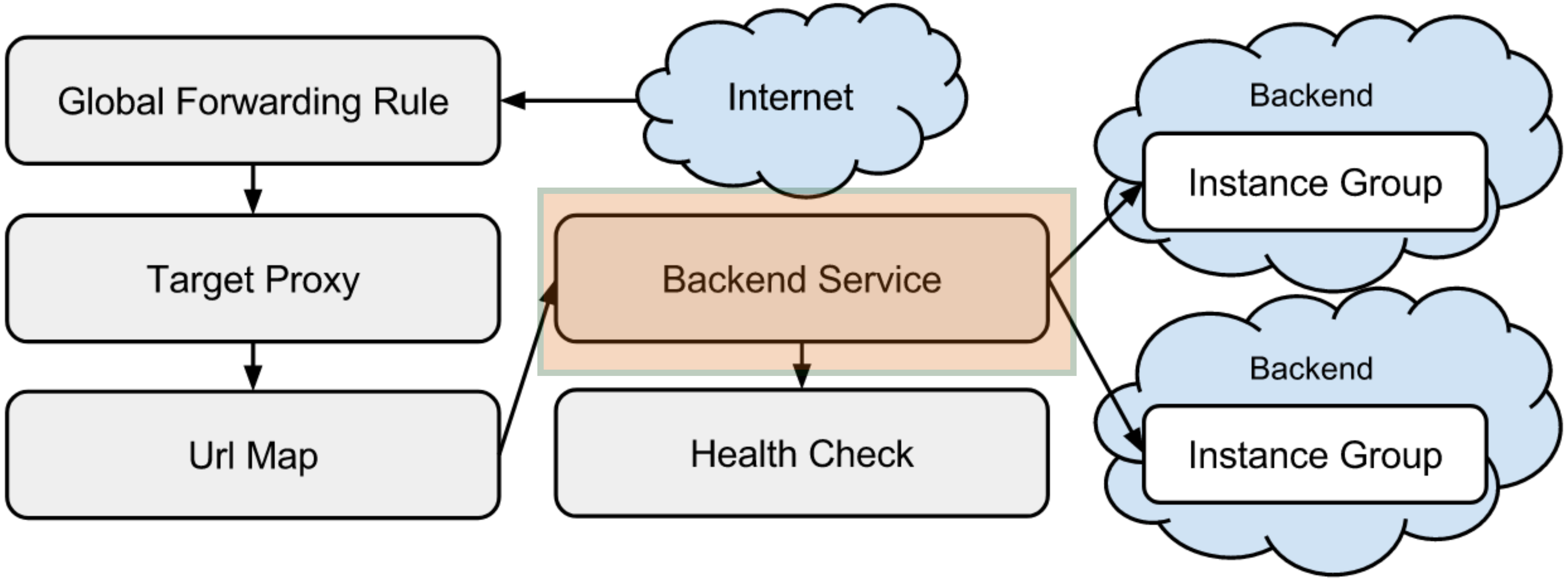
The global forwarding rule directs incoming requests to a target HTTP proxy

HTTP/HTTPS Load Balancing



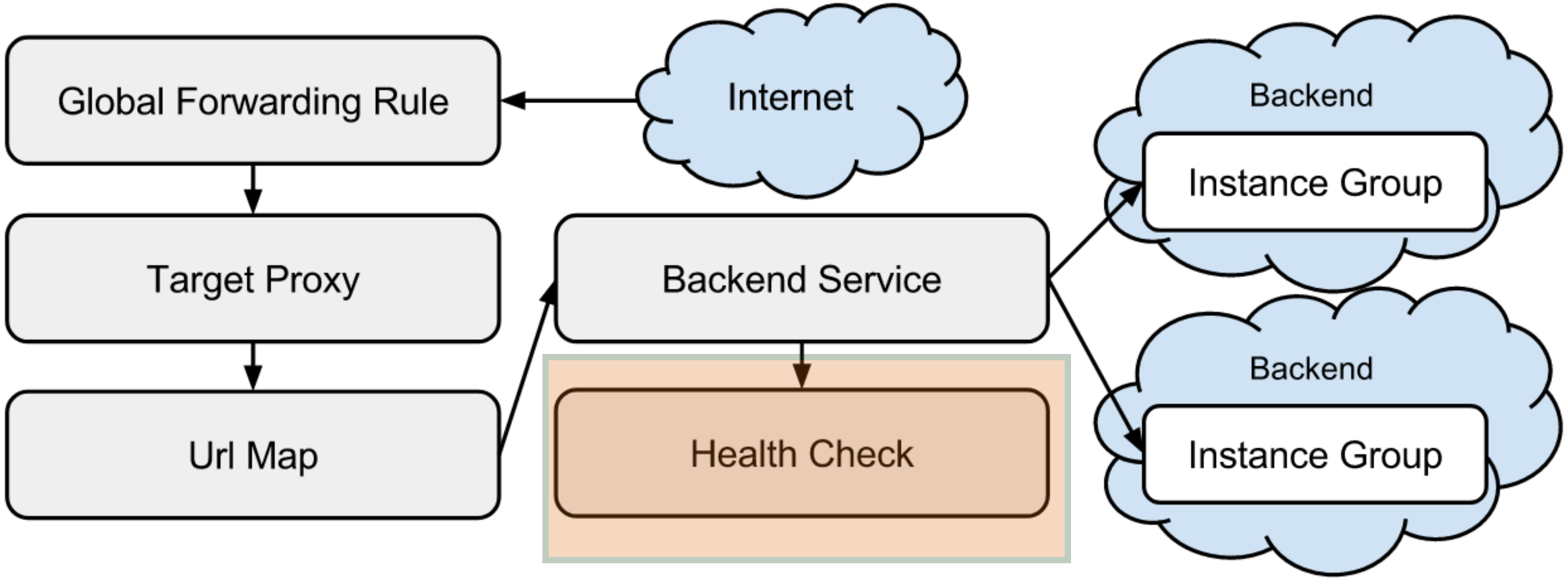
The target HTTP proxy checks each request against a URL map to determine the appropriate backend service for the request

HTTP/HTTPS Load Balancing



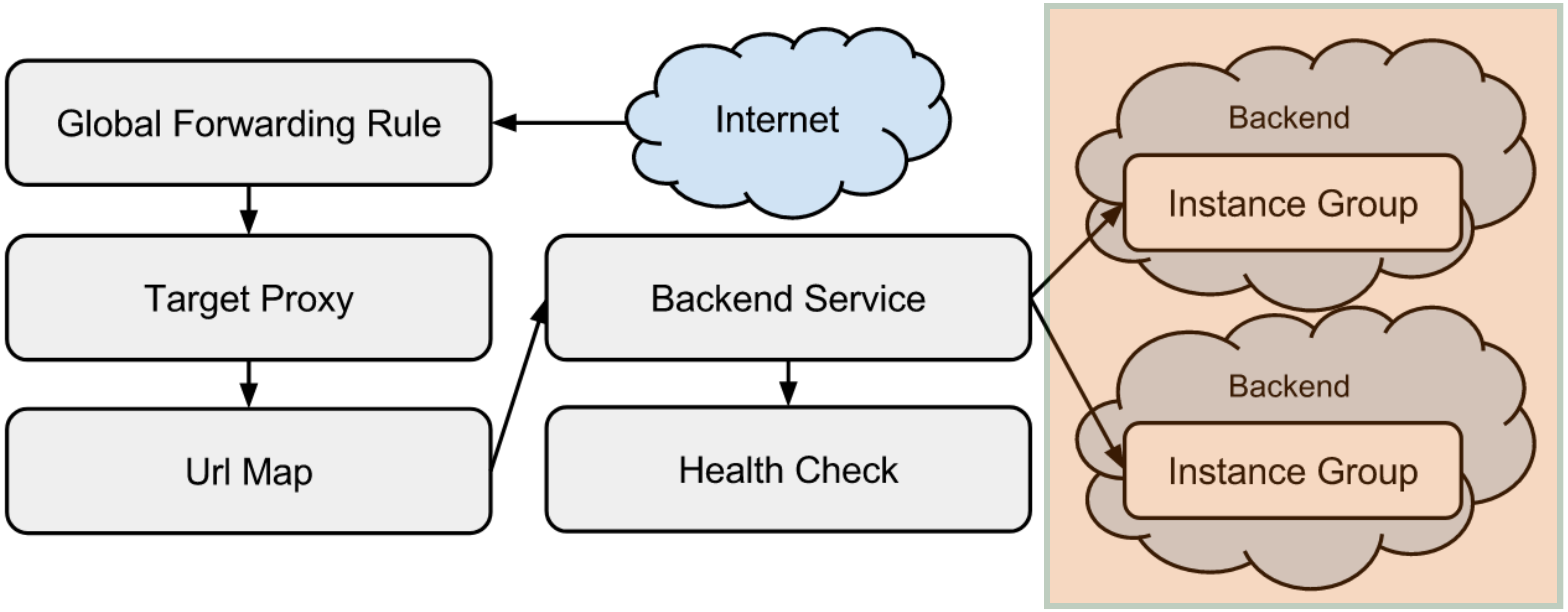
The backend service directs each request to an appropriate backend based on serving capacity, zone, and instance health of its attached backends

HTTP/HTTPS Load Balancing



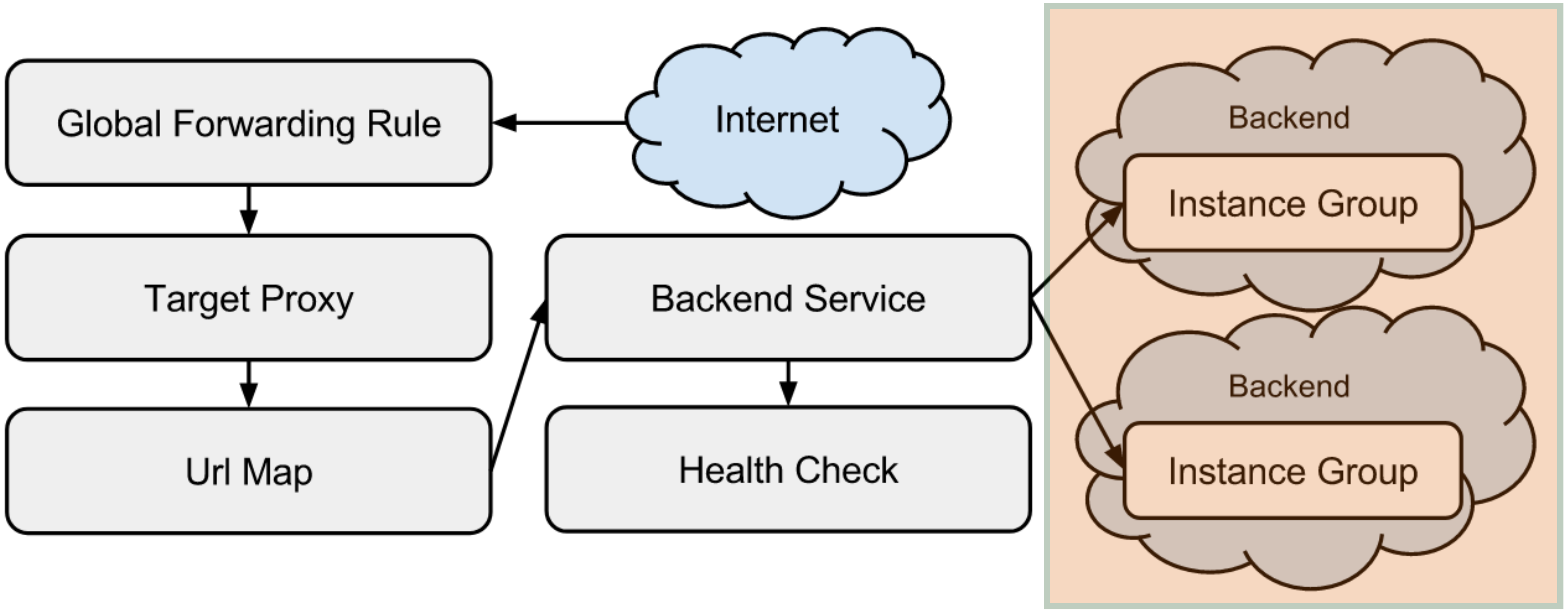
The health of each backend instance is verified using either an HTTP health check or an HTTPS health check - if HTTPS, request is encrypted

HTTP/HTTPS Load Balancing



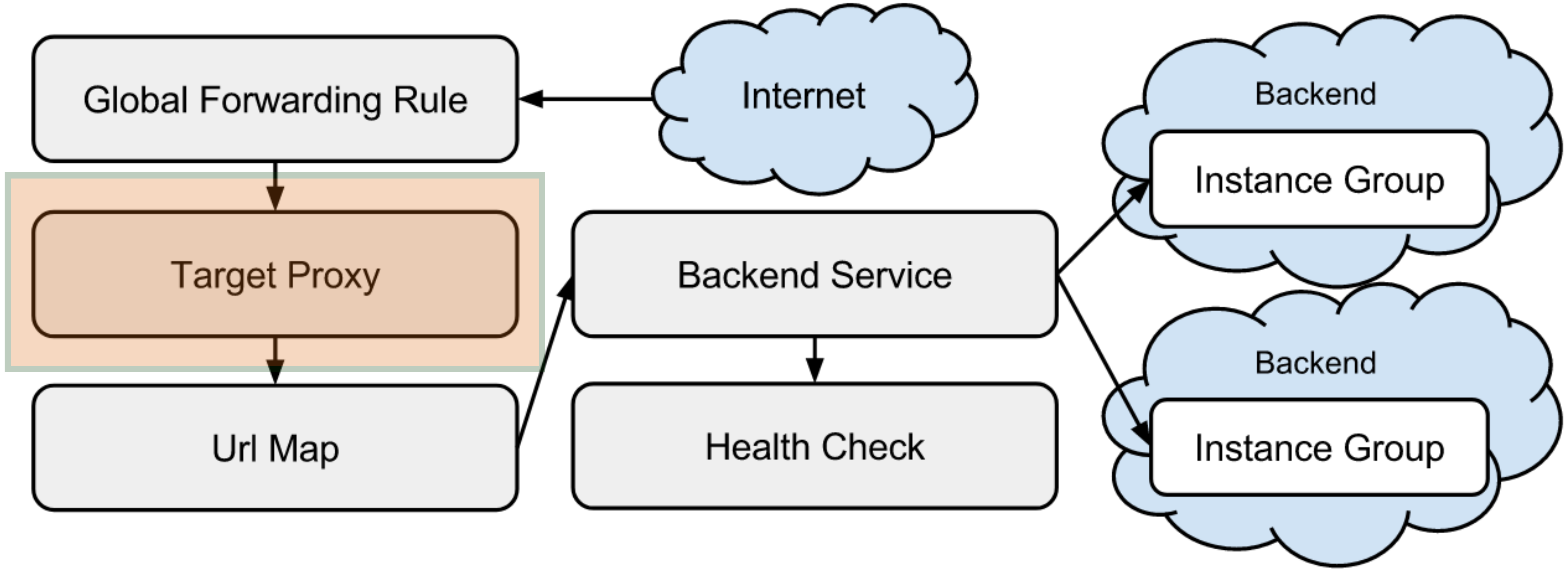
Actual request distribution can happen based on CPU utilization, requests per instance

HTTP/HTTPS Load Balancing



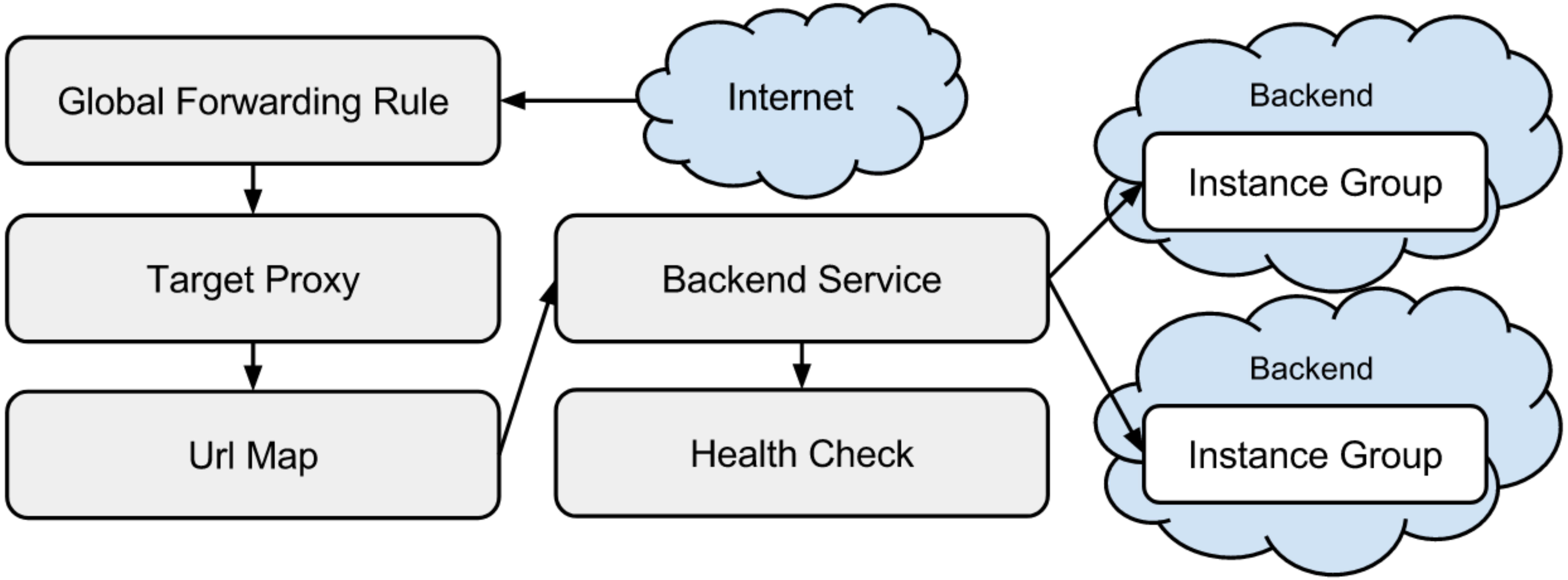
Can configure the managed instance groups making up the backend to scale as the traffic scales (based on the parameters of utilization or requests per second)

HTTP/HTTPS Load Balancing



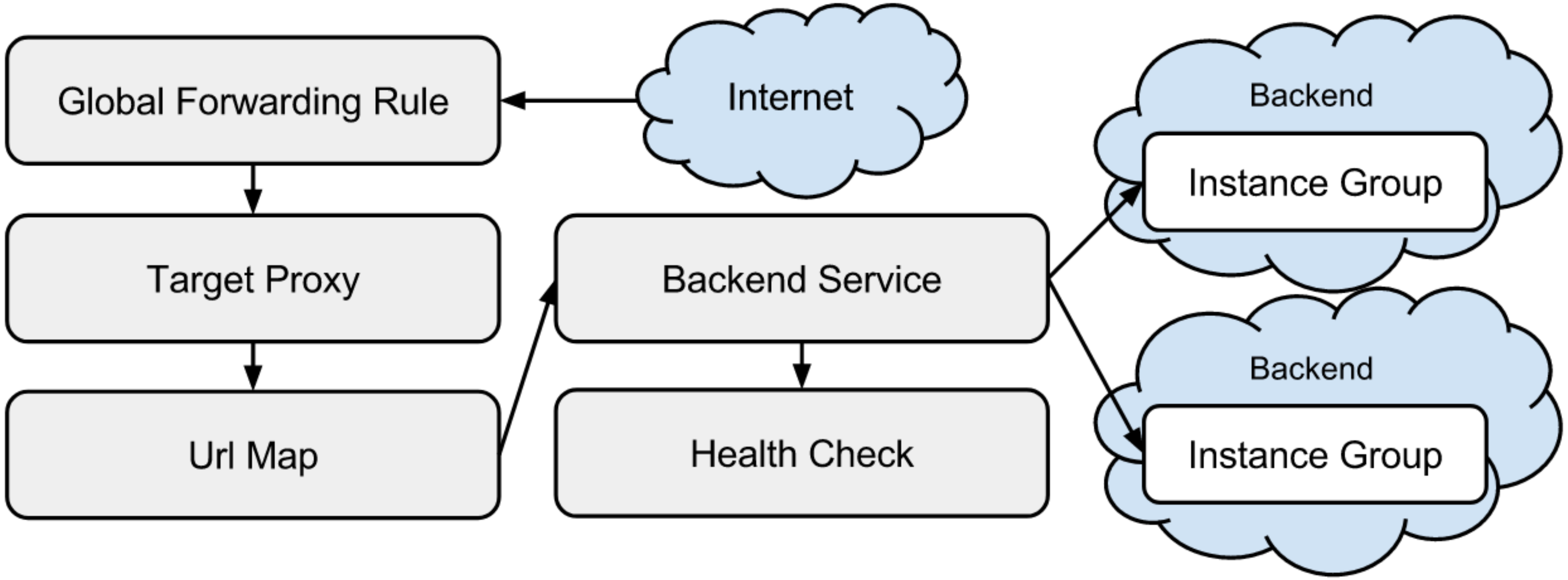
HTTPS load balancing requires the target proxy to have a signed certificate to terminate the SSL connection

HTTP/HTTPS Load Balancing



BTW, must create firewall rules to allow requests from load balancer and health checker to get through to the instances

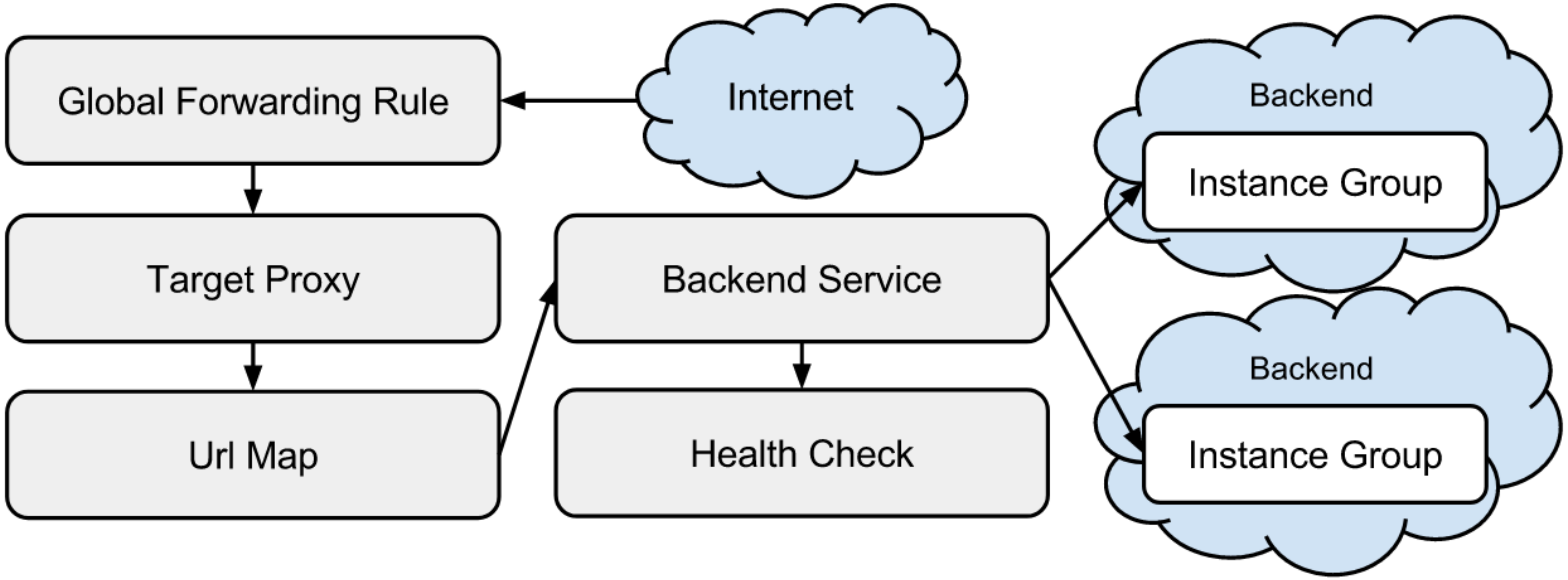
HTTP/HTTPS Load Balancing



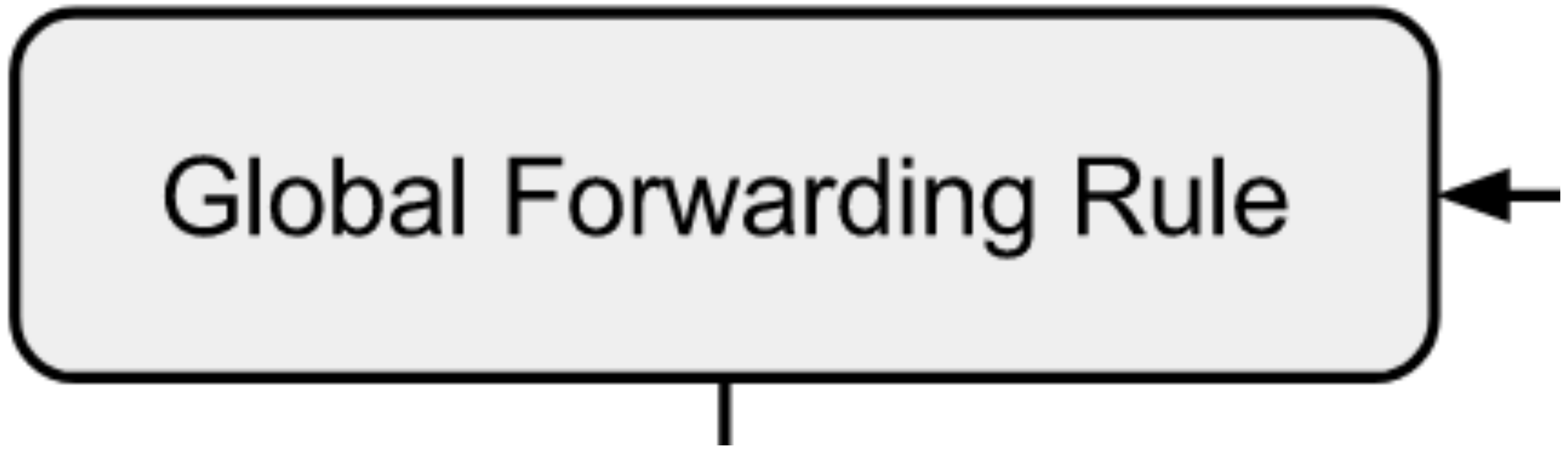
Session affinity: All requests from same client to same server based on either

- client IP
- cookie

HTTP/HTTPS Load Balancing



Global Forwarding Rules

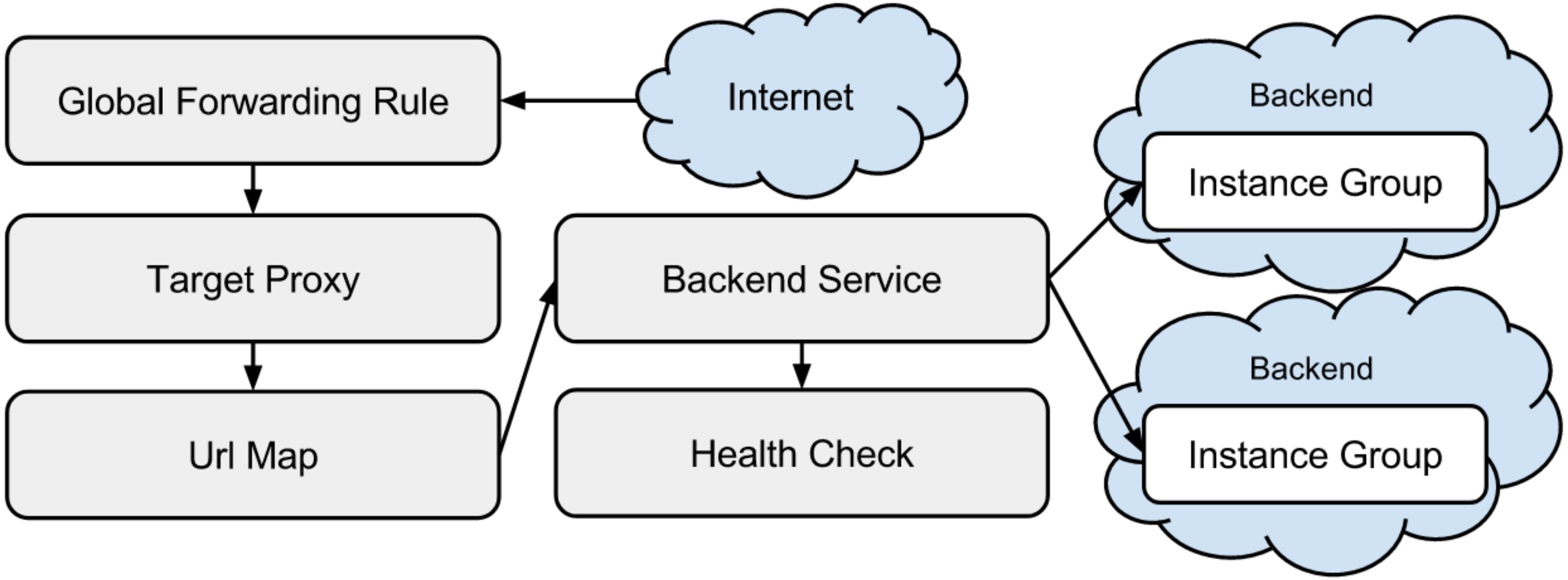


Global Forwarding Rule

Global Forwarding Rules

- Route traffic by IP address, port and protocol to a load balancing proxy
- Can only be used with **global** load balancing HTTP(S), SSL Proxy and TCP Proxy
- **Regional** forwarding rules can be used with regional load balancing and individual instances

HTTP/HTTPS Load Balancing



Global Forwarding Rule

Target Proxy

Url Map

Internet

Backend Service

Health Check

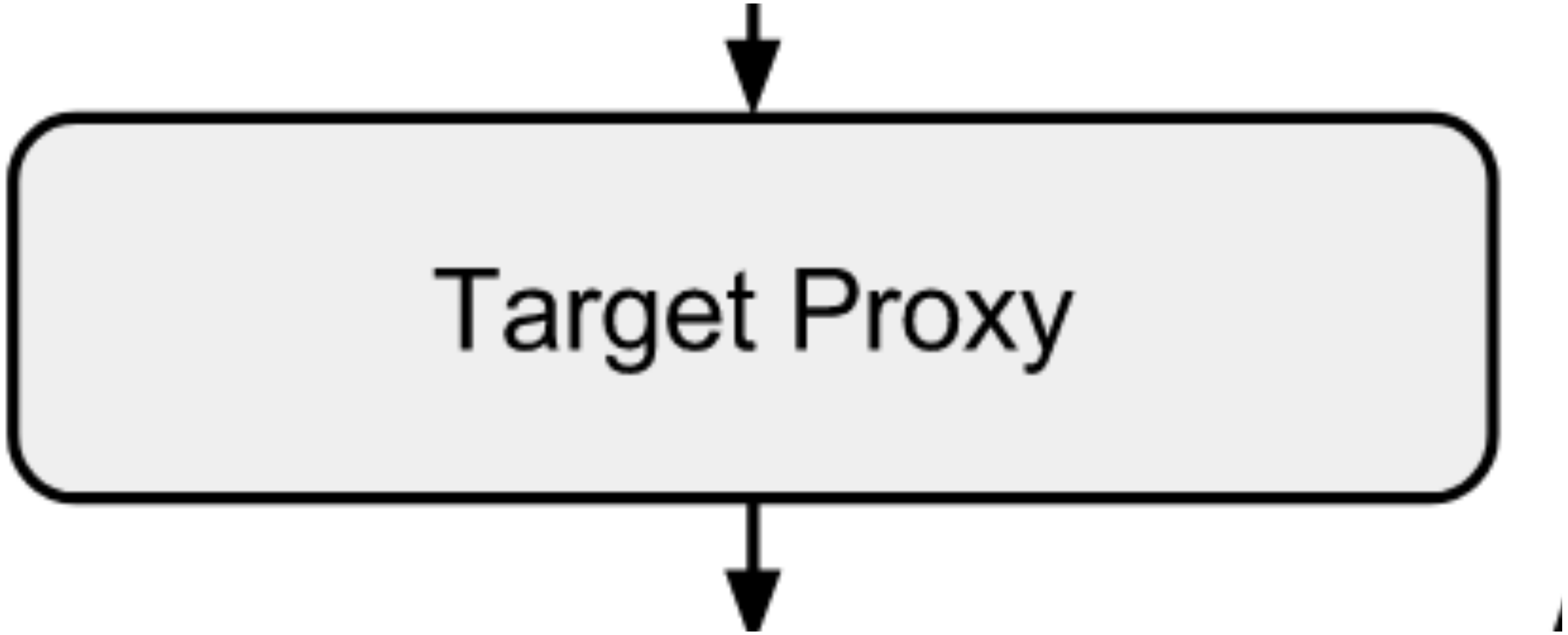
Backend

Instance Group

Backend

Instance Group

Target Proxy

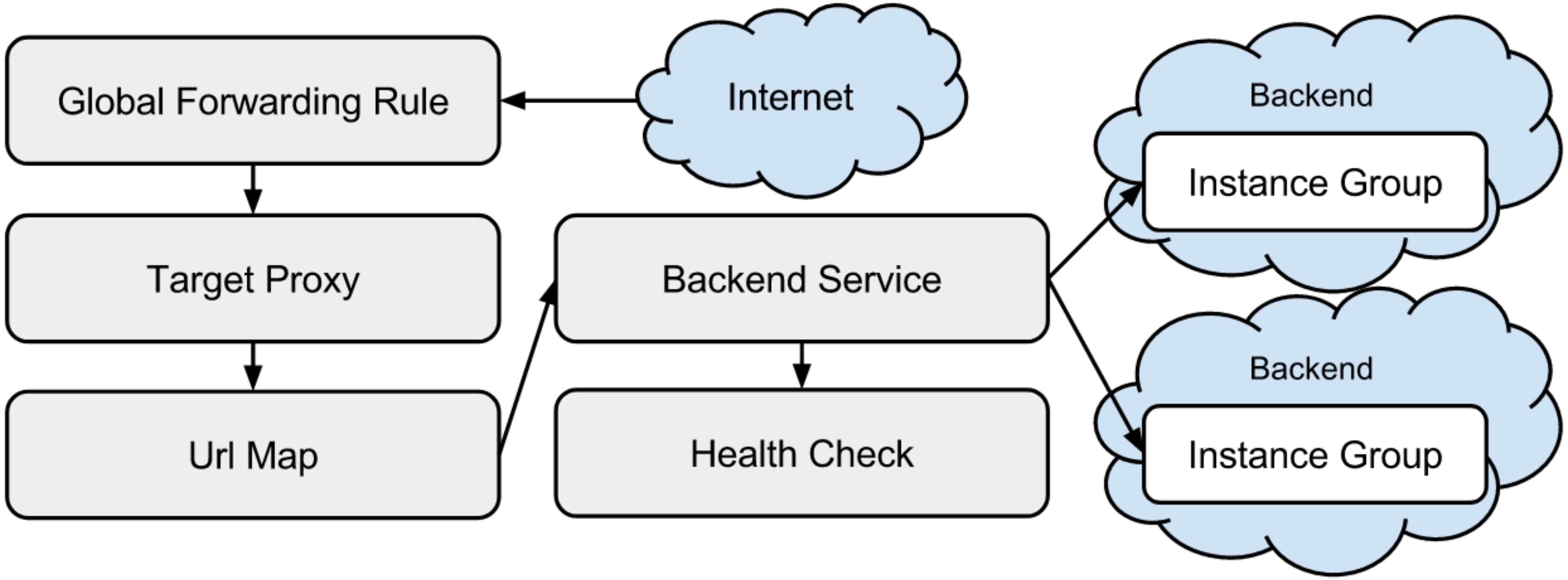


Target Proxy

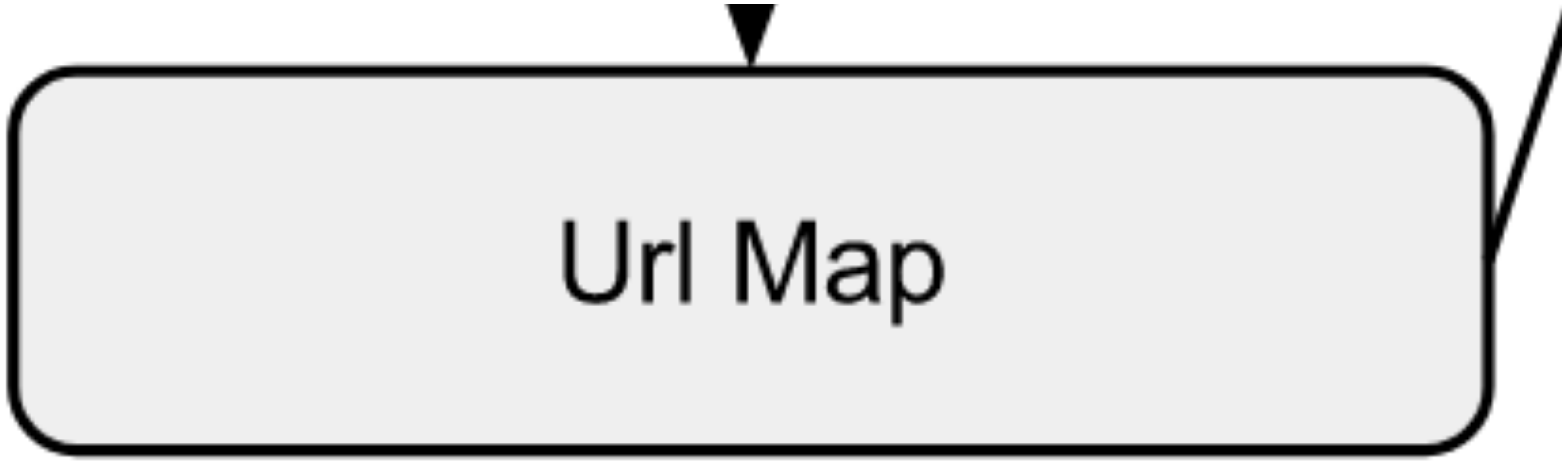
Target Proxy

- Referenced by one or more global forwarding rules
- Route the incoming requests to a URL map to determine where they should be sent
- Specific to a protocol (HTTP, HTTPS, SSL and TCP)
- Should have a SSL certificate if it terminates HTTPS connections (limit of 10 SSL certificates)
- Can connect to backend services via HTTP or HTTPS

HTTP/HTTPS Load Balancing



URL Map



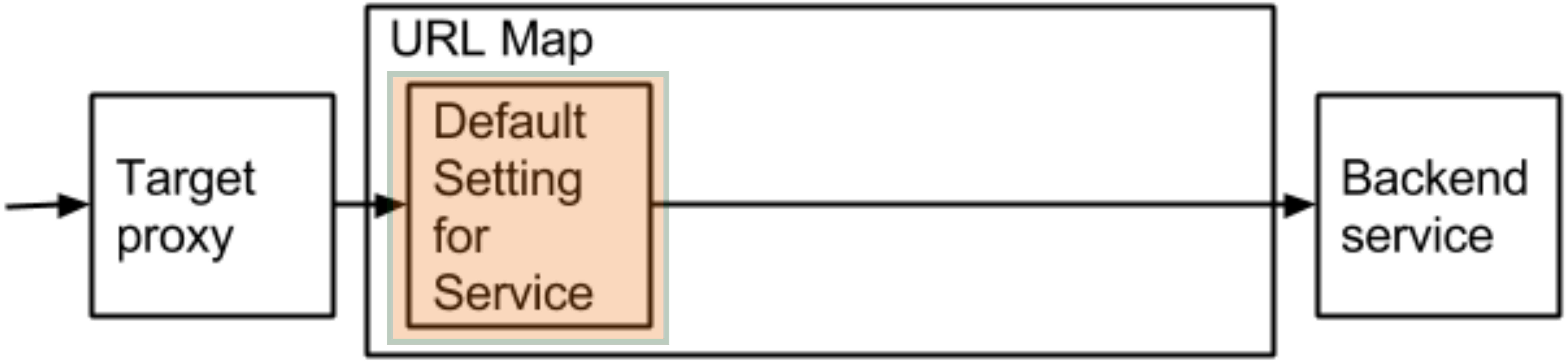
Url Map

URL Map

- Used to direct traffic to different instances based on the incoming URL
  - http://www.example.com/audio -> backend service1
  - http://www.example.com/vide -> backend service2

URL Map

URL map with no rules except default



Target proxy

URL Map

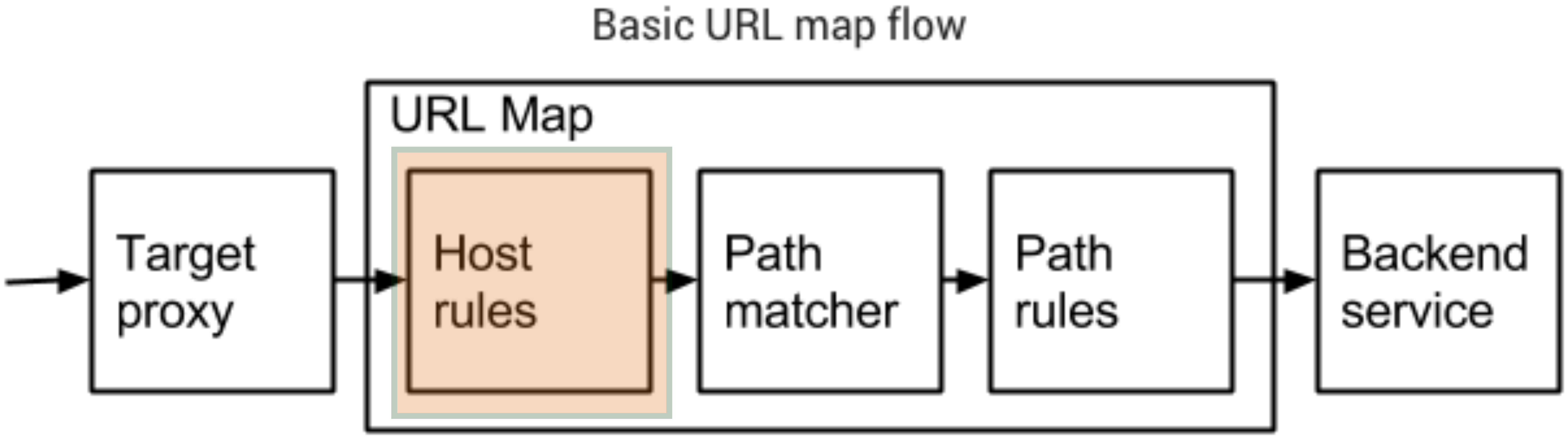
Default Setting for Service

Backend service

All traffic sent to the same groups of instances

Only the /* path matcher is created automatically and directs all traffic to the same backend service

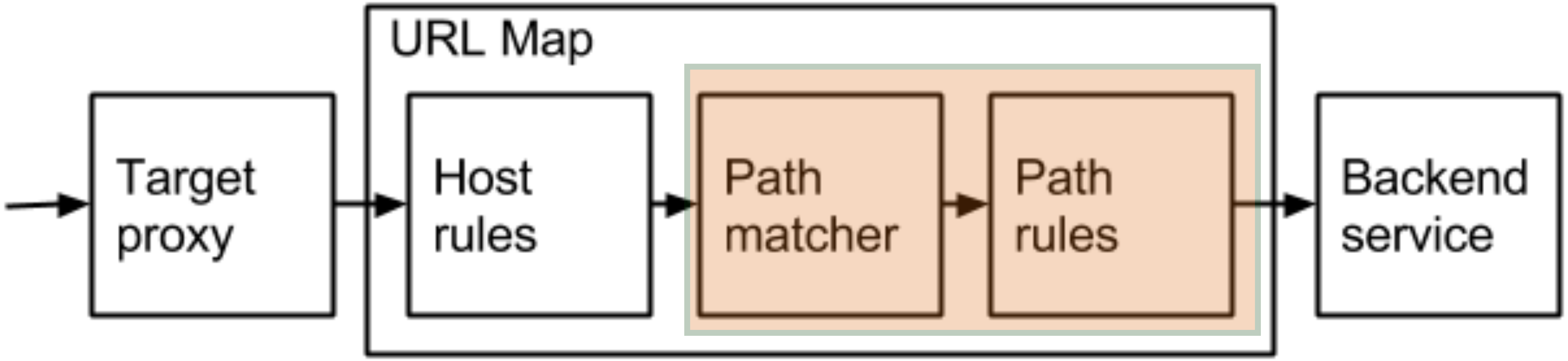
URL Map



Host rules — example.com, customer.com

URL Map

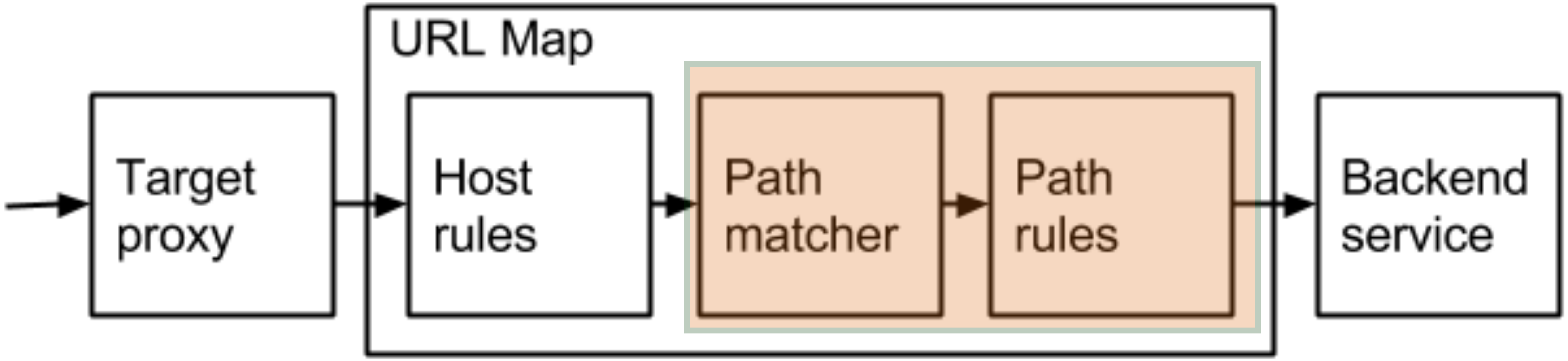
Basic URL map flow



Path rules — /video, /video/hd, /video/sd

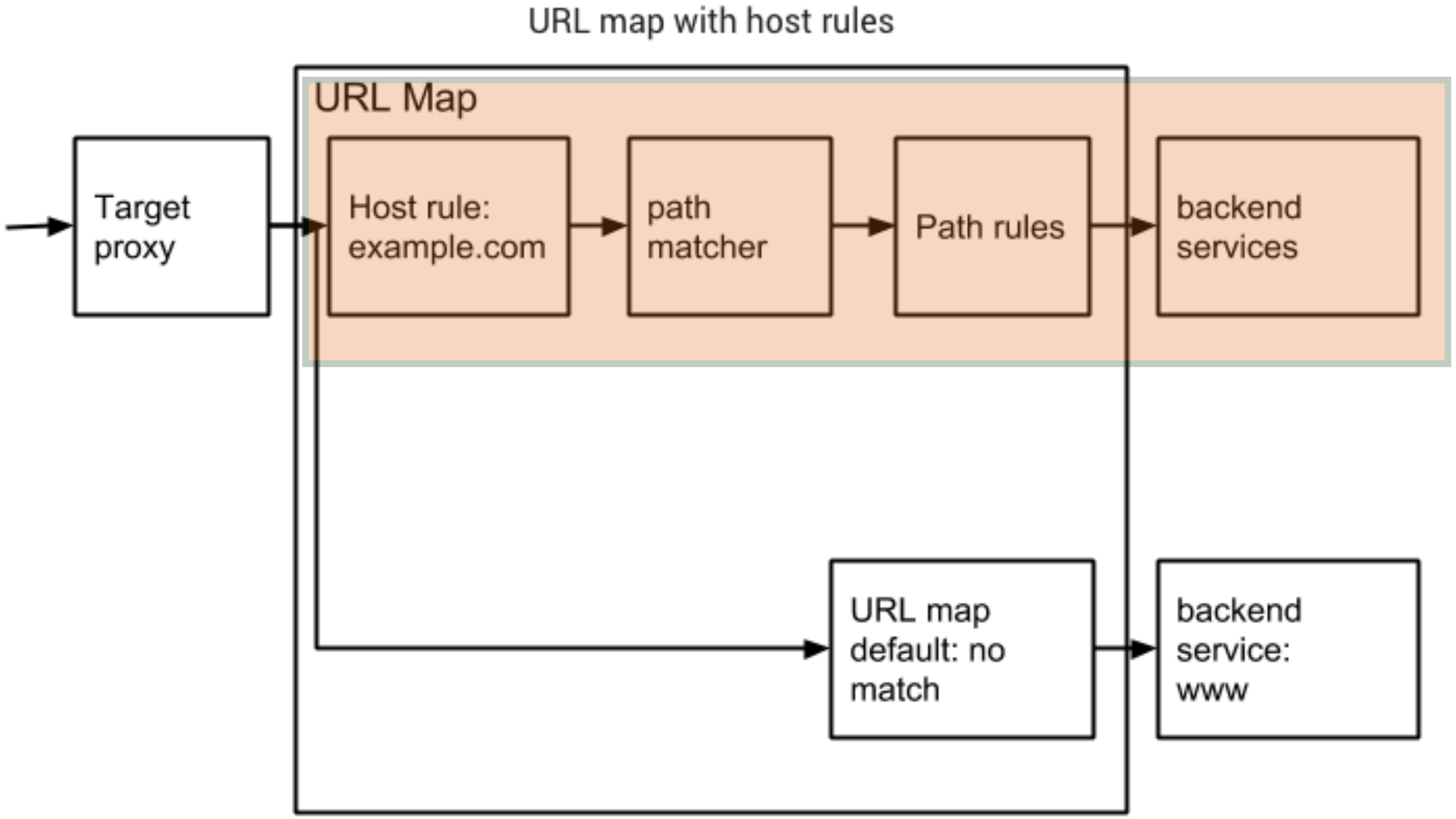
URL Map

Basic URL map flow



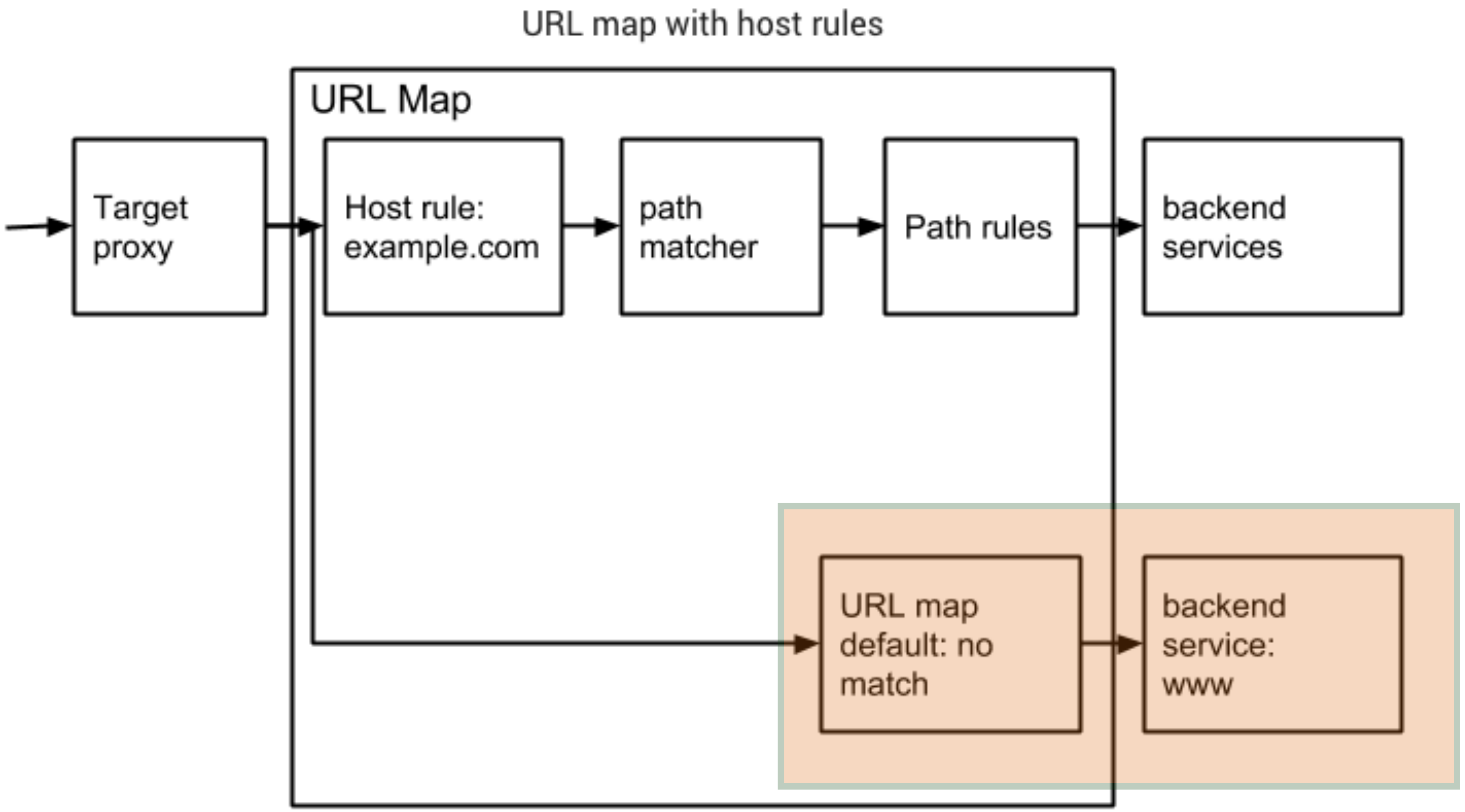
A default path matcher /* is created automatically. Traffic which does not match other path rules is sent to this default service

URL Map With Host Rule



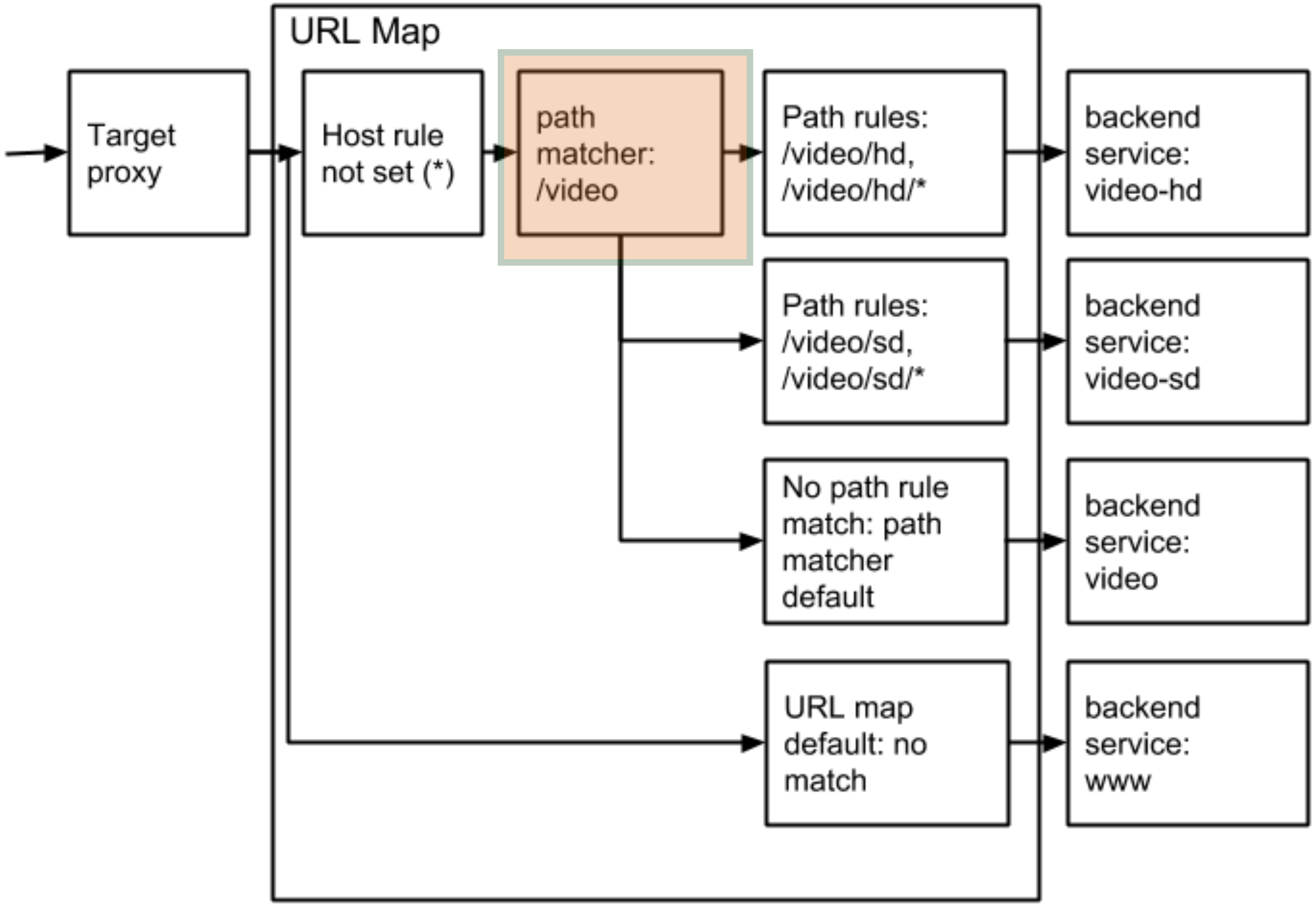
example.com requests will be sent to one set of backends

URL Map With Host Rule



Requests for all other hosts will go to the default backend

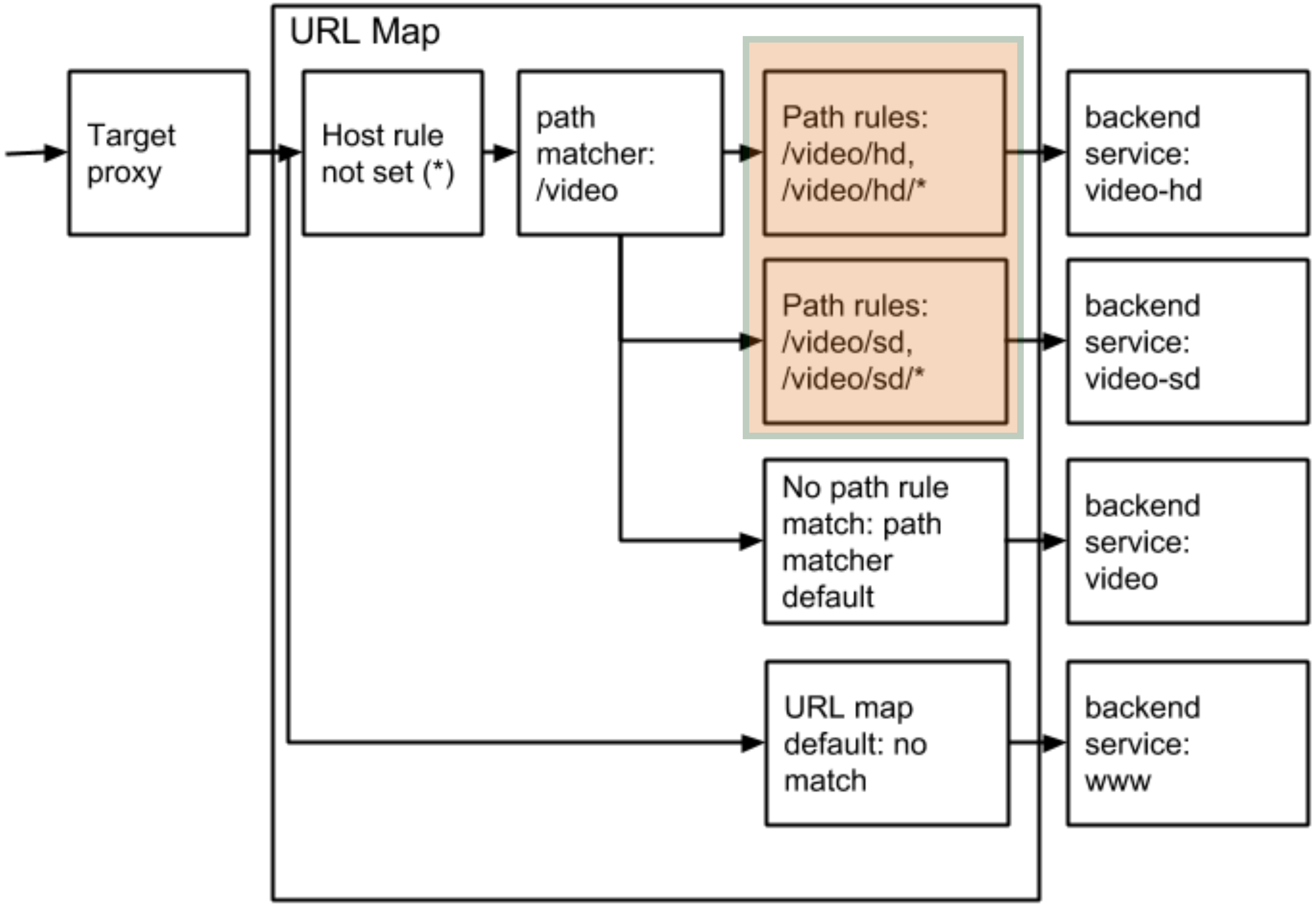
URL Map With Path Rules



URL Map
Target proxy
Host rule not set (*)
path matcher: /video
Path rules: /video/hd, /video/hd/*
backend service: video-hd
Path rules: /video/sd, /video/sd/*
backend service: video-sd
No path rule match: path matcher default
backend service: video
URL map default: no match
backend service: www

Path rules for video

URL Map With Path Rules



URL Map

Target proxy

Host rule not set (*)

path matcher: /video

Path rules: /video/hd, /video/hd/*

Path rules: /video/sd, /video/sd/*

No path rule match: path matcher default

URL map default: no match

backend service: video-hd

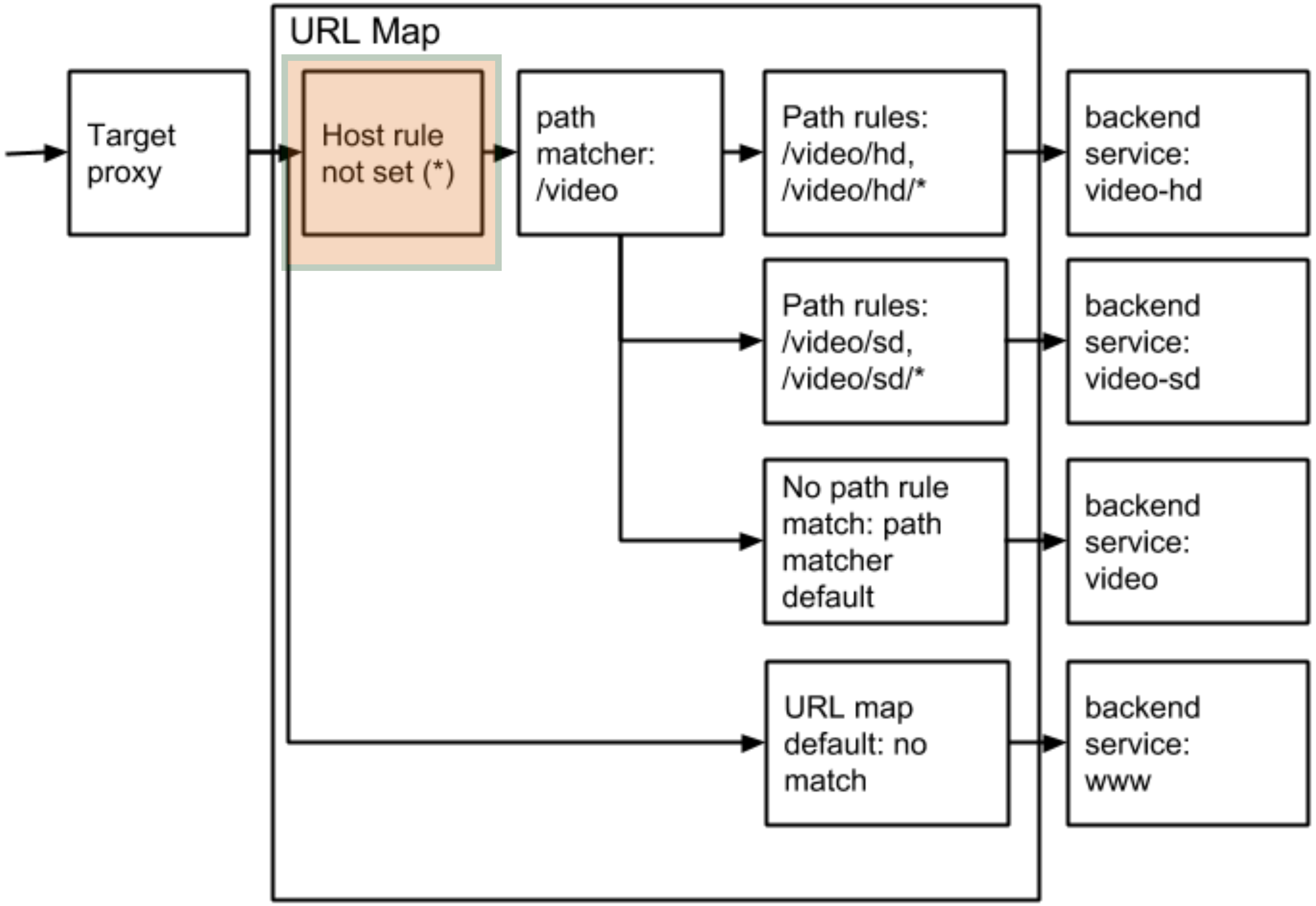
backend service: video-sd

backend service: video

backend service: www

More specific rules for /video/sd and /video/hd

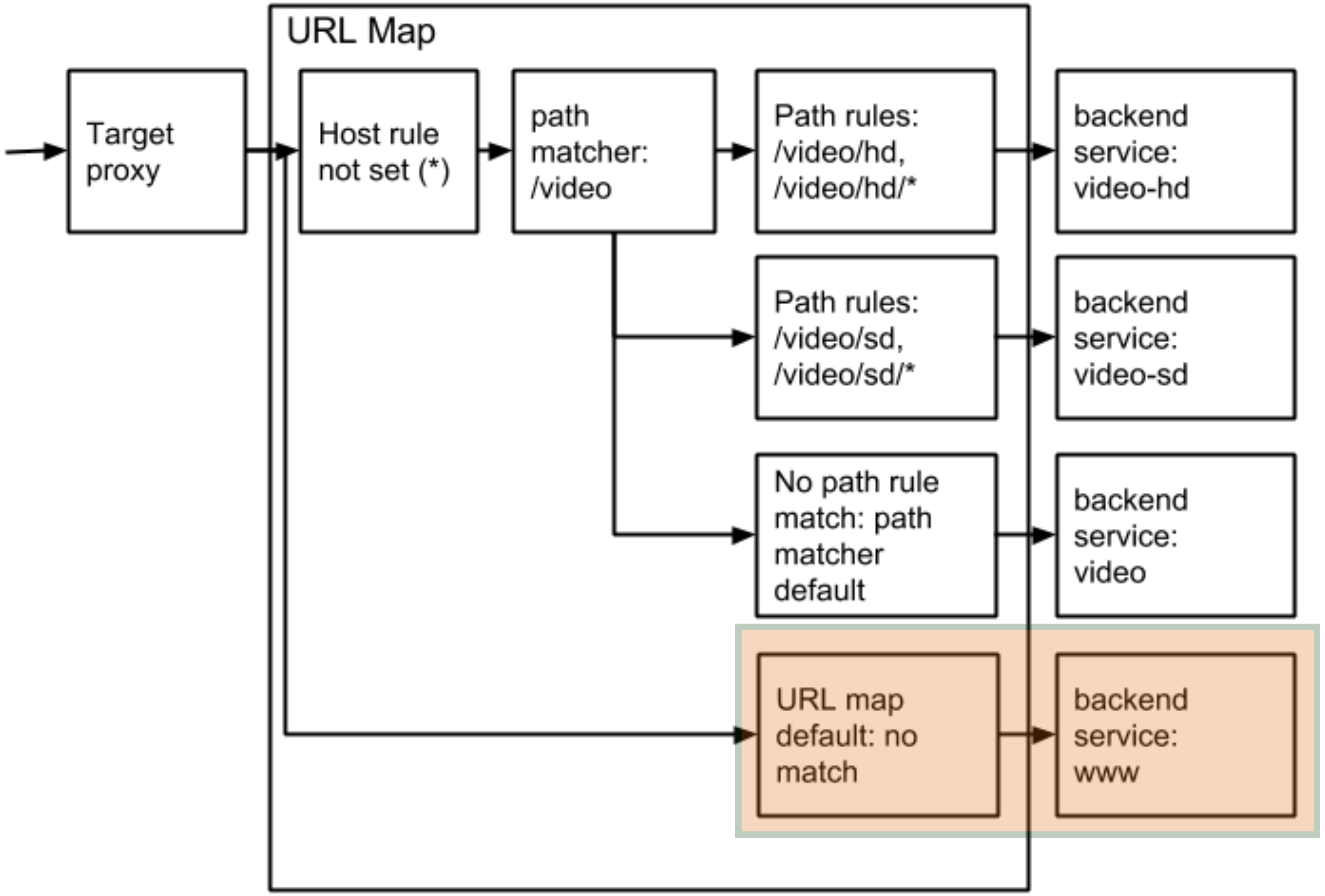
URL Map With Path Rules



URL Map
Target proxy
Host rule not set (*)
path matcher: /video
Path rules: /video/hd, /video/hd/*
Path rules: /video/sd, /video/sd/*
No path rule match: path matcher default
URL map default: no match
backend service: video-hd
backend service: video-sd
backend service: video
backend service: www

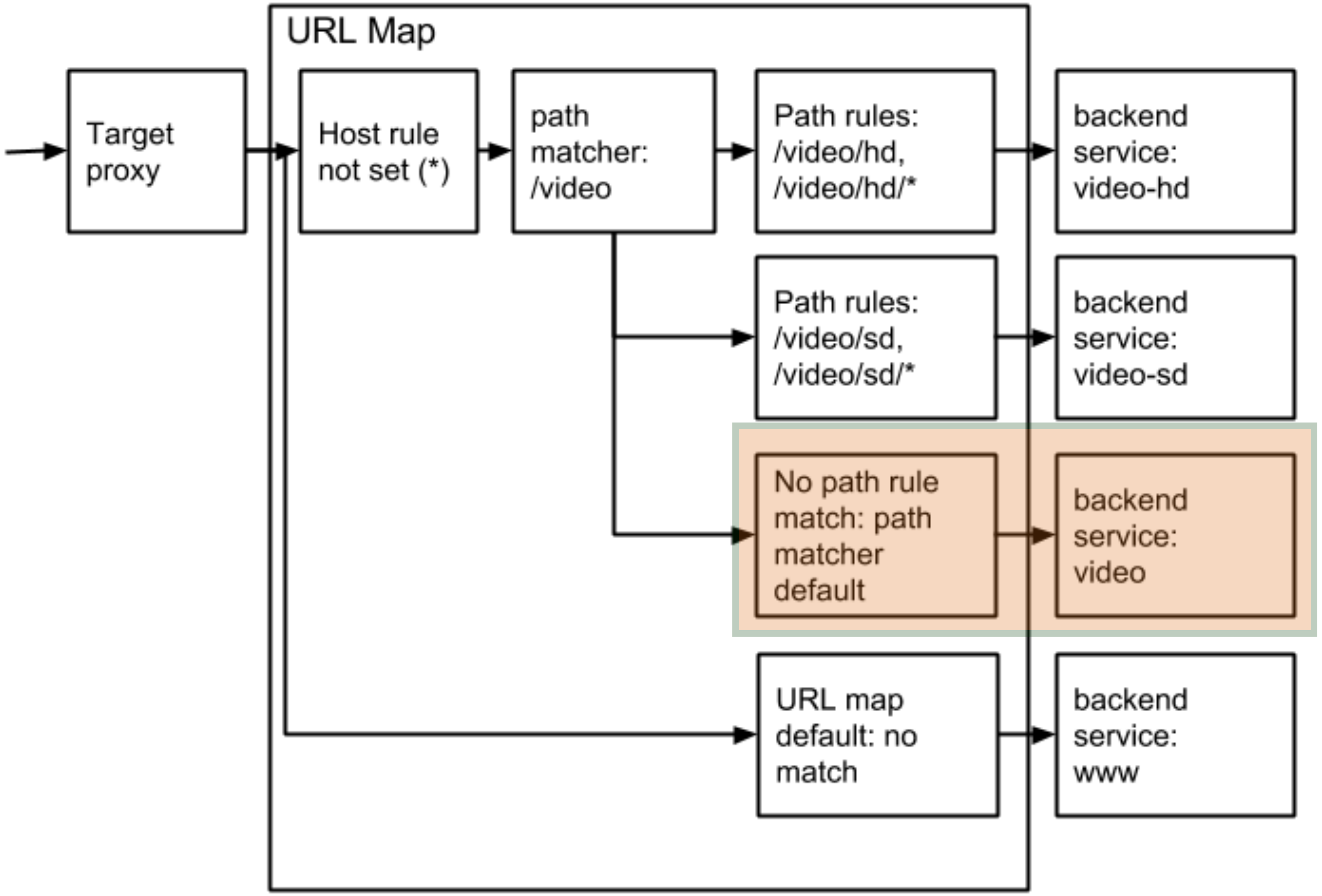
No host name

URL Map With Path Rules


URL Map
Target proxy
Host rule not set (*)
path matcher: /video
Path rules: /video/hd, /video/hd/*
backend service: video-hd
Path rules: /video/sd, /video/sd/*
backend service: video-sd
No path rule match: path matcher default
backend service: video
URL map default: no match
backend service: www

Default backend service when no path rule matches

URL Map With Path Rules



URL Map

Target proxy

Host rule not set (*)

path matcher: /video

Path rules: /video/hd, /video/hd/*

backend service: video-hd

Path rules: /video/sd, /video/sd/*

backend service: video-sd

No path rule match: path matcher default

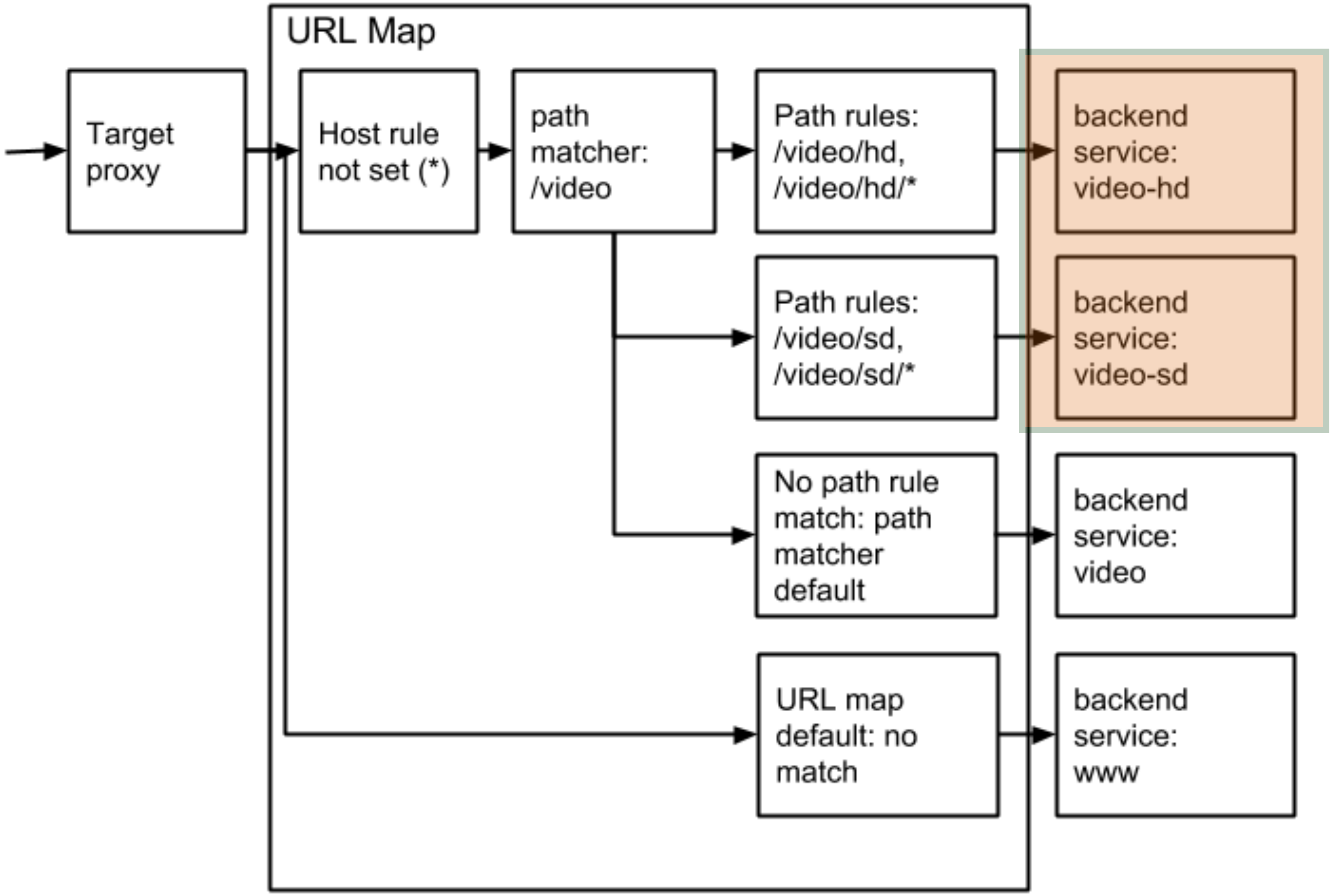
backend service: video

URL map default: no match

backend service: www

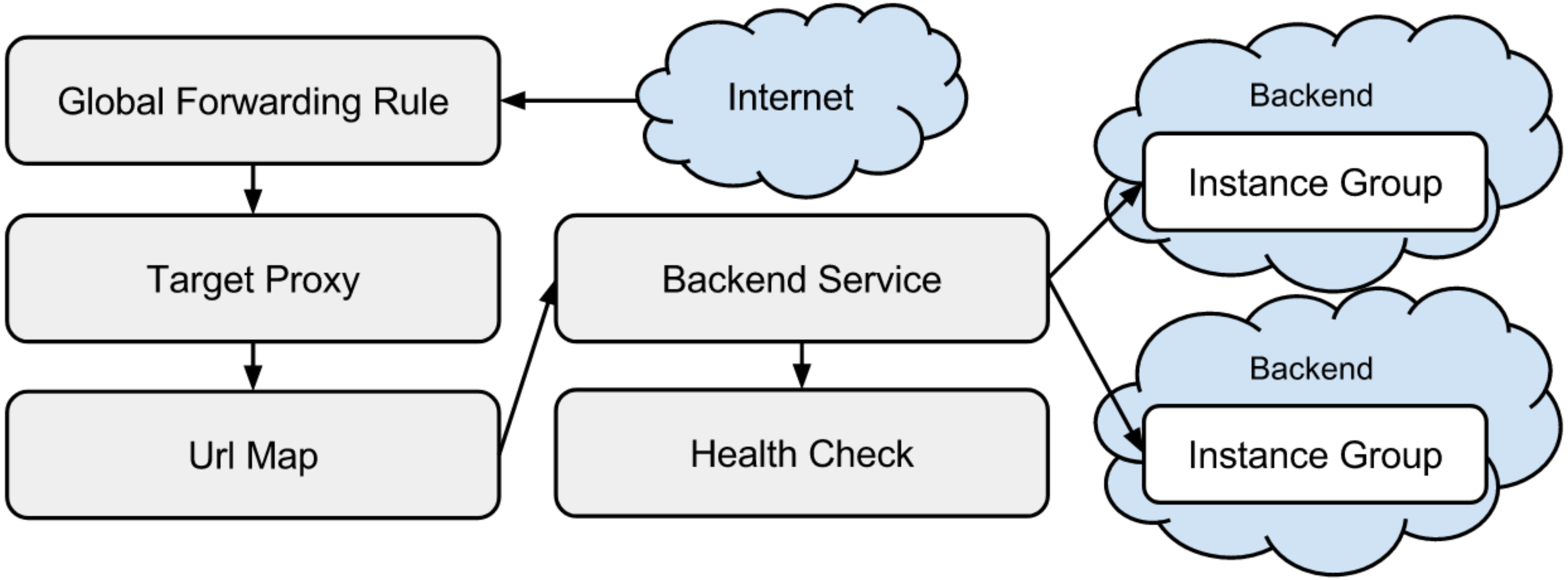
Path **other than** /video/hd and /video/sd

URL Map With Path Rules

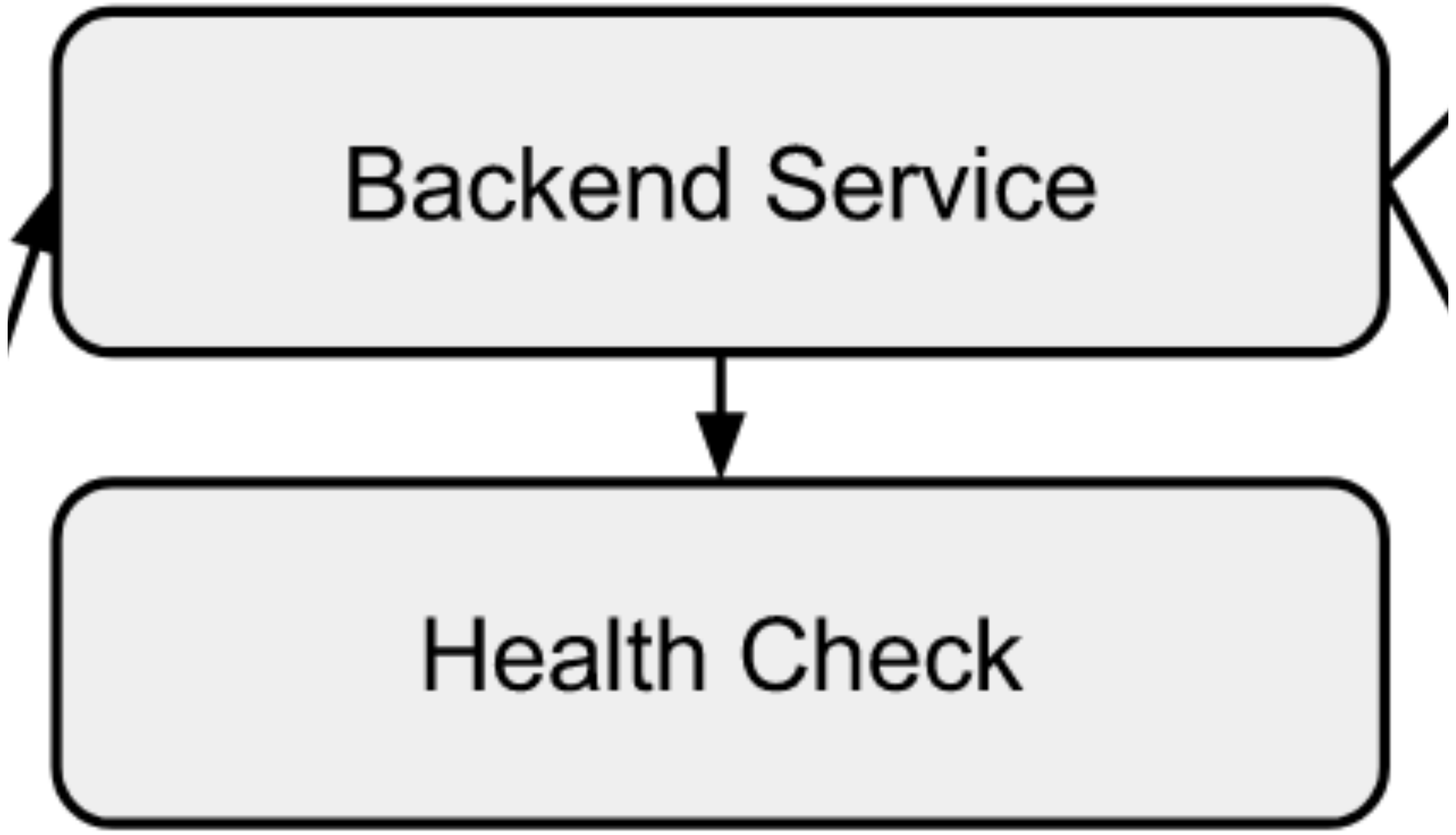

URL Map
Target proxy
Host rule not set (*)
path matcher: /video
Path rules: /video/hd, /video/hd/*
Path rules: /video/sd, /video/sd/*
No path rule match: path matcher default
URL map default: no match
backend service: video-hd
backend service: video-sd
backend service: video
backend service: www

Backends for paths which match /video/hd and /video/sd

HTTP/HTTPS Load Balancing



Backend Service



Backend Service

- Centralized service for managing backends
- Backends contain instance groups which handle user requests
- Knows which instances it can use, how much traffic they can handle
- Monitors the health of backends and does not send traffic to unhealthy instances

Backend Service Components

- **Health Check:** Pools instances to determine which one can receive requests
- **Backends:** Instance group of VMs which can be automatically scaled
- **Session Affinity:** Attempts to send requests from the same client to the same VM
- **Timeout:** Time the backend service will wait for a backend to respond

Backend Service Components

- **Health Check:** Pools instances to determine which one can receive requests
- **Backends:** Instance group of VMs which can be automatically scaled
- **Session Affinity:** Attempts to send requests from the same client to the same VM
- **Timeout:** Time the backend service will wait for a backend to respond

Health Checks

- HTTP(S), SSL and TCP health checks
- HTTP(S): Verifies that the instance is healthy and the web server is serving traffic
- TCP, SSL: Used when the service expects TCP or SSL connection i.e. not HTTP(S)
- GCP creates redundant copies of the health checker automatically so health checks might happen more frequently that you expect

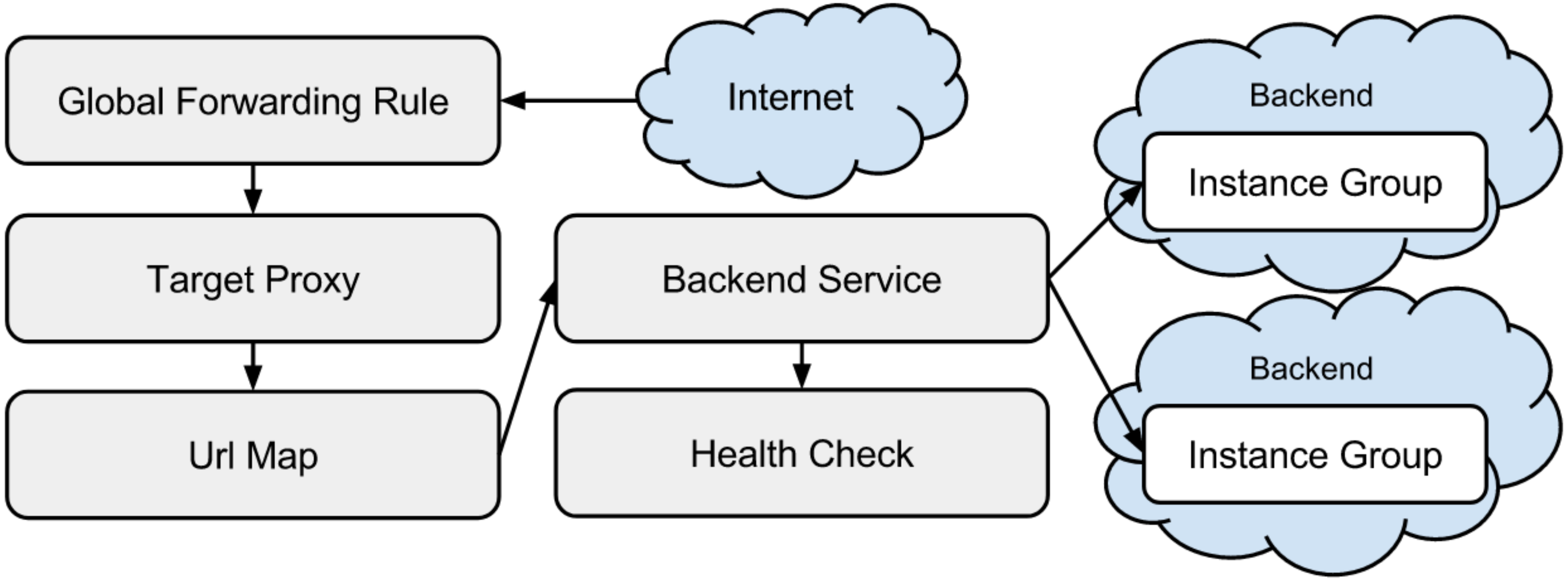
Backend Service Components

- **Health Check:** Pools instances to determine which one can receive requests
- **Backends:** Instance group of VMs which can be automatically scaled
- **Session Affinity:** Attempts to send requests from the same client to the same VM
- **Timeout:** Time the backend service will wait for a backend to respond

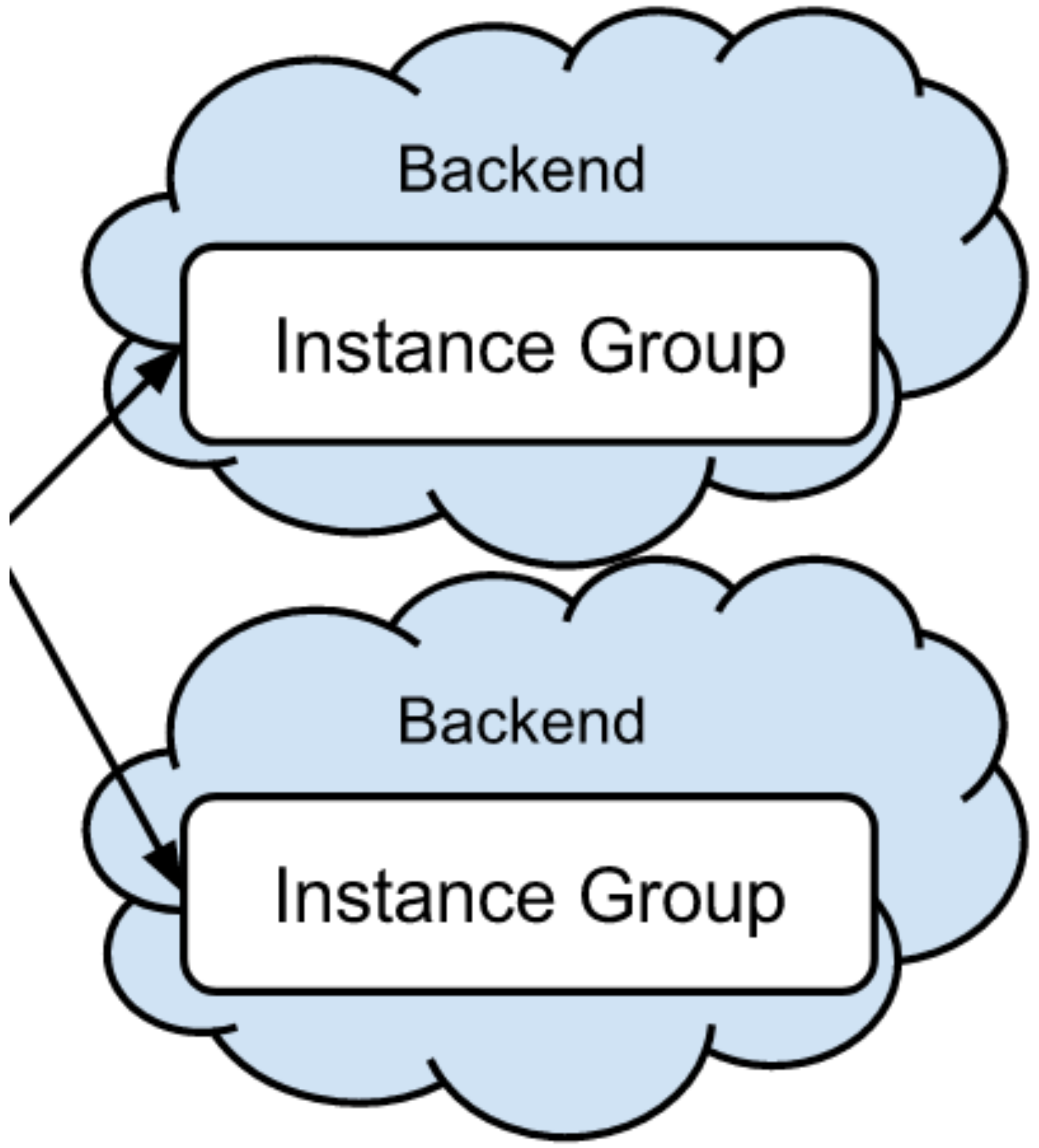
Session Affinity

- **Client IP:** Hashes the IP address to send requests from the same IP to the same VM
    - Requests from different users might look like it is from the same IP
    - Users which move networks might lose affinity
- **Cookie:** Issues a cookie named GCLB in the first request.
    - Subsequent requests from clients with the cookie are sent to the same instance

HTTP/HTTPS Load Balancing



Backend



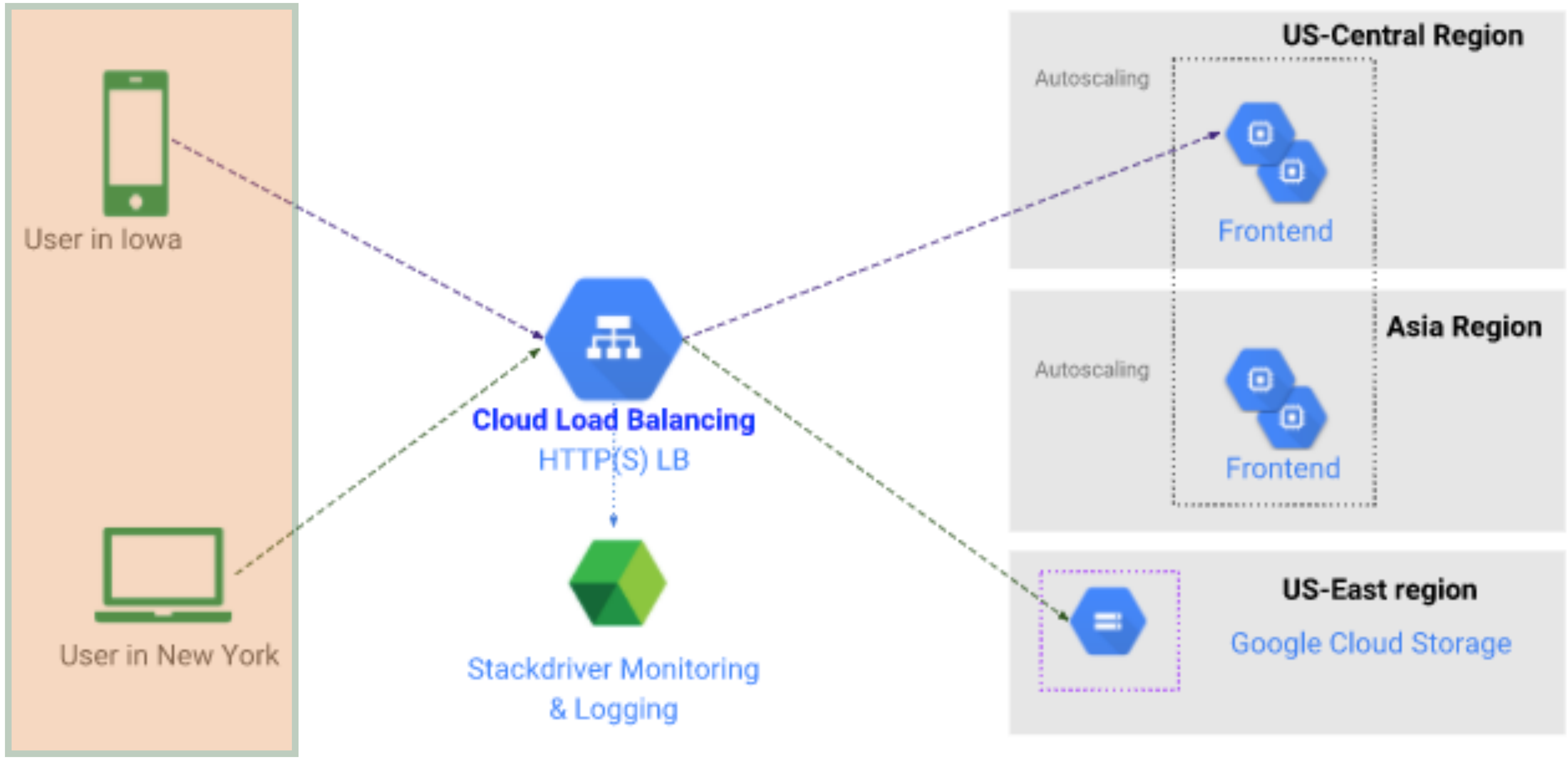
Backends

- **Instance Group:** Can be a managed or unmanaged instance group
- **Balancing Mode:** Determines when the backend is at full usage
  - CPU utilization, Requests per second
- **Capacity Setting:** A % of the balancing mode which determines the capacity of the backend

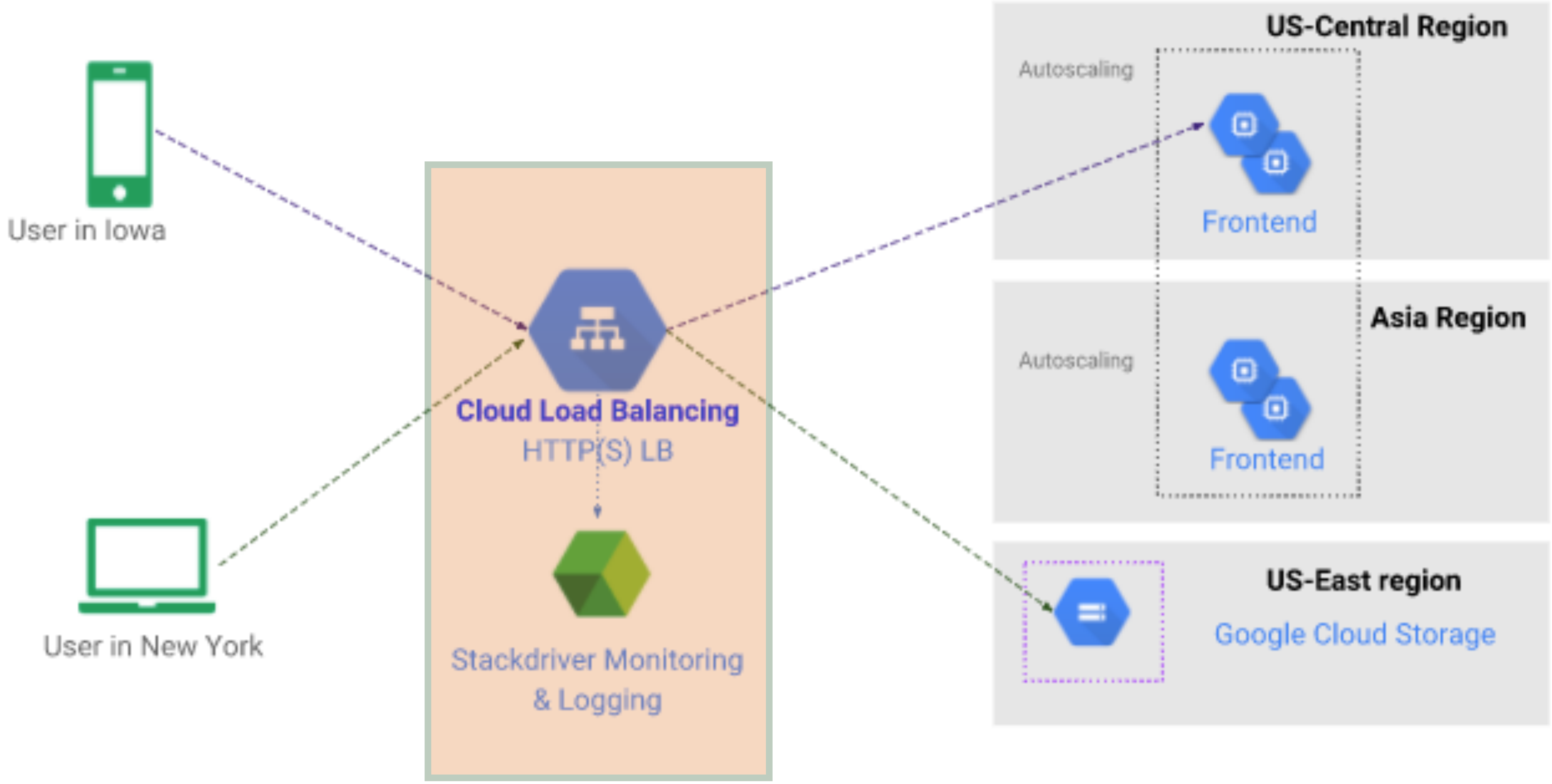
Backend Buckets

- Allow you to use Cloud Storage buckets with HTTP(S) load balancing
- Traffic is directed to the bucket instead of a backend
- Useful in load balancing requests to **static content**

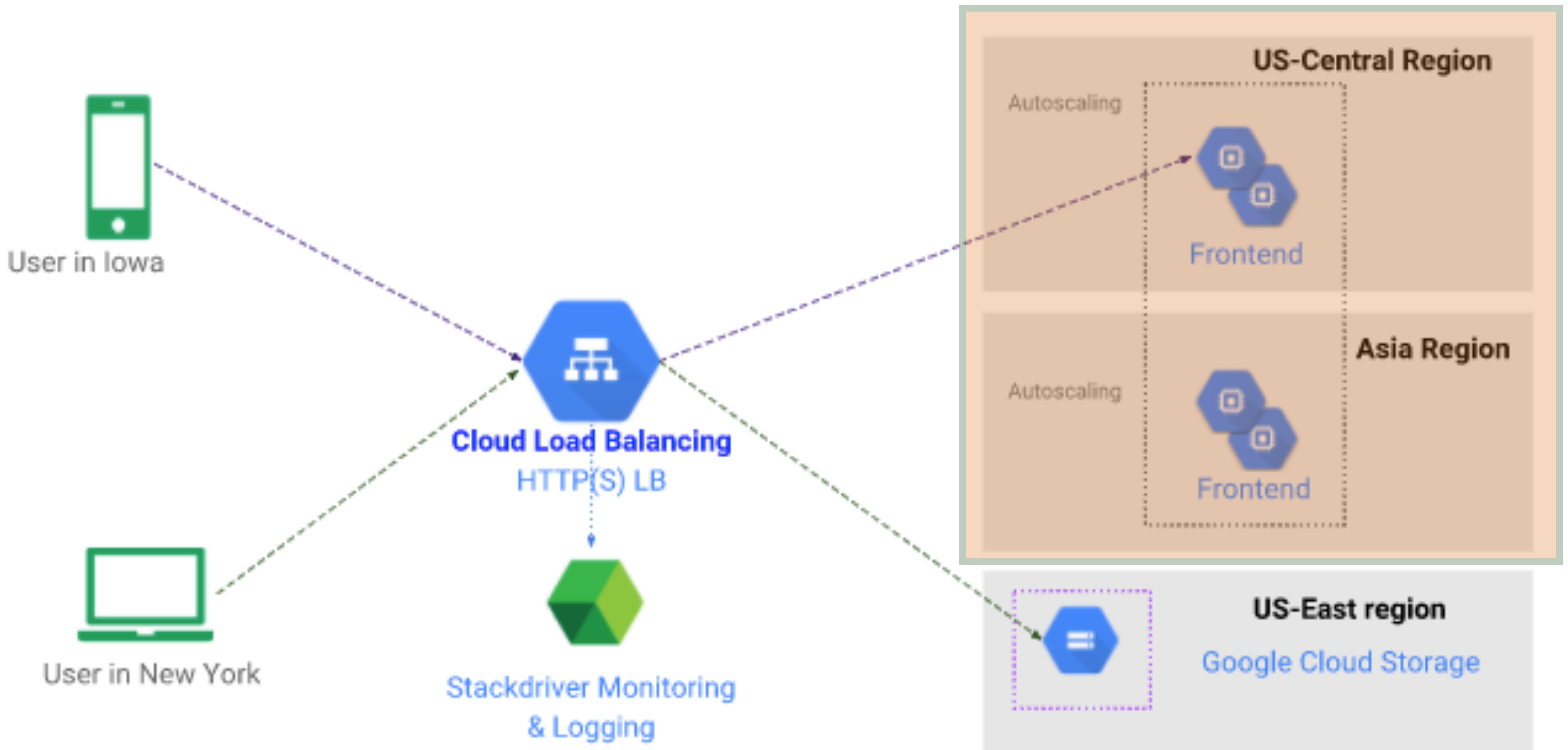
Backend Buckets



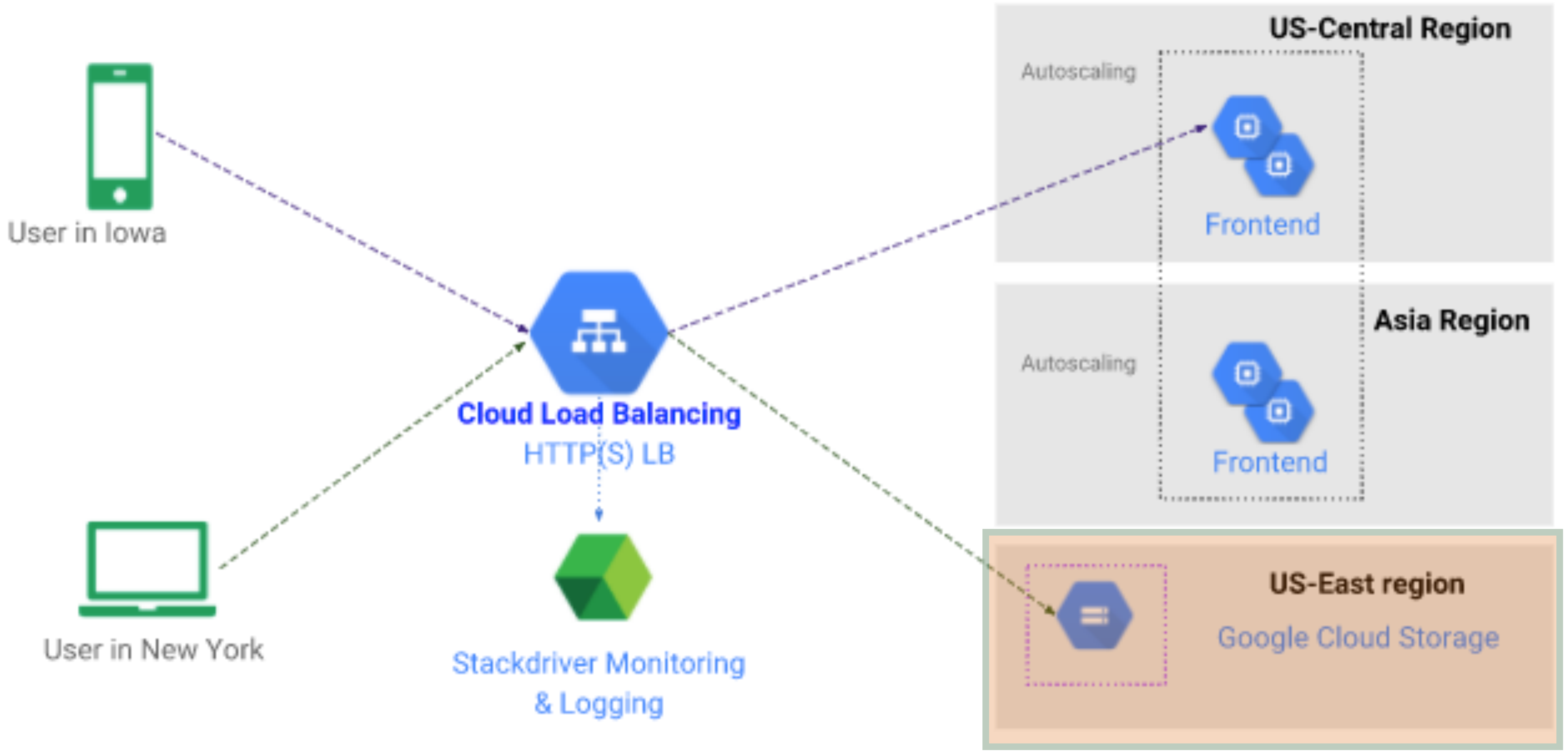
Backend Buckets



Backend Buckets



Backend Buckets



A path of /static can be sent to the storage bucket and all other paths go to the instances

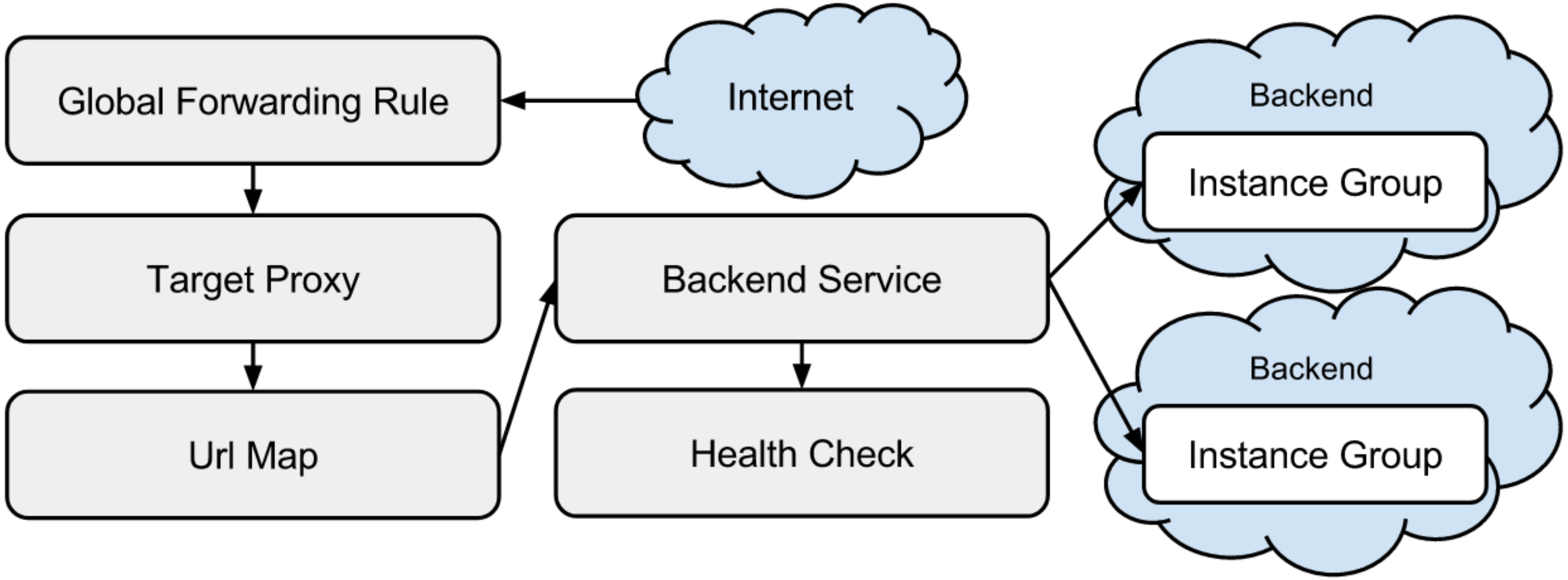
Load Distribution

- Uses CPU utilization of the backend or requests per second as the **balancing mode**
- Maximum values can be specified for both
- Short bursts of traffic above the limit can occur

Load Distribution

- Incoming requests are first sent to the **region closest to the user**, if that region has capacity
- Traffic distributed amongst zone instances based on capacity
- Round robin distribution across instances in a zone
- Round robin can be overridden by session affinity

HTTP/HTTPS Load Balancing


Global Forwarding Rule

Internet

Target Proxy

Url Map

Backend Service

Health Check

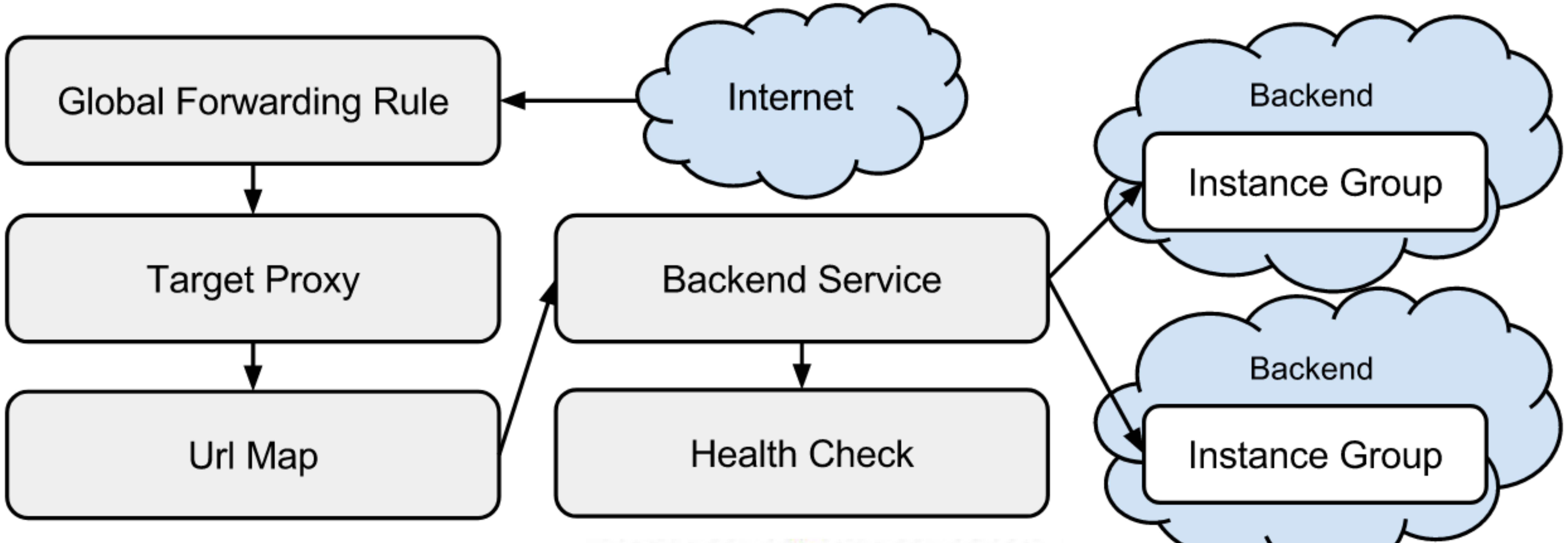
Backend

Instance Group

Backend

Instance Group

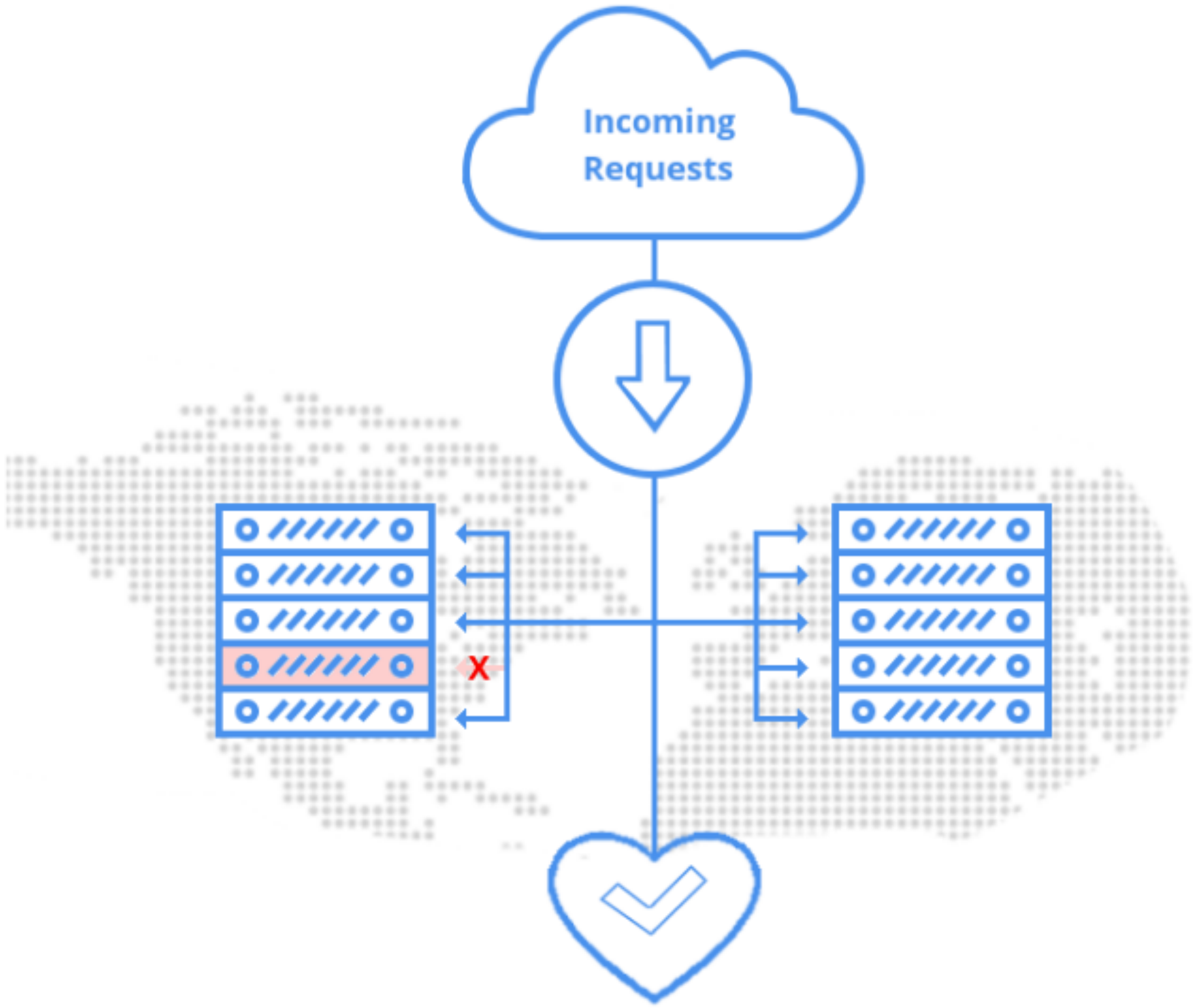
Firewall Rules



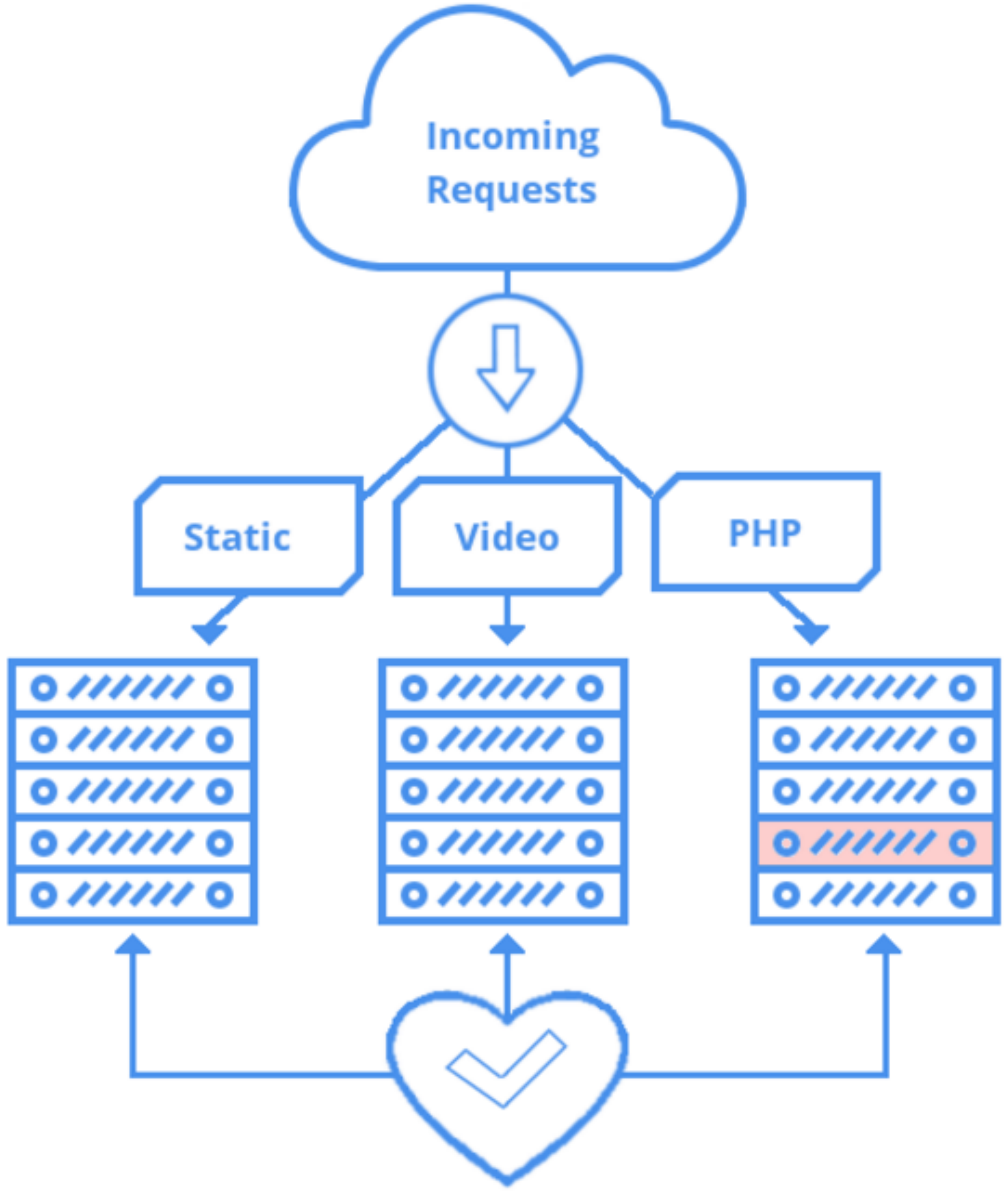
Firewall Rules

- Allow traffic from 130.211.0.0/22 and 35.191.0.0/16 to reach your instances
- IP ranges that the **load balancer** and the **health checker** use to connect to backends
- Allow traffic on the **port** that the global forwarding rule has been configured to use

HTTP/HTTPS Load Balancing

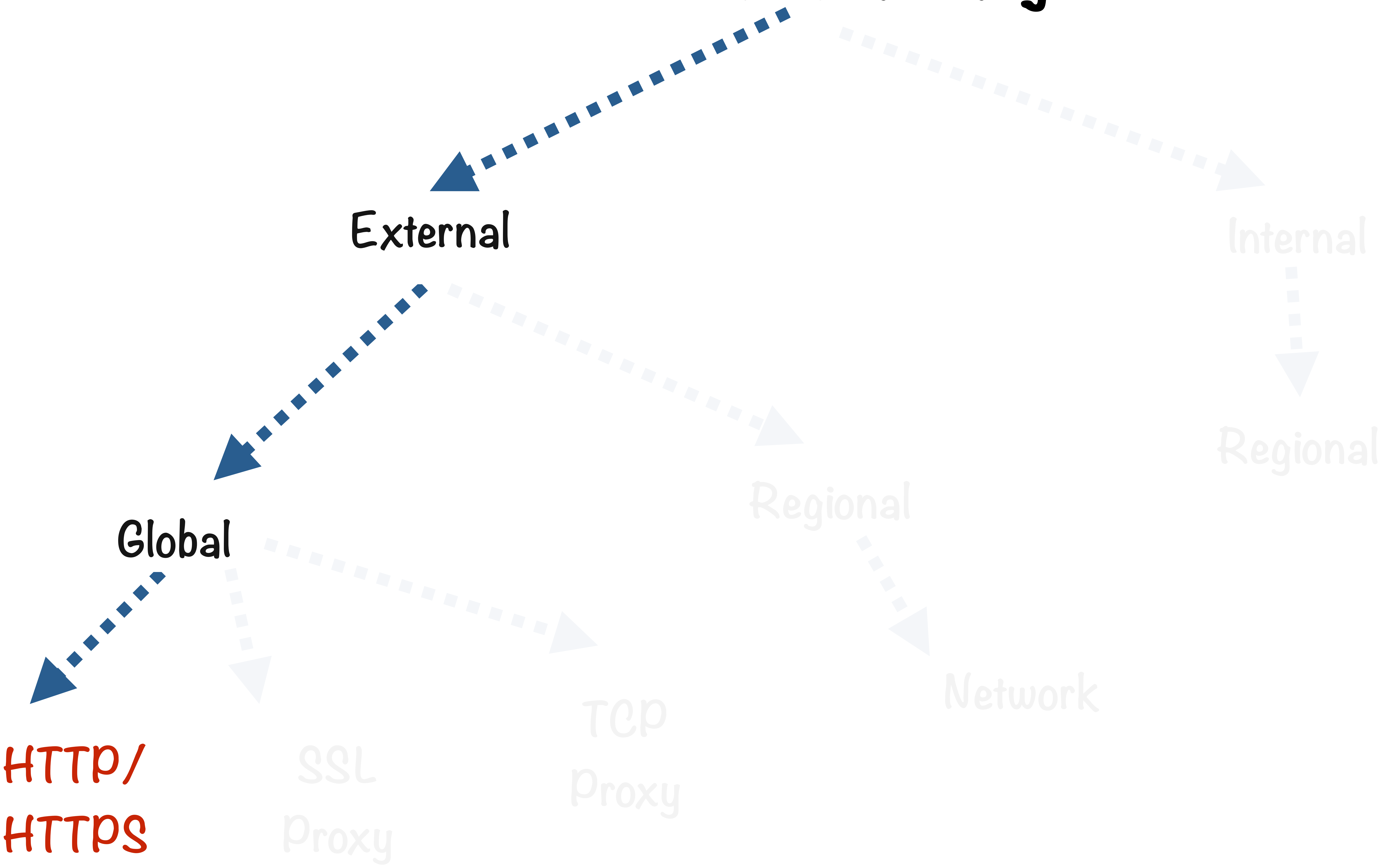


Cross-Regional



Content-based

Load Balancing



External

Internal

Global

Regional

Regional

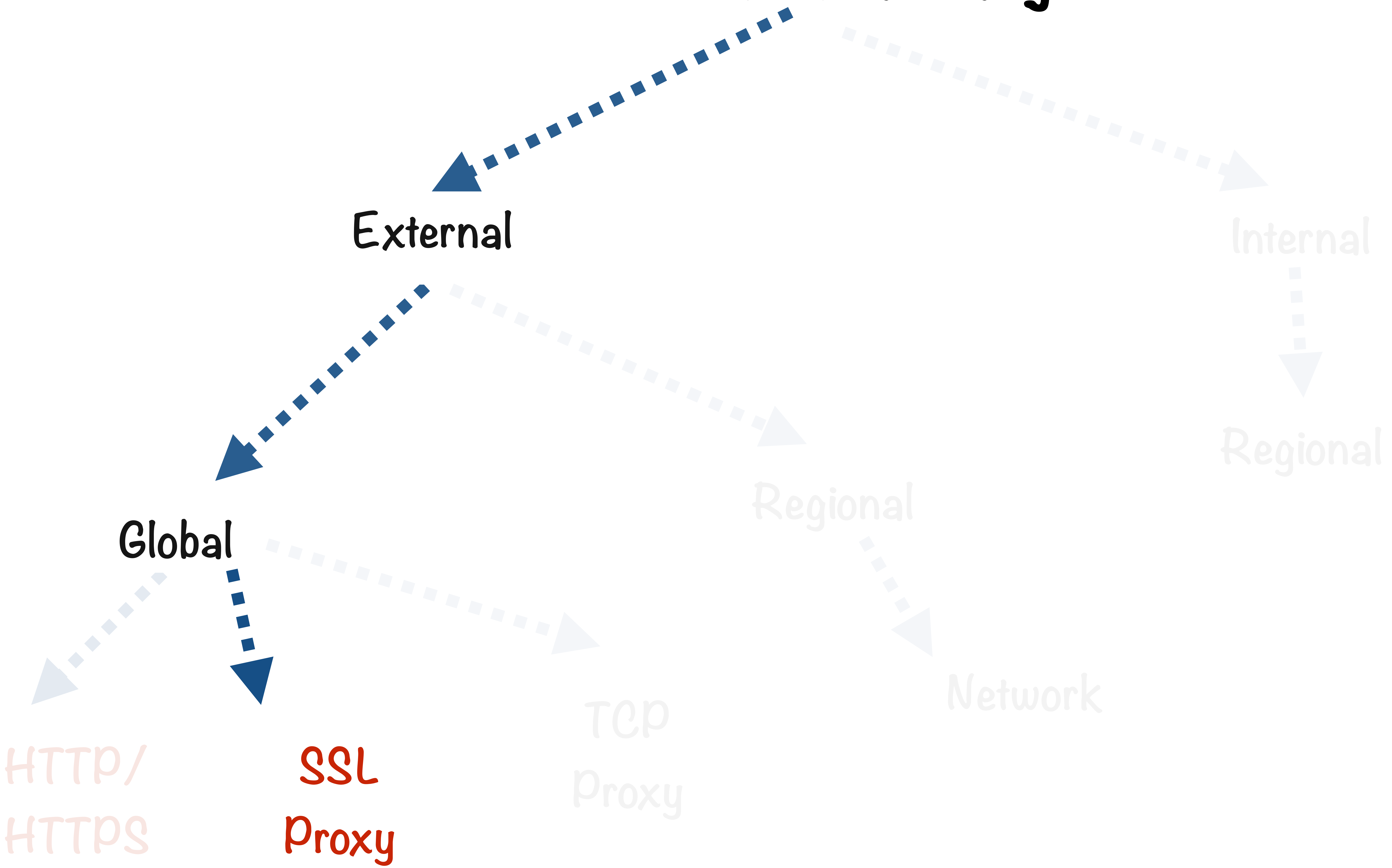
HTTP/
HTTPS

SSL
Proxy

TCP
Proxy

Network

Load Balancing



External

Internal

Global

Regional

Regional

HTTP/
HTTPS

SSL
Proxy

TCP
Proxy

Network

OSI Network Stack

User	
Application Layer	HTTP/HTTPS
Presentation Layer	
Session Layer	SSL Proxy
Transport Layer	TCP Proxy
Network Layer	Network
Data Link Layer	
Physical Layer	

SSL operates in the session layer

SSL Proxy Load Balancing

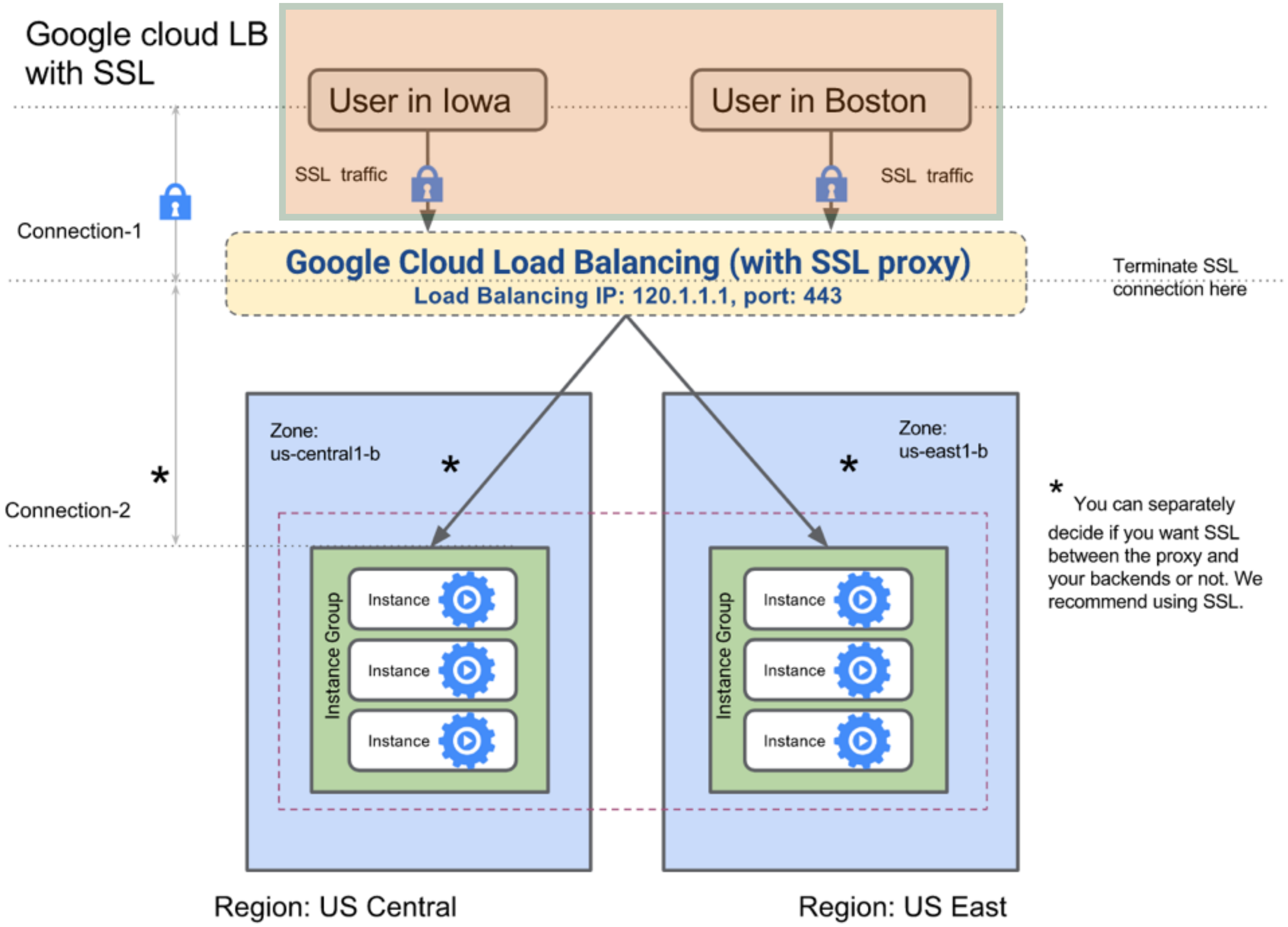
- Remember the OSI network layer stack: physical, data link, network, transport, session, presentation, application?
- The usual combination is TCP/IP: network = IP, transport = TCP, application = HTTP
- For secure traffic: add session layer = SSL (secure socket layer), and application layer = HTTPS

SSL Proxy Load Balancing

- Use only for non-HTTP(S) SSL traffic
- For HTTP(S), just use HTTP(S) load balancing
- SSL connections are terminated at the global layer then proxied to the closest available instance group

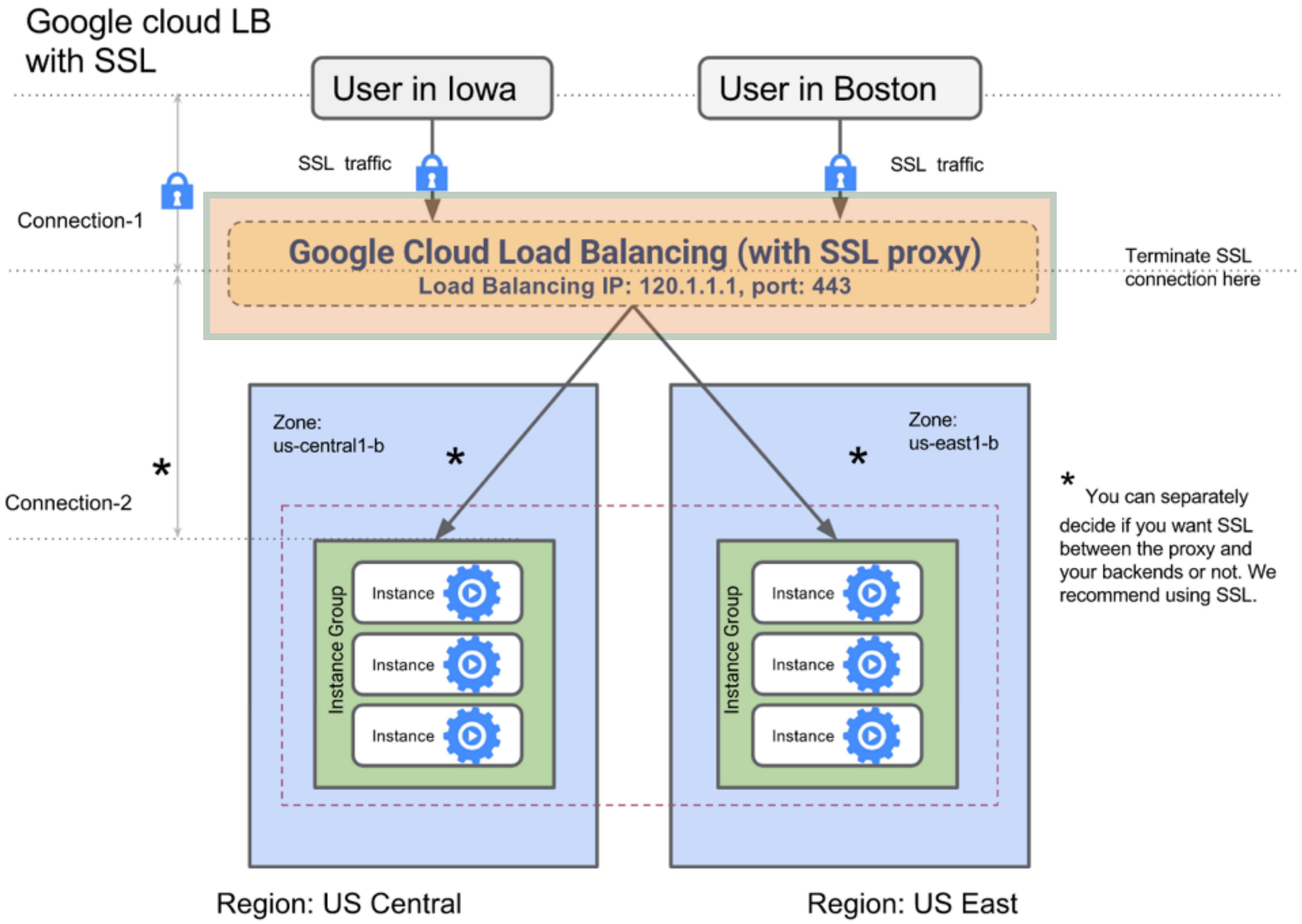
SSL Proxy Load Balancing

Users have a secure connection to the SSL proxy



SSL Proxy Load Balancing

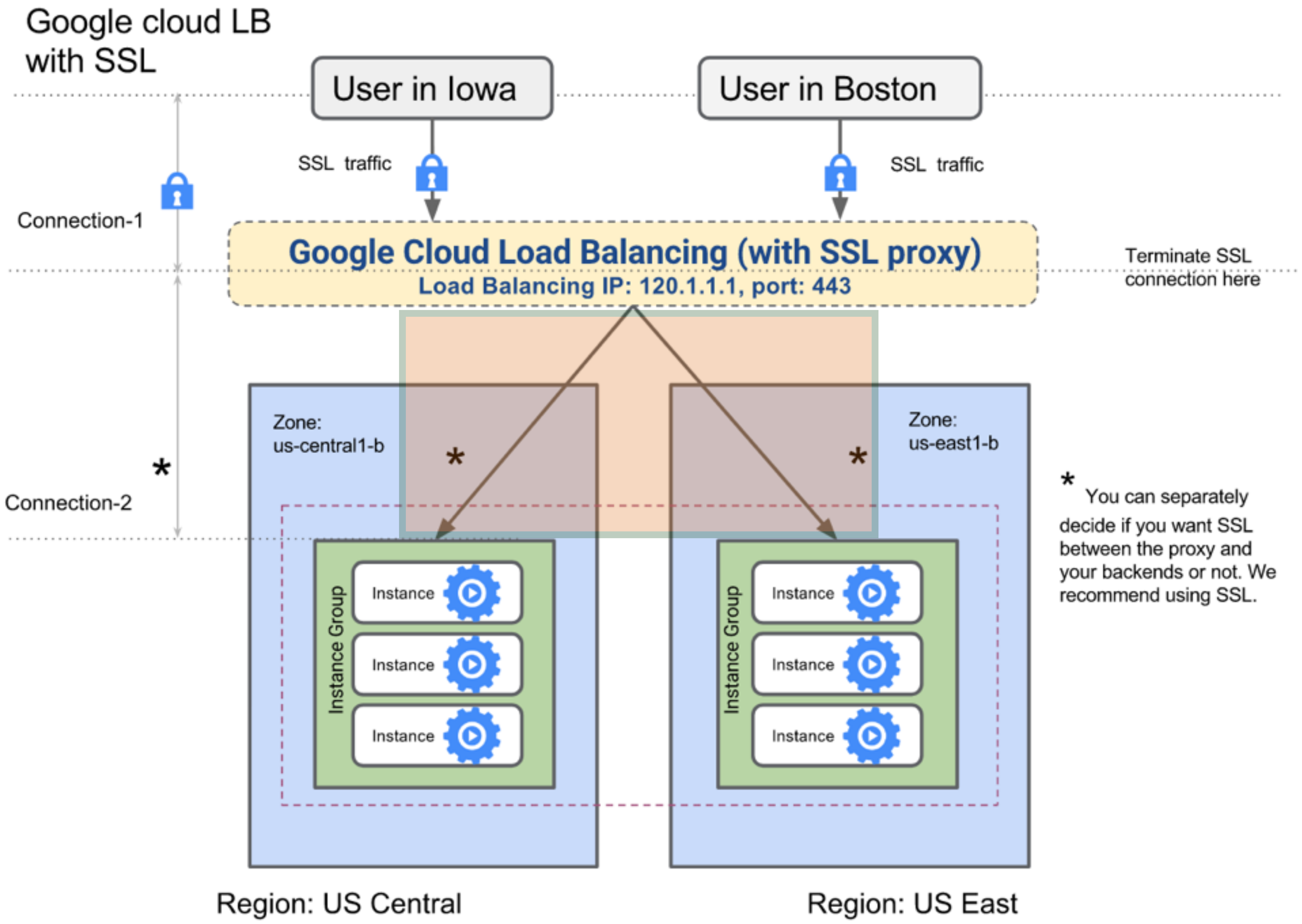
Load balancing SSL proxy



* You can separately decide if you want SSL between the proxy and your backends or not. We recommend using SSL.

SSL Proxy Load Balancing

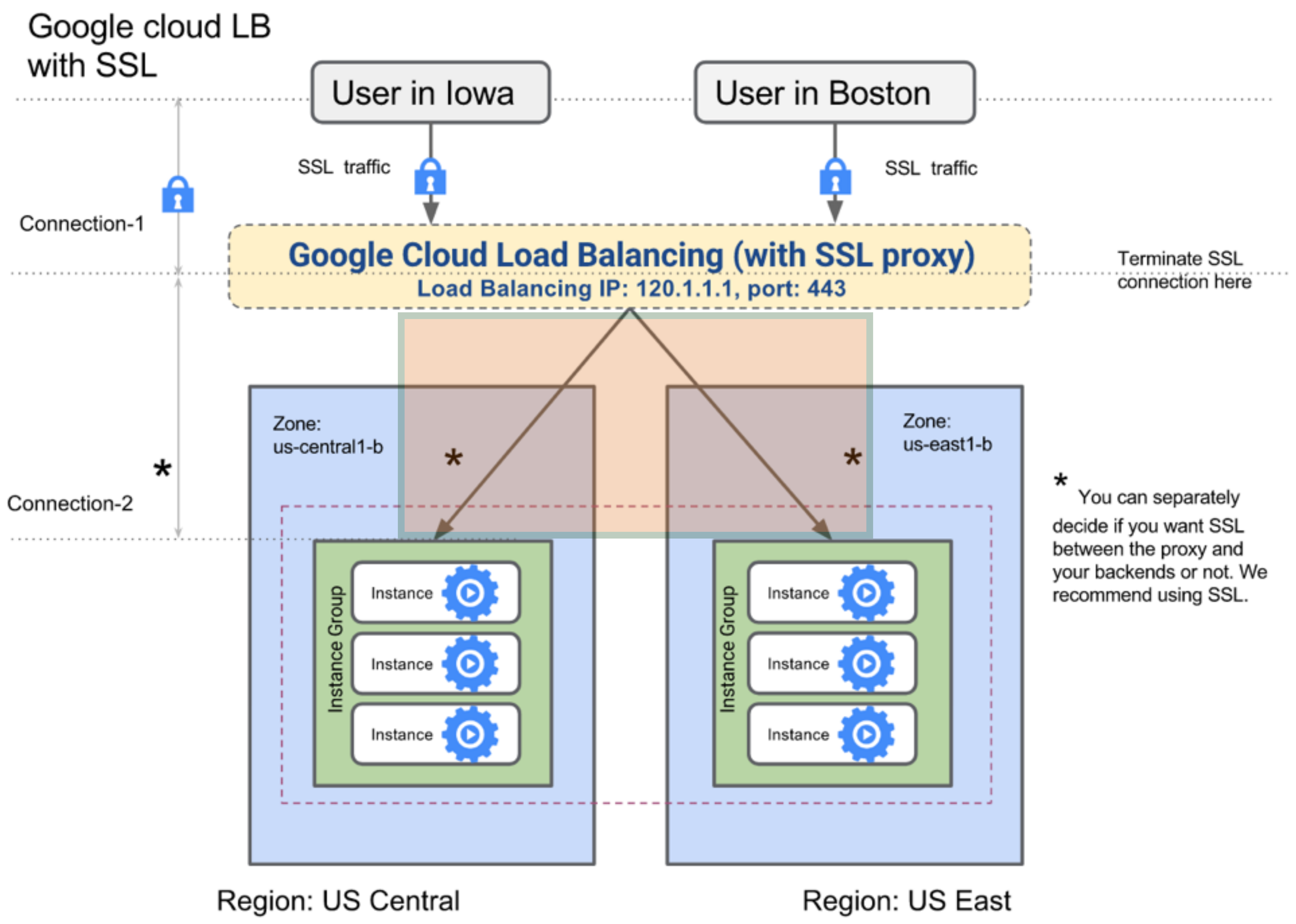
Makes fresh connections to the backends - this connection can be SSL or non-SSL



* You can separately decide if you want SSL between the proxy and your backends or not. We recommend using SSL.

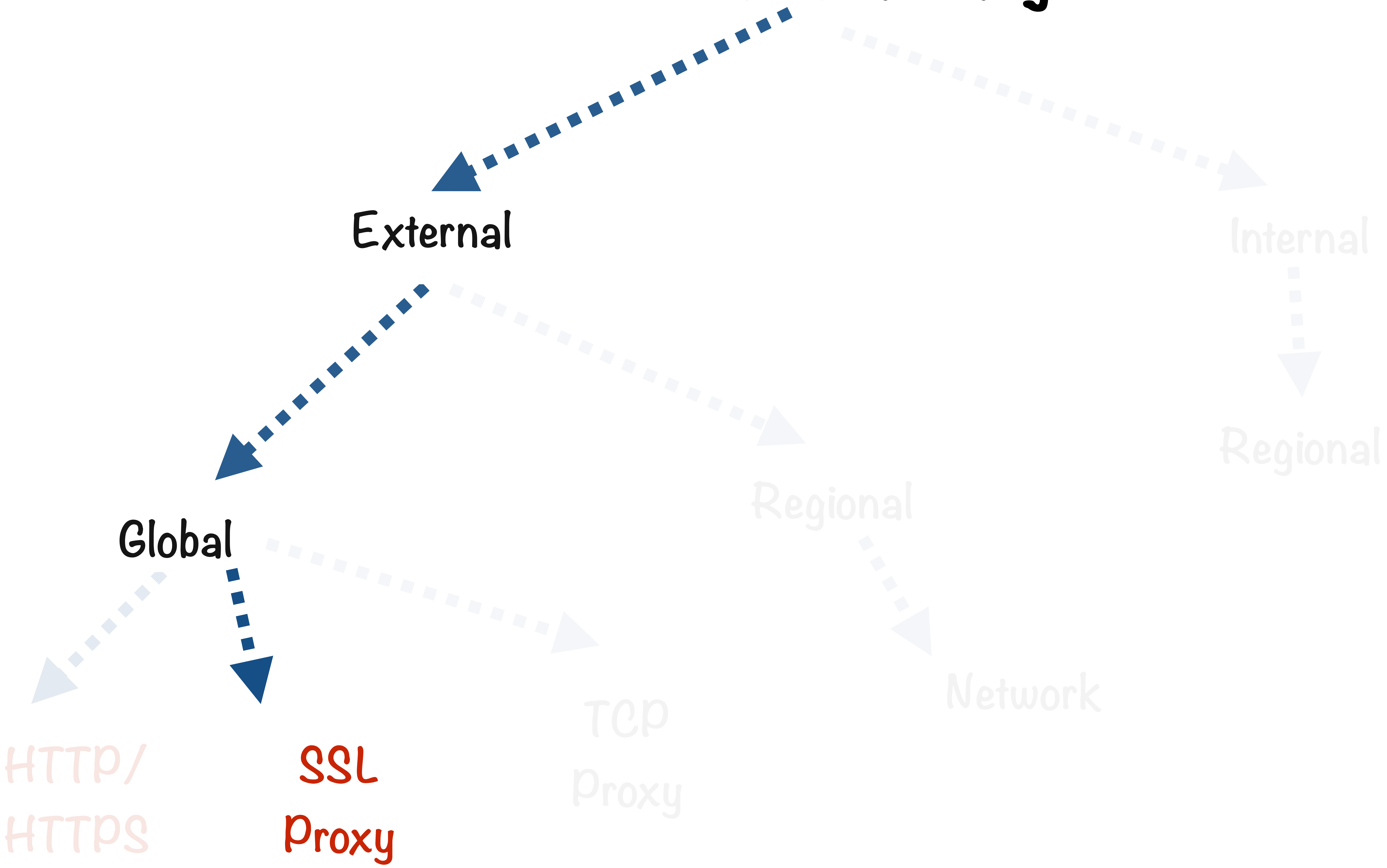
SSL Proxy Load Balancing

The SSL connections are terminated at the global layer and then proxied to the closest available instance group



* You can separately decide if you want SSL between the proxy and your backends or not. We recommend using SSL.

Load Balancing



External

Internal

Global

Regional

Regional

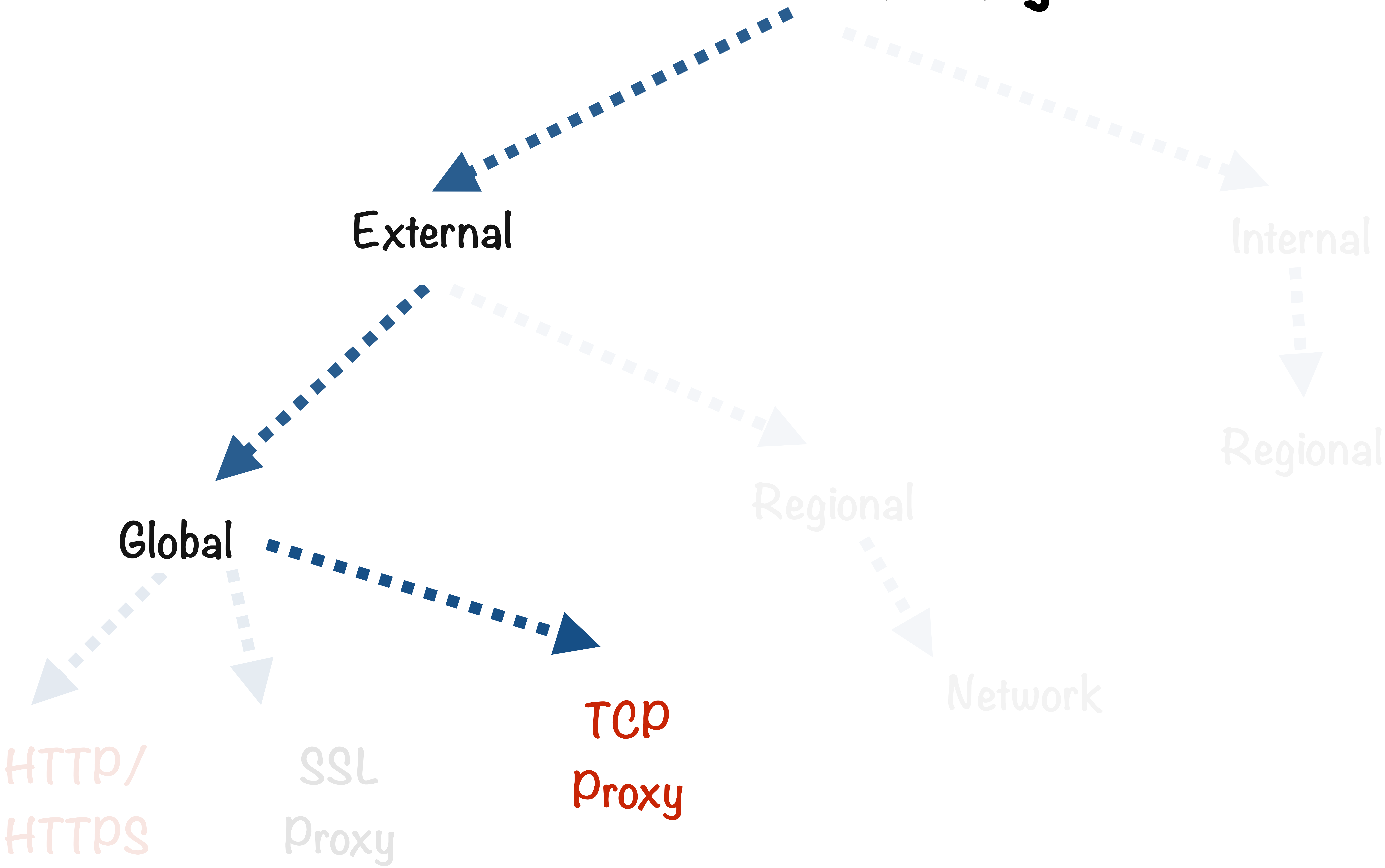
HTTP/
HTTPS

SSL
Proxy

TCP
Proxy

Network

Load Balancing



External

Internal

Global

Regional

Regional

HTTP/ HTTPS

SSL Proxy

TCP Proxy

Network

OSI Network Stack

User	
Application Layer	HTTP/HTTPS
Presentation Layer	
Session Layer	SSL Proxy
Transport Layer	TCP Proxy
Network Layer	Network
Data Link Layer	
Physical Layer	

TCP Proxy Load Balancing

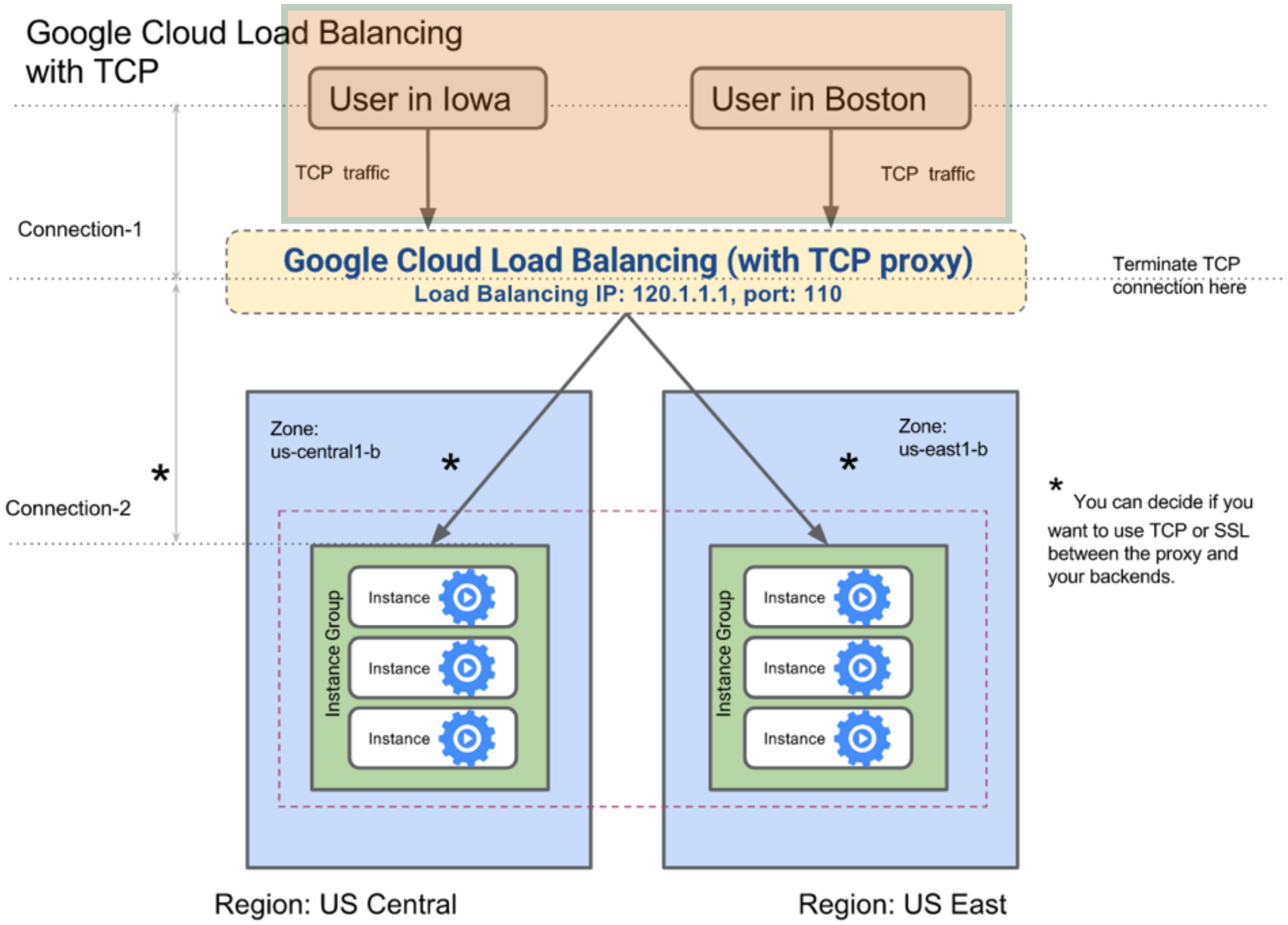
- Perform load balancing based on transport layer (TCP)
- Allows you to use a single IP address for all users around the world.
- Automatically routes traffic to the instances that are closest to the user.

TCP Proxy Load Balancing

- Advantage of transport layer load balancing:
    - more intelligent routing possible than with network layer load balancing
    - better security - TCP vulnerabilities can be patched at the load balancer

TCP Proxy Load Balancing

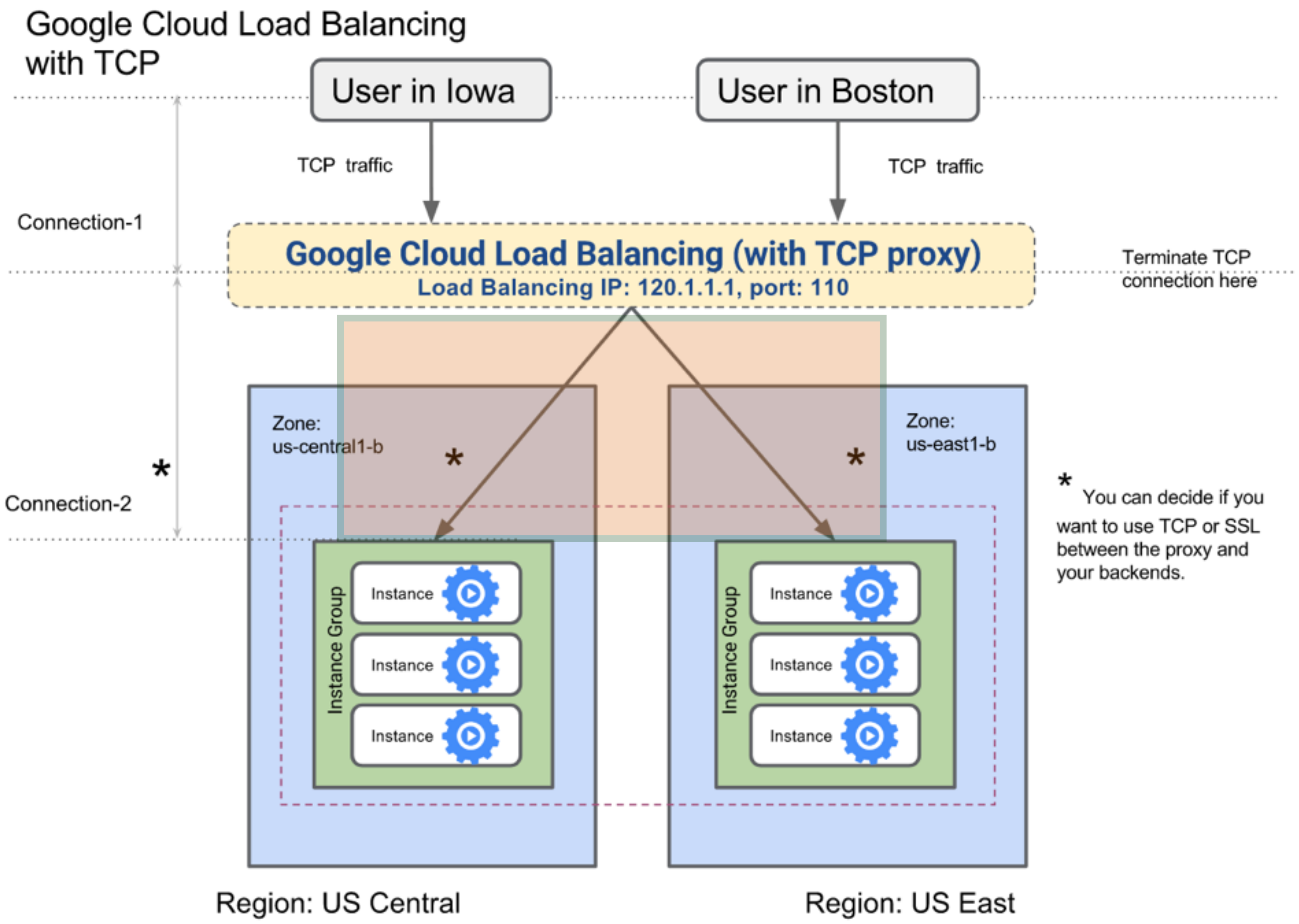
TCP traffic from users goes to the TCP proxy load balancer



* You can decide if you want to use TCP or SSL between the proxy and your backends.

TCP Proxy Load Balancing

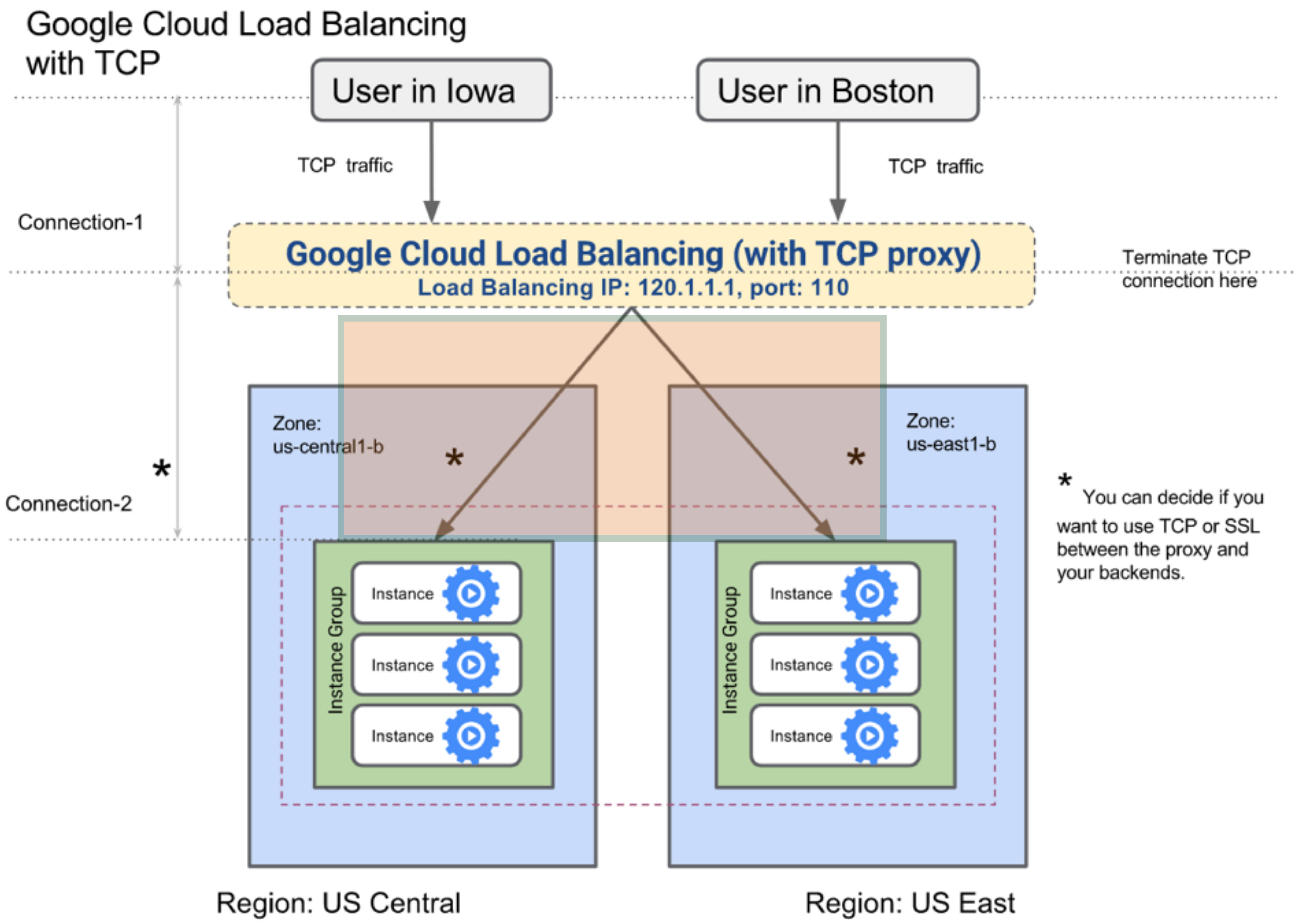
Proxy makes new connections to the backend - these can be TCP connections or even SSL connections



* You can decide if you want to use TCP or SSL between the proxy and your backends.

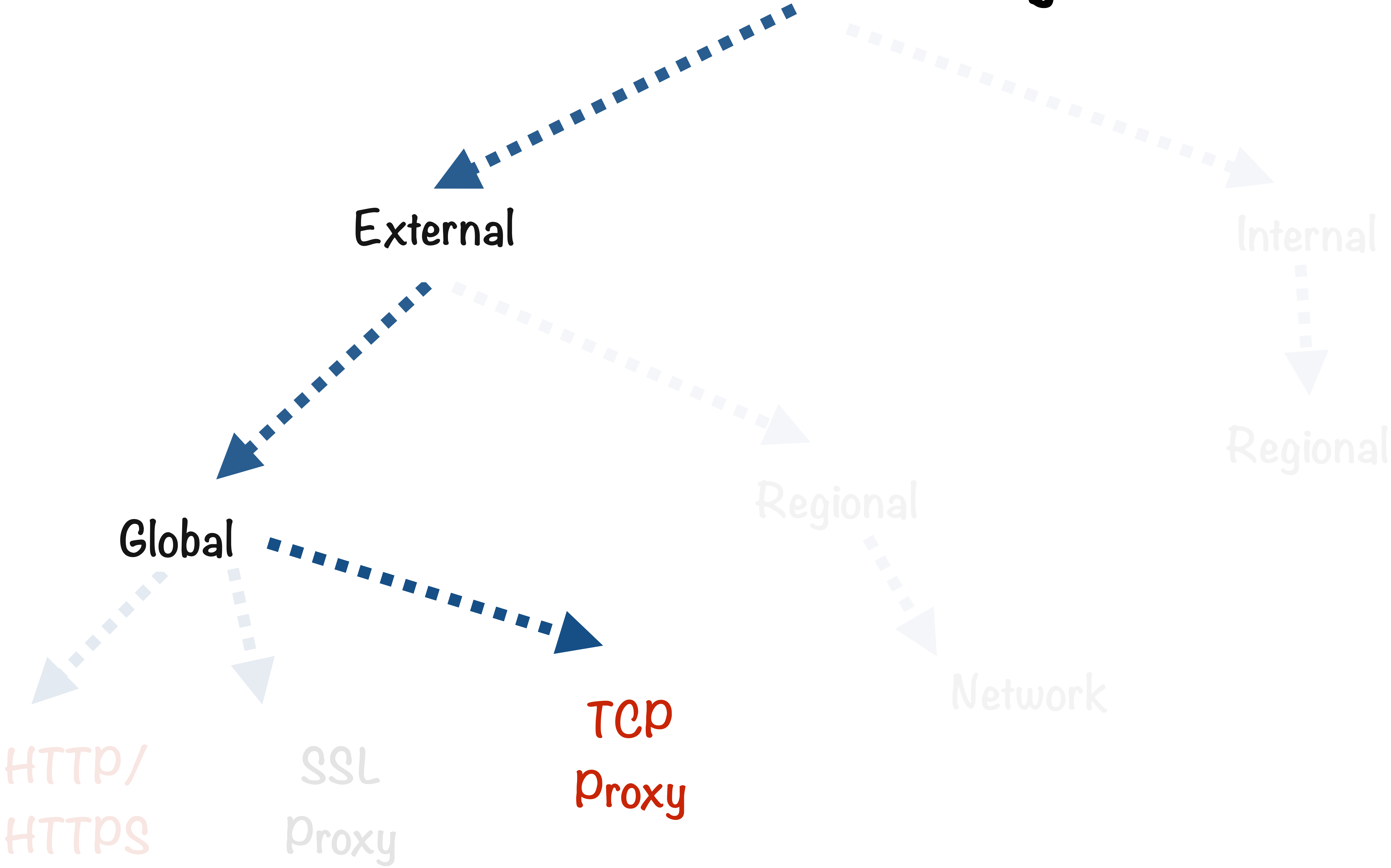
TCP Proxy Load Balancing

The TCP connections are terminated at the global layer and then proxied to the closest available instance group



* You can decide if you want to use TCP or SSL between the proxy and your backends.

Load Balancing



External

Internal

Global

Regional

Regional

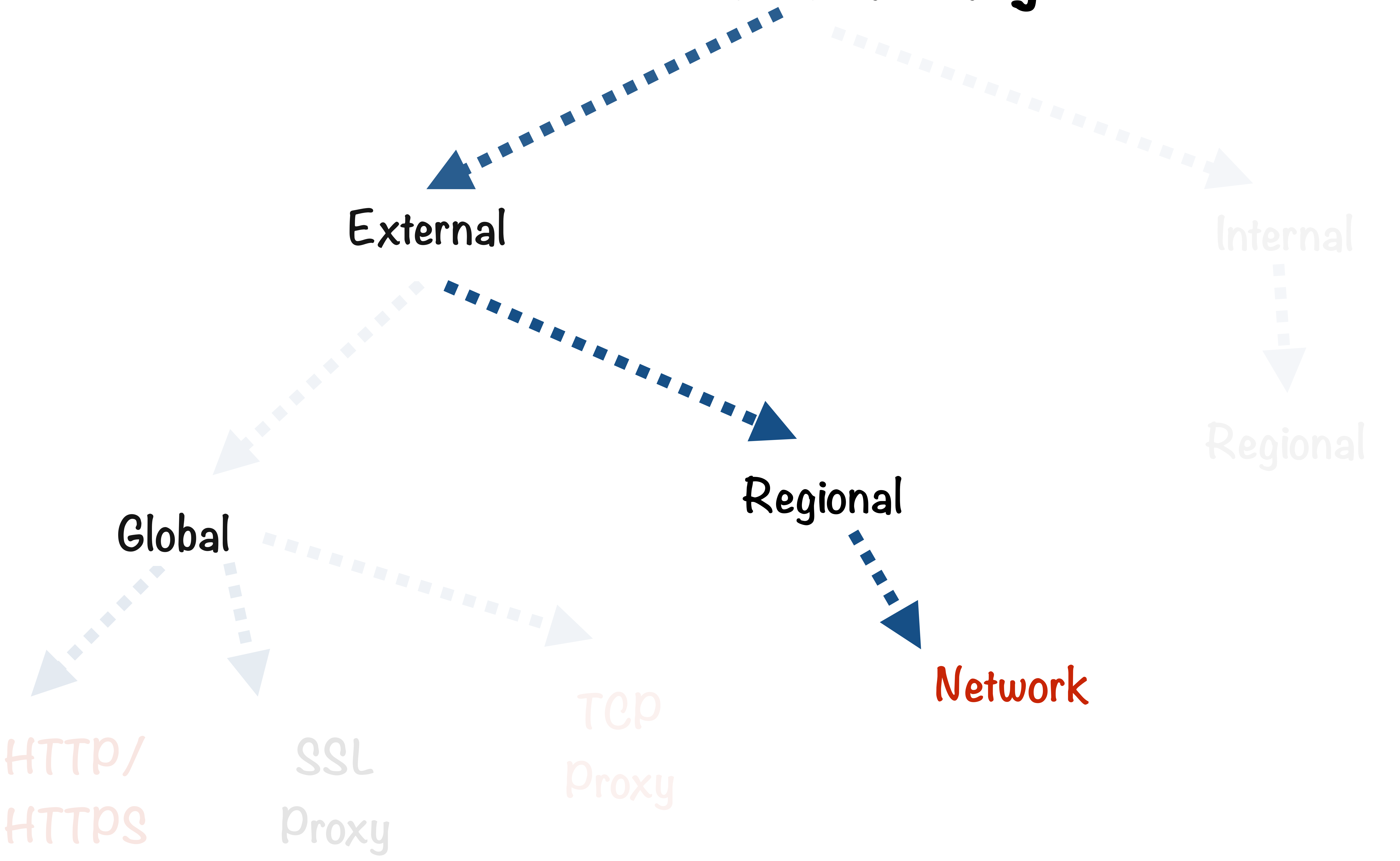
Network

HTTP/
HTTPS

SSL
Proxy

TCP
Proxy

Load Balancing



External

Internal

Global

Regional

Regional

HTTP/
HTTPS

SSL
Proxy

TCP
Proxy

Network

OSI Network Stack

User	
Application Layer	HTTP/HTTPS
Presentation Layer	
Session Layer	SSL Proxy
Transport Layer	TCP Proxy
Network Layer	Network
Data Link Layer	
Physical Layer	

Network Load Balancing

- Based on incoming IP protocol data, such as address, port, and protocol type
- **Pass-through, regional** load balancer - does not proxy connections from clients
- Use it to load balance UDP traffic, and TCP and SSL traffic
- Load balances traffic on ports that are not supported by the SSL proxy and TCP proxy load balancers

Load Balancing Algorithm

- Picks an instance based on a hash of:
  - the source IP and port
  - destination IP and port
  - protocol
- This means that incoming TCP connections are spread across instances and each new connection may go to a different instance.
- Regardless of the session affinity setting, all packets for a connection are directed to the chosen instance until the connection is closed and have no impact on load balancing decisions for new incoming connections
- This can result in imbalance between backends if long-lived TCP connections are in use.

Target Pools

- Network load balancing forwards traffic to target pools
- A **group of instances** which receive incoming traffic from forwarding rules
- Can only be used with forwarding rules for TCP and UDP traffic
- Can have **backup pools** which will receive requests if the first pool is unhealthy
- **failoverRatio** is the ratio of healthy instances to failed instances in a pool
- If primary target pool's ratio is **below the failoverRatio** traffic is sent to the backup pool

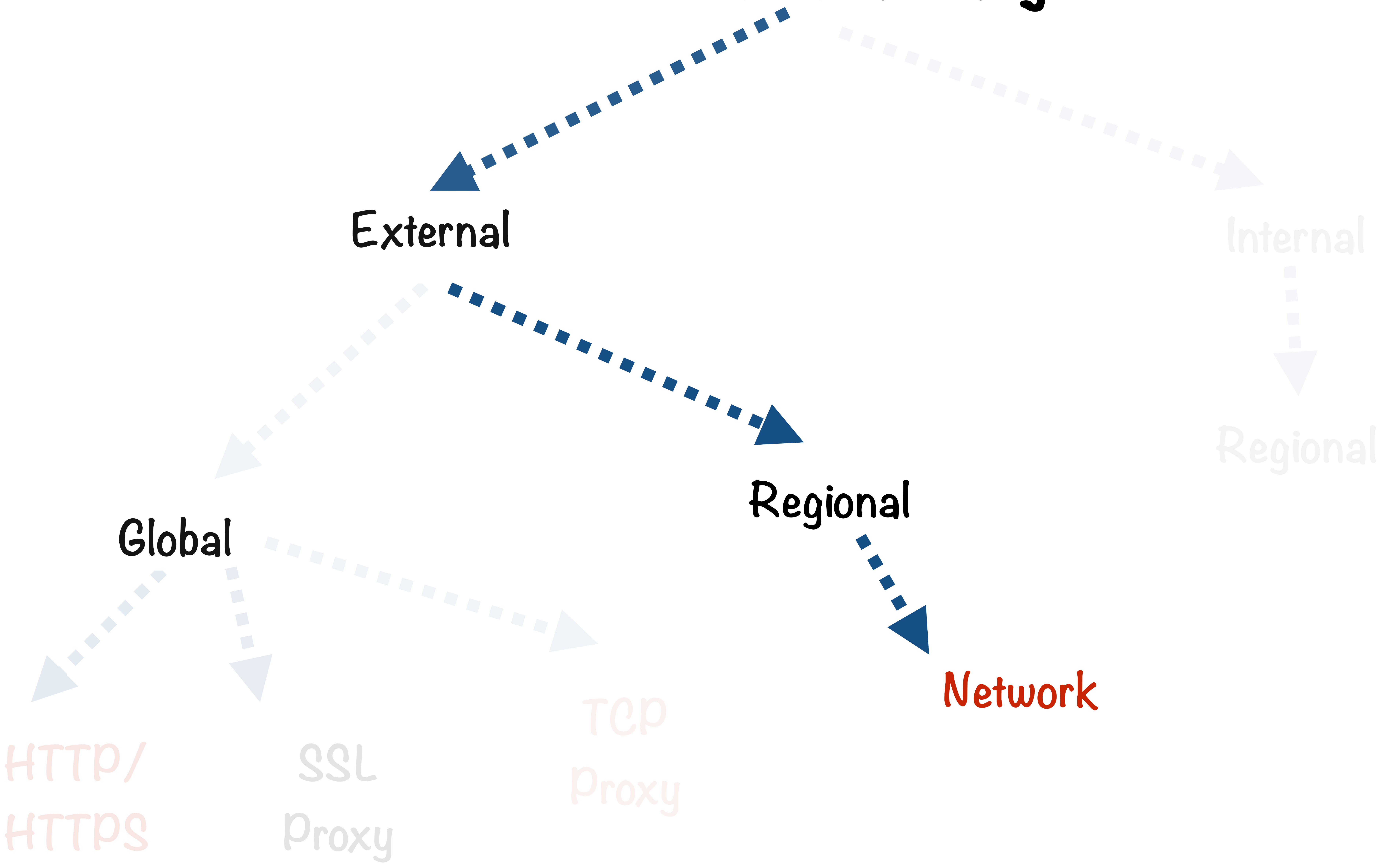
Health Checks

- Configured to check instance health in target pools
- Network load balancing uses legacy health checks for determining instance health

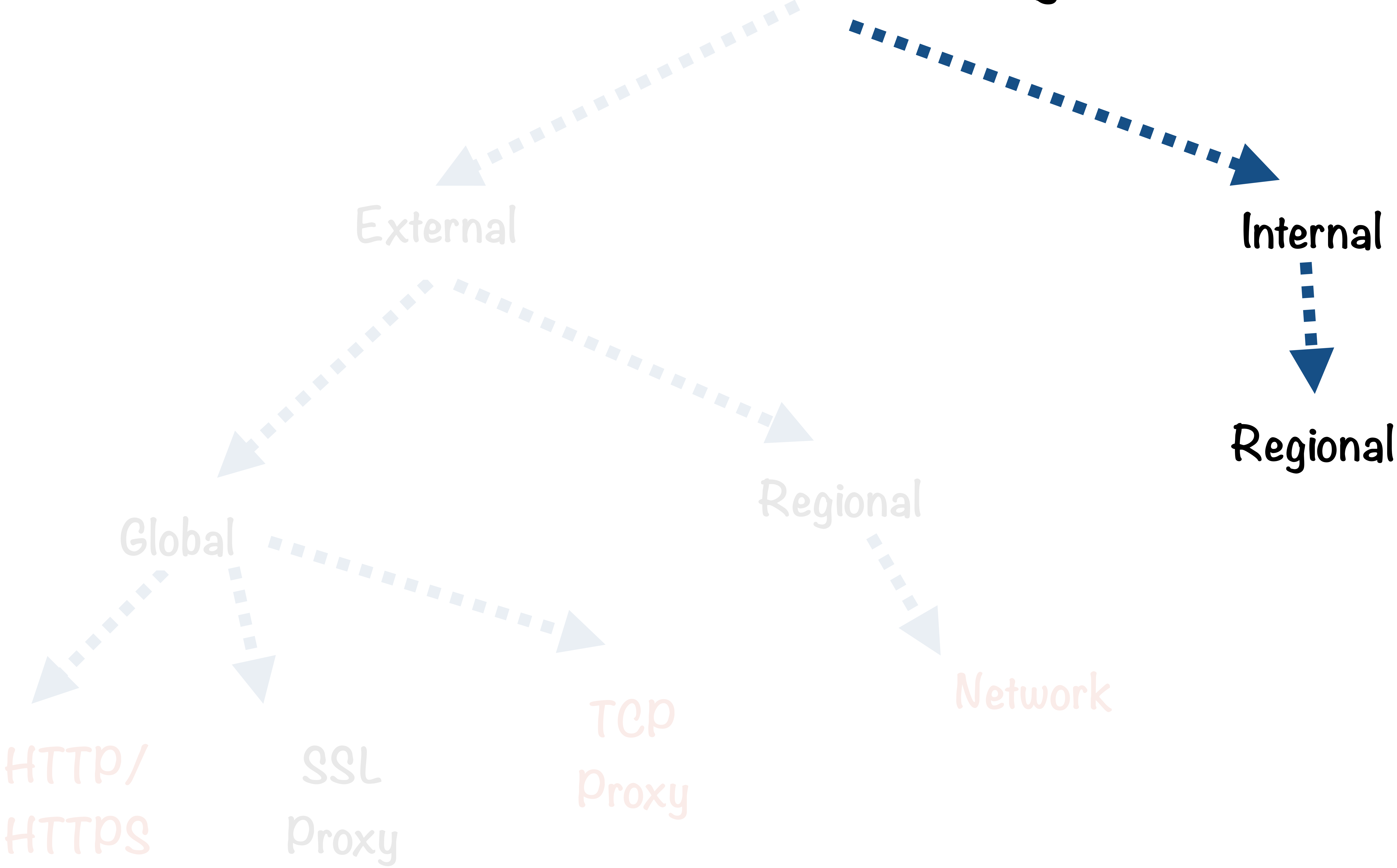
Firewall Rules

- HTTP health check probes are sent from the IP ranges 209.85.152.0/22, 209.85.204.0/22, and 35.191.0.0/16.
- The load balancer uses the same ranges to connect to the instances
- Firewall rules should be configured to allow traffic from these IP ranges

Load Balancing



Load Balancing



External

Internal

Global

Regional

Regional

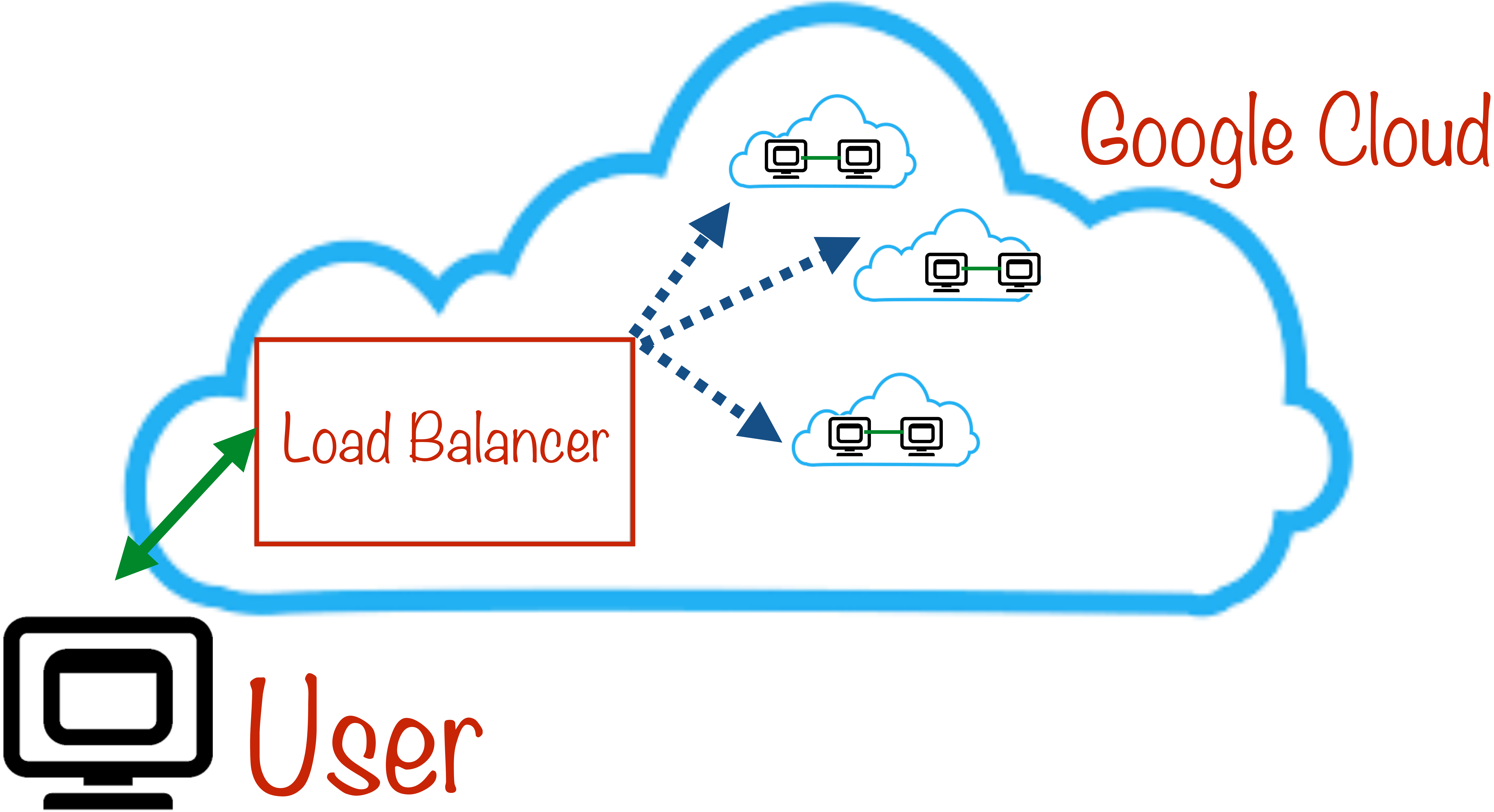
HTTP/ HTTPS

SSL Proxy

TCP Proxy

Network

External Load Balancing

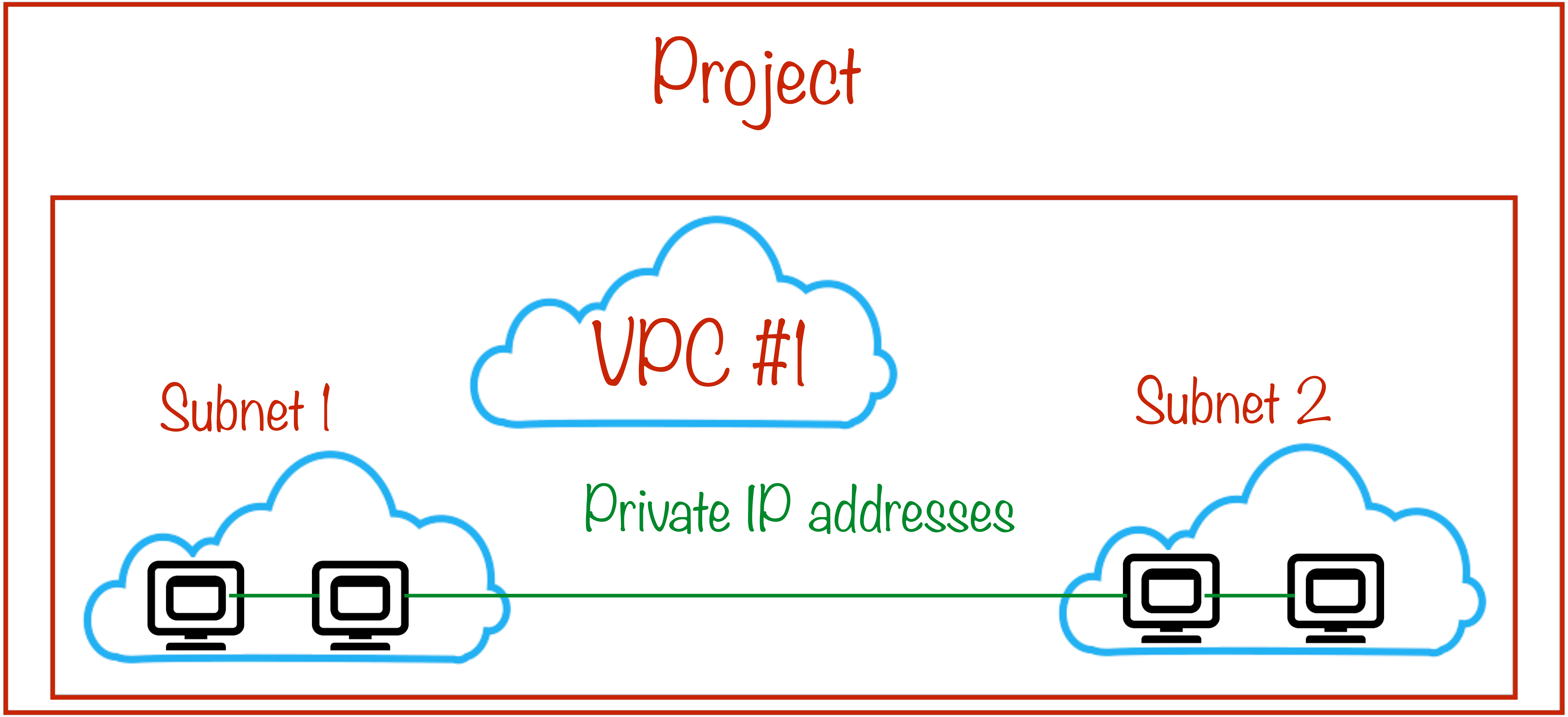


Google Cloud

Load Balancer

User

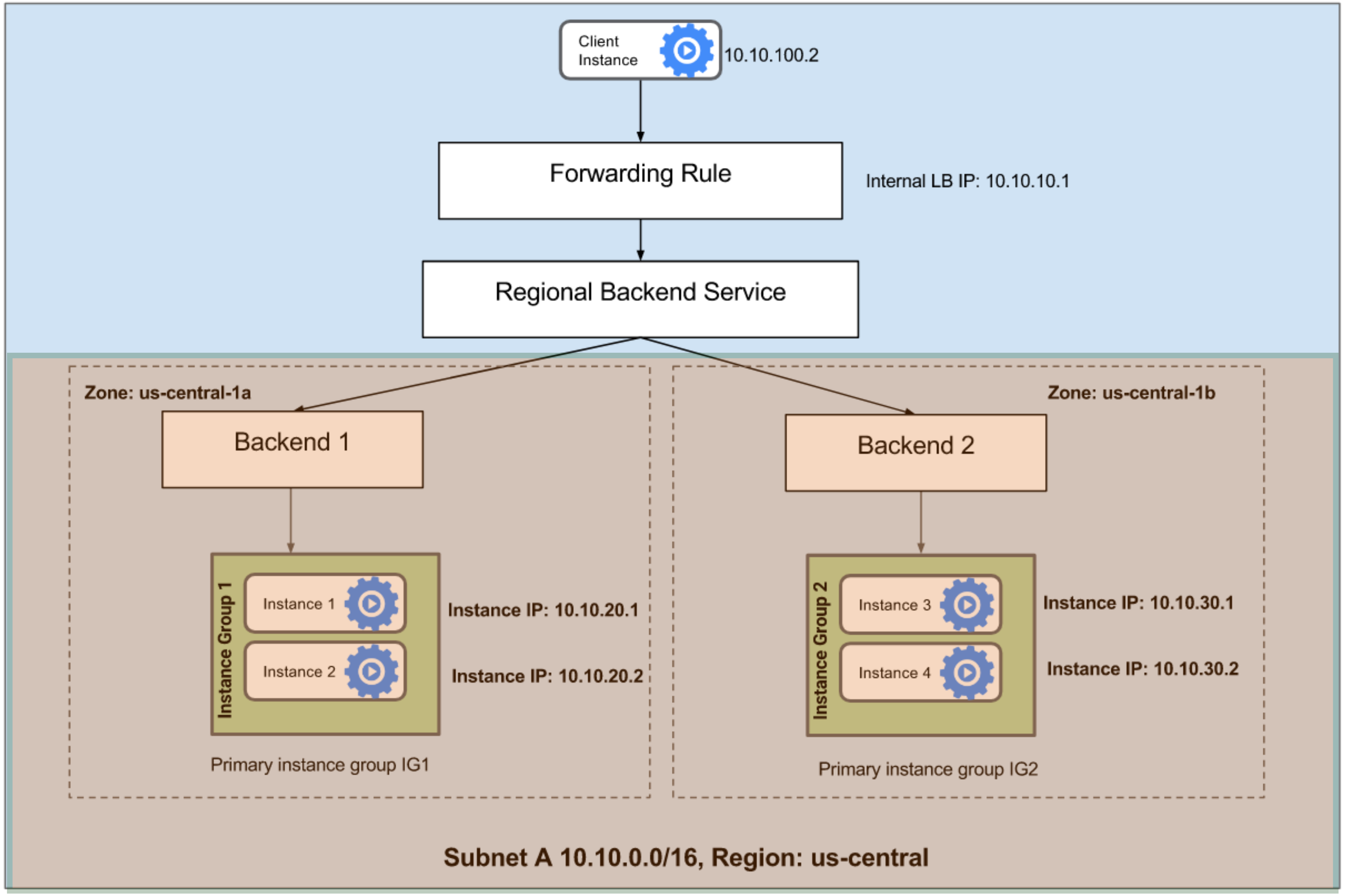
Internal Load Balancing



Internal Load Balancing

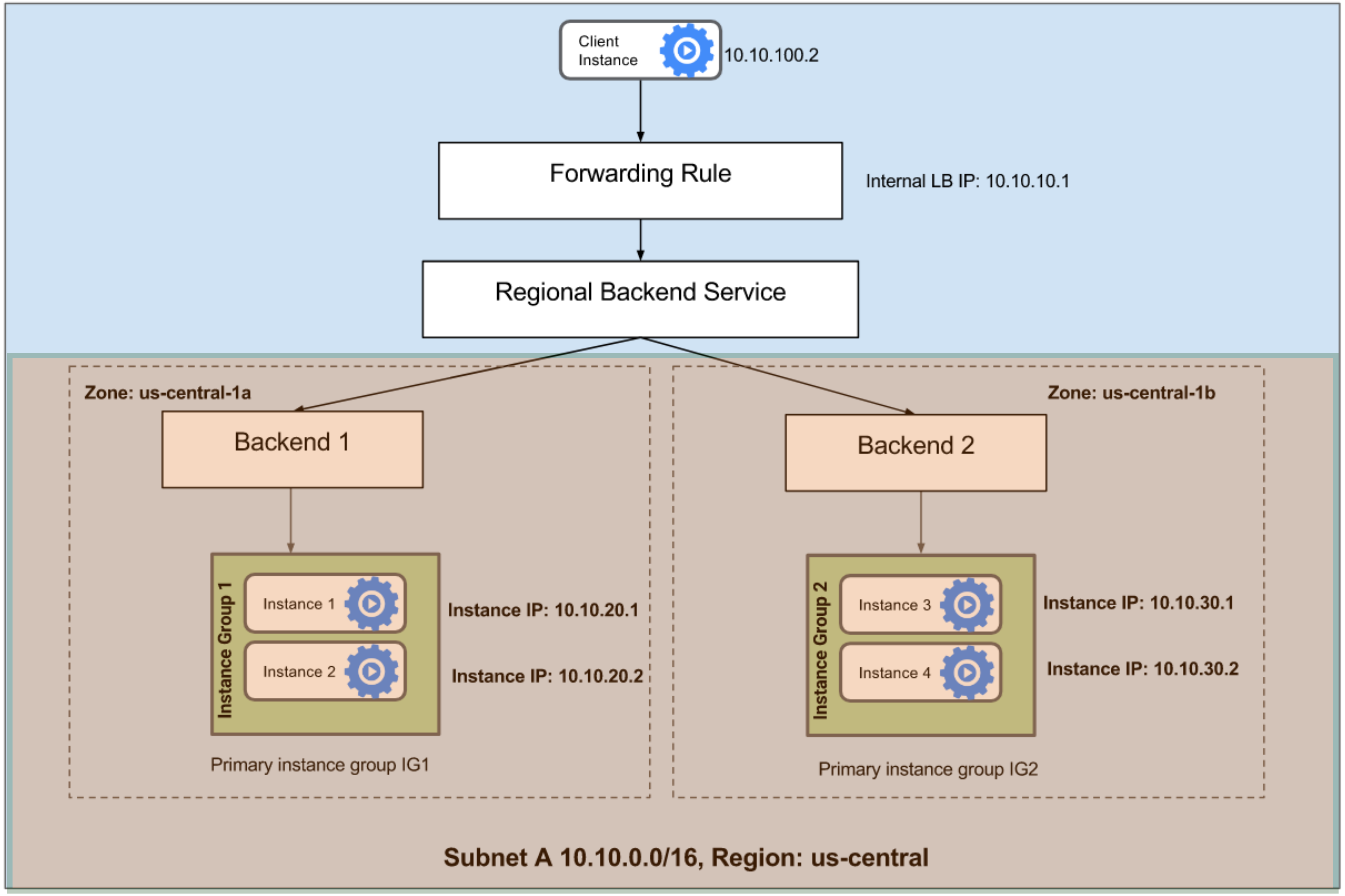
- **Private load balancing IP address** that only your VPC instances can access
- VPC traffic stays **internal** - less latency, more security
- No public IP address needed
- Useful to balance requests from your **frontend instances to your backend instances**

Internal Load Balancing



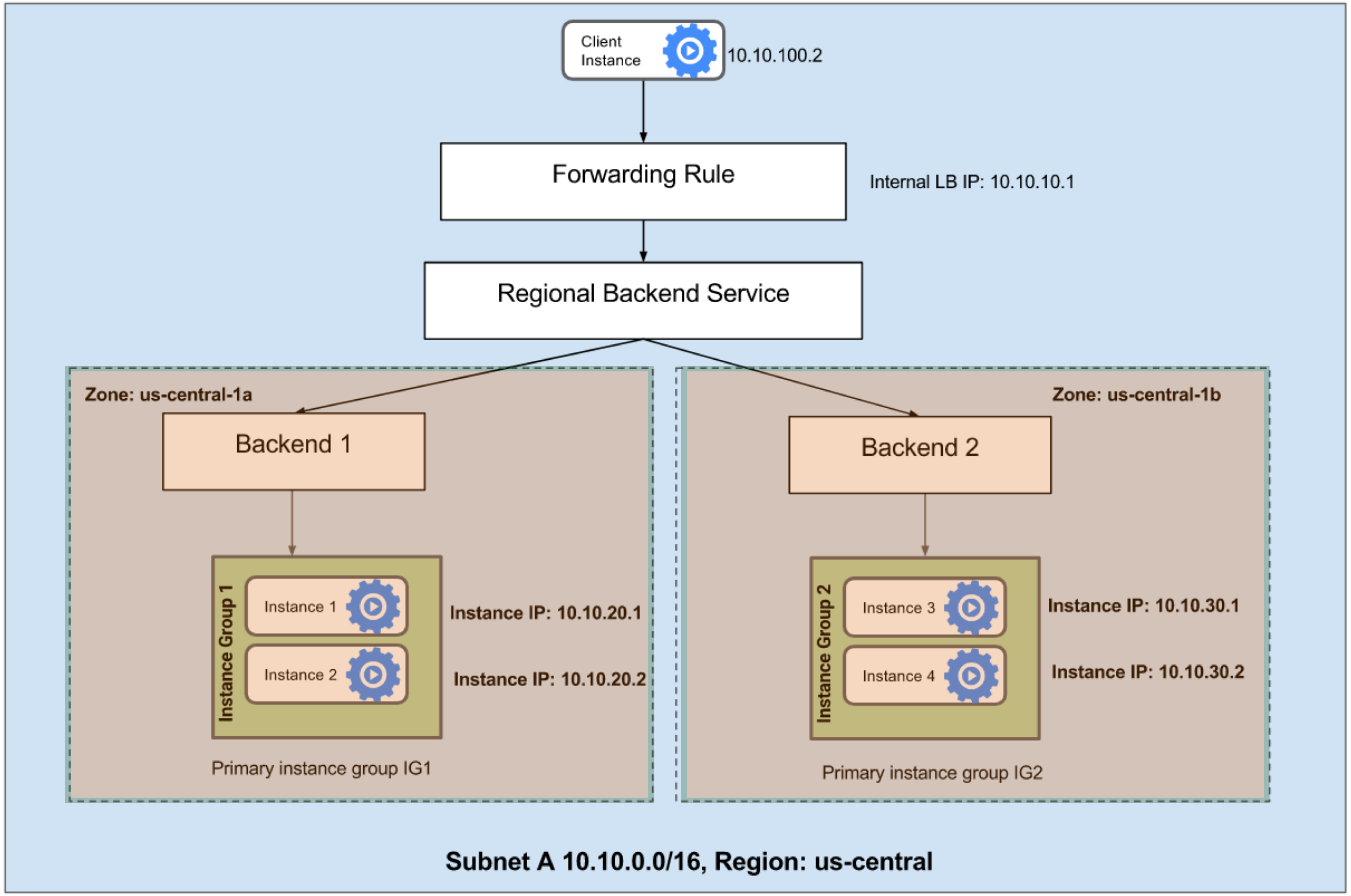
A single subnet in the region us-central

Internal Load Balancing



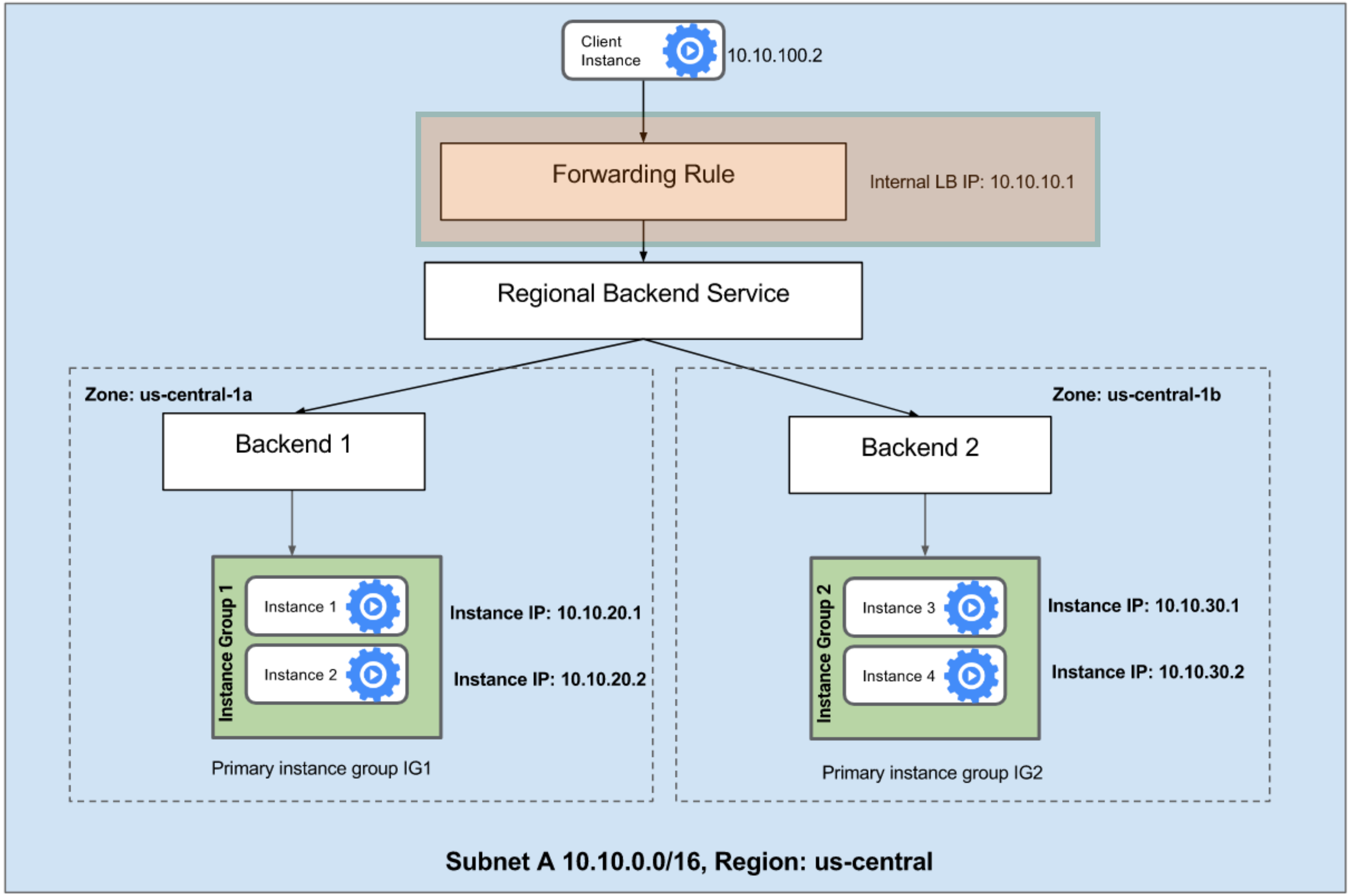
All instances belong to the same VPC and region but can be in different subnets

Internal Load Balancing



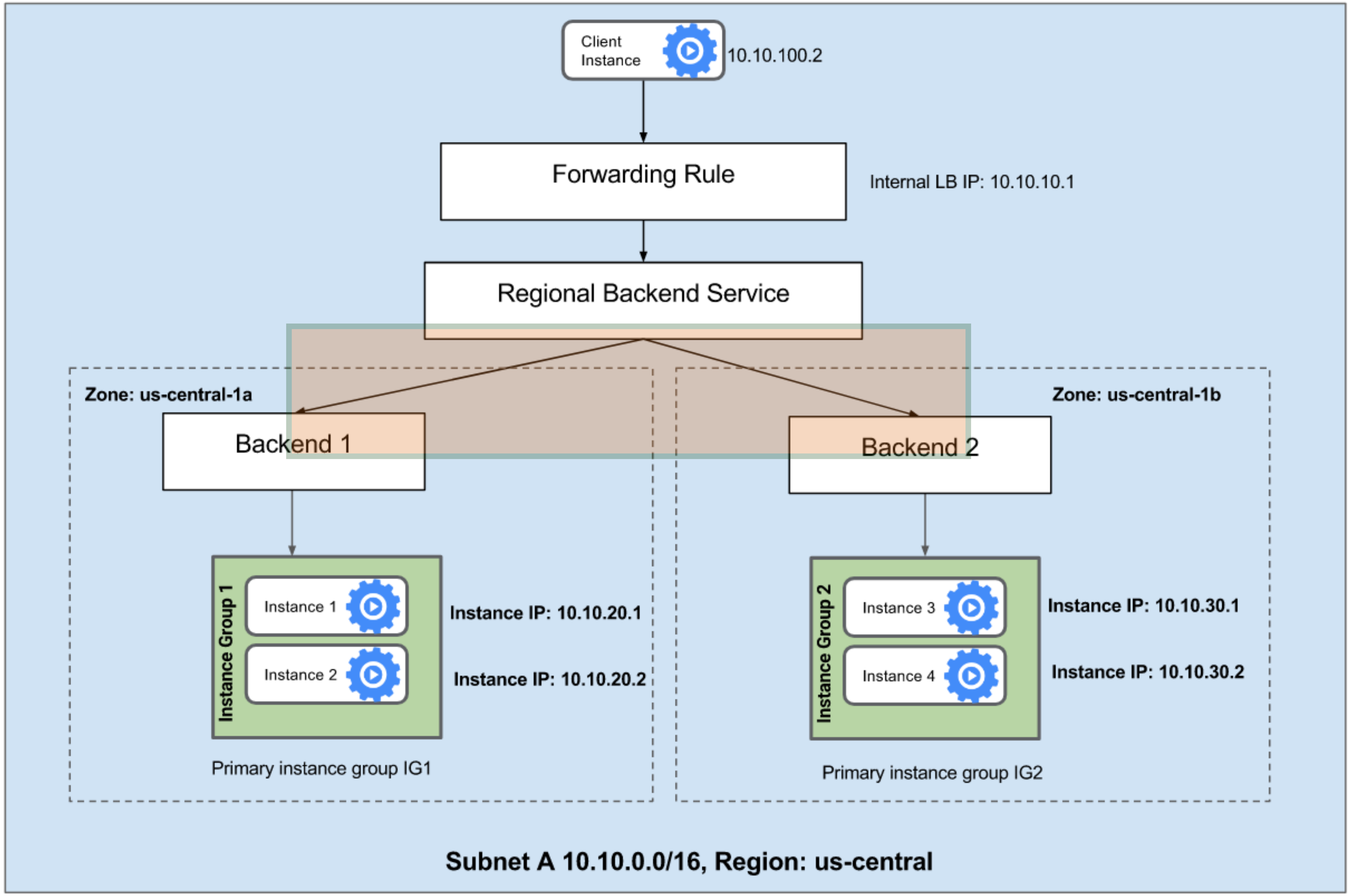
2 backend instance groups across two zones

Internal Load Balancing



The load balancing IP is from the same VPC network

Internal Load Balancing



The request gets forwarded to one of the two instance groups with the subnet

Load Balancing Algorithm

- The backend instance for a client is selected using a hashing algorithm that takes instance health into consideration.

- Using a 5-tuple hash, five parameters for hashing:

    - client source IP
    - client port
    - destination IP (the load balancing IP)
    - destination port
    - protocol (either TCP or UDP)

Load Balancing Algorithm

- Introduce session affinity by hashing on only some of the 5 parameters
    - Hash based on 3-tuple (Client IP, Dest IP, Protocol)
    - Hash based on 2-tuple (Client IP, Dest IP)

Health Checks

- **HTTP, HTTPS health checks:** These provide the highest fidelity, they verify that the web server is up and serving traffic, not just that the instance is healthy.

- **SSL (TLS) health checks:** Configure the SSL health checks if your traffic is not HTTPS but is encrypted via SSL(TLS)

- **TCP health checks:** For all TCP traffic that is not HTTP(S) or SSL(TLS), you can configure a TCP health check

High Availability

- Managed service. **no additional configuration** needed to ensure high availability
- Can configure **multiple instance groups in different zones** to guard against failures in a single zone
- With multiple instance groups all instances are treated as if they are in a **single pool** and the load balancer distributes traffic amongst them using the load balancing algorithm

Traditional (Proxy) Internal Load Balancing

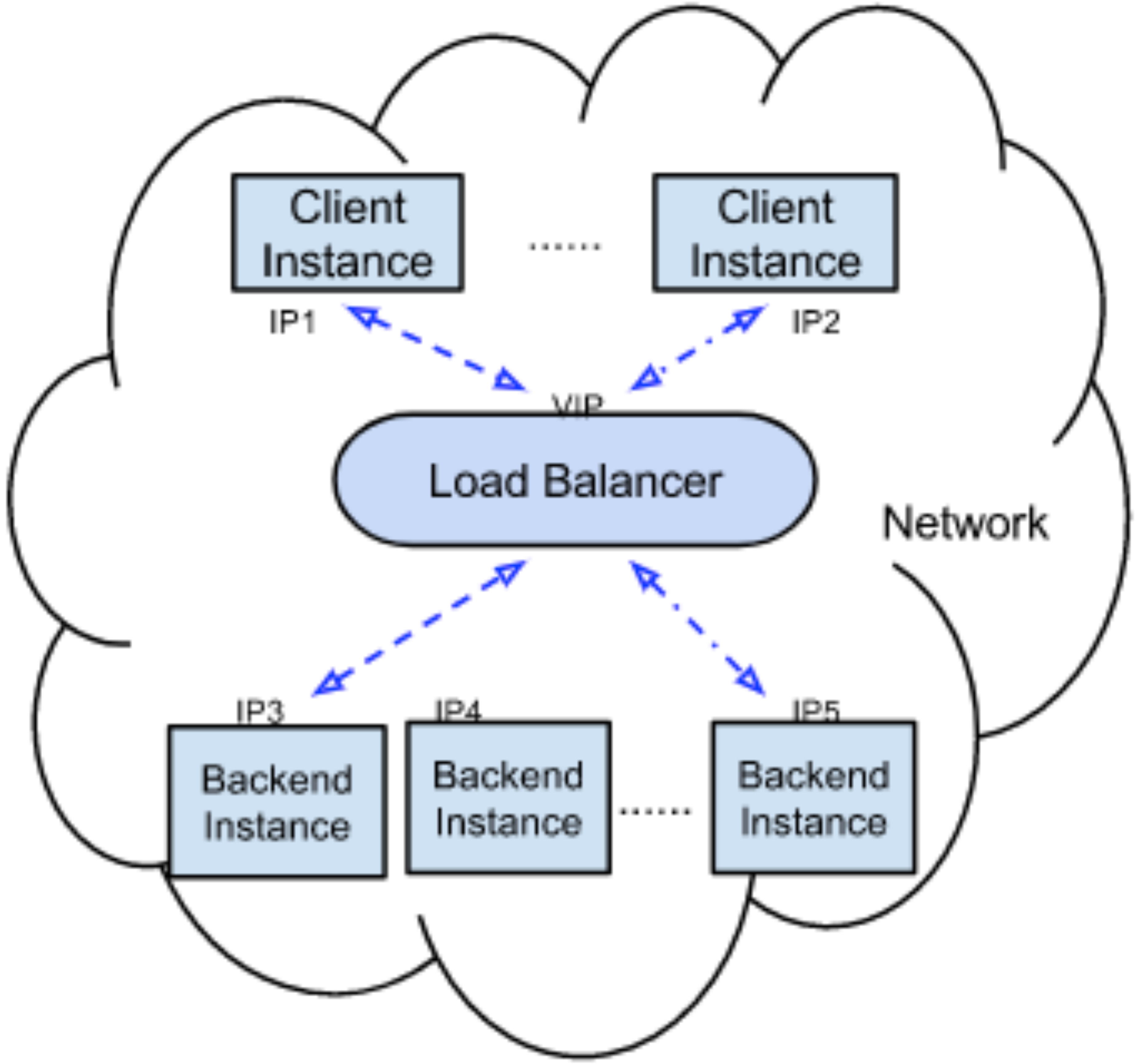
- Configure an internal IP on a load balancing device or instance(s) and your client instance connects to this IP

- Traffic coming to the IP is terminated at the load balancer

- The load balancer selects a backend and establishes a new connection to it

- In effect, there are two connections: Client<->Load Balancer and Load Balancer<->Backend.

Traditional (Proxy) Internal Load Balancing

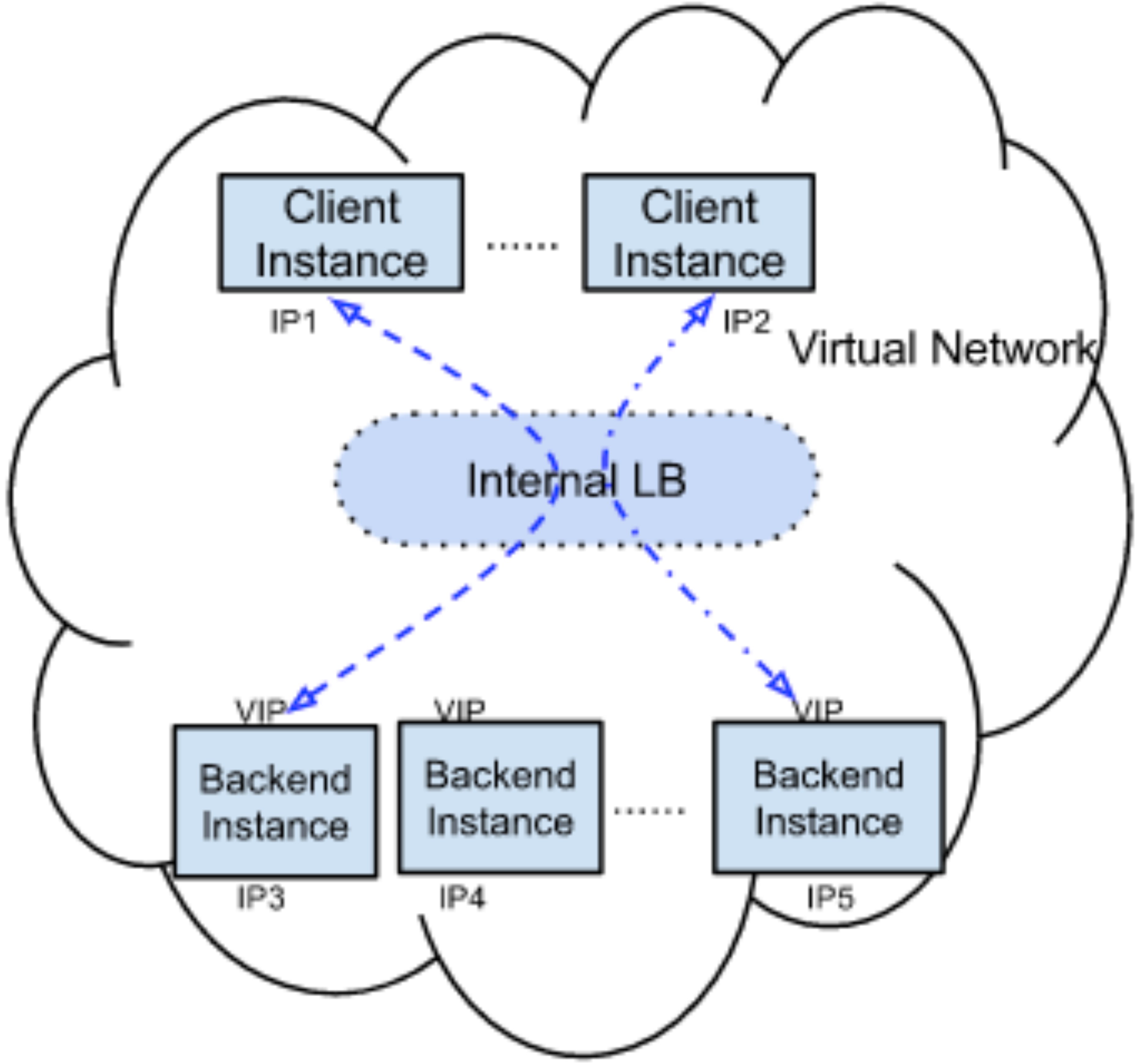


1.Proxy Internal
Load Balancing

GCP Internal Load Balancing

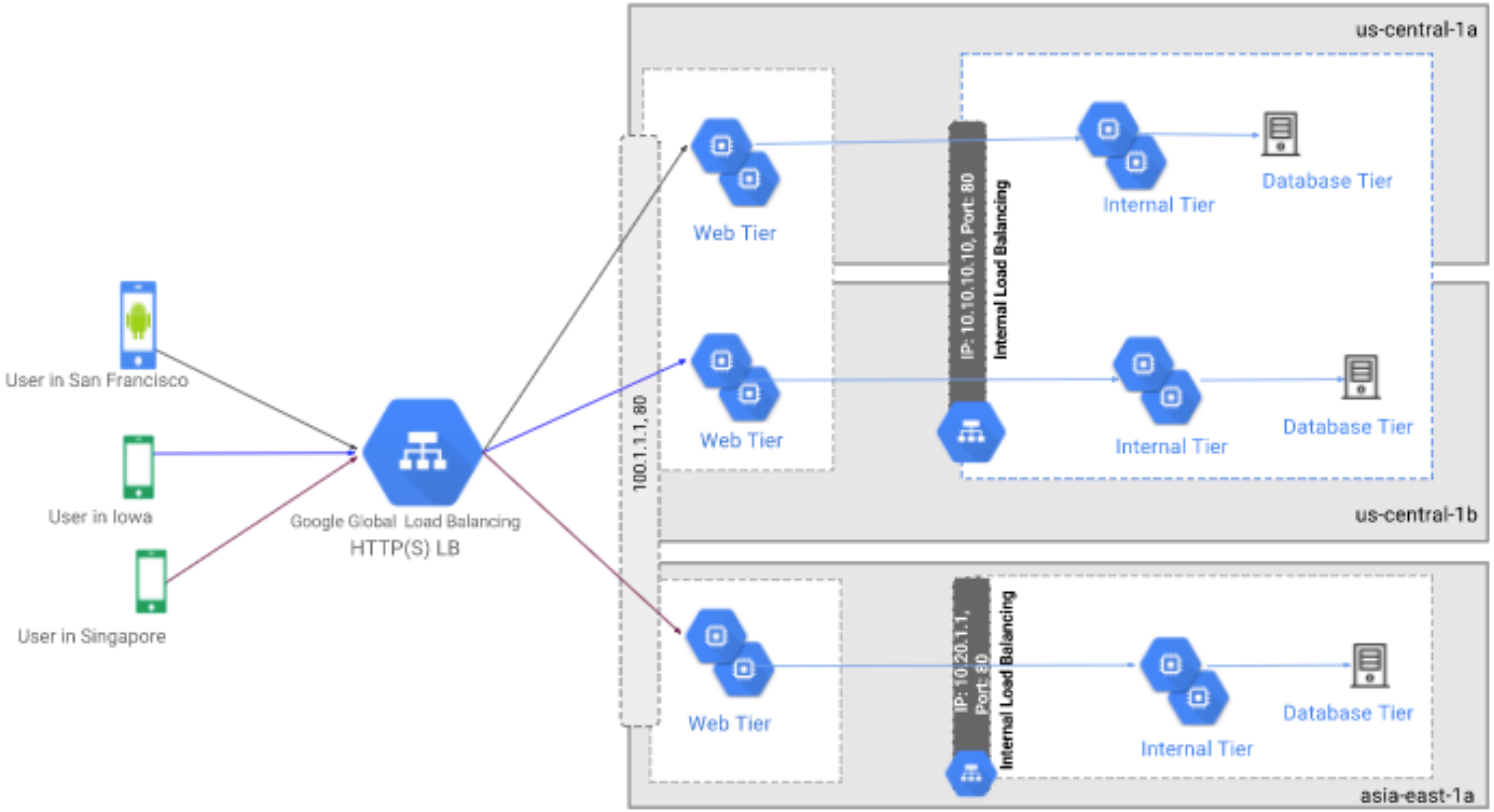
- Not proxied - differs from traditional model
- lightweight load-balancing built on top of Andromeda network virtualization stack
- provides software-defined load balancing that directly delivers the traffic from the client instance to a backend instance

GCP Internal Load Balancing

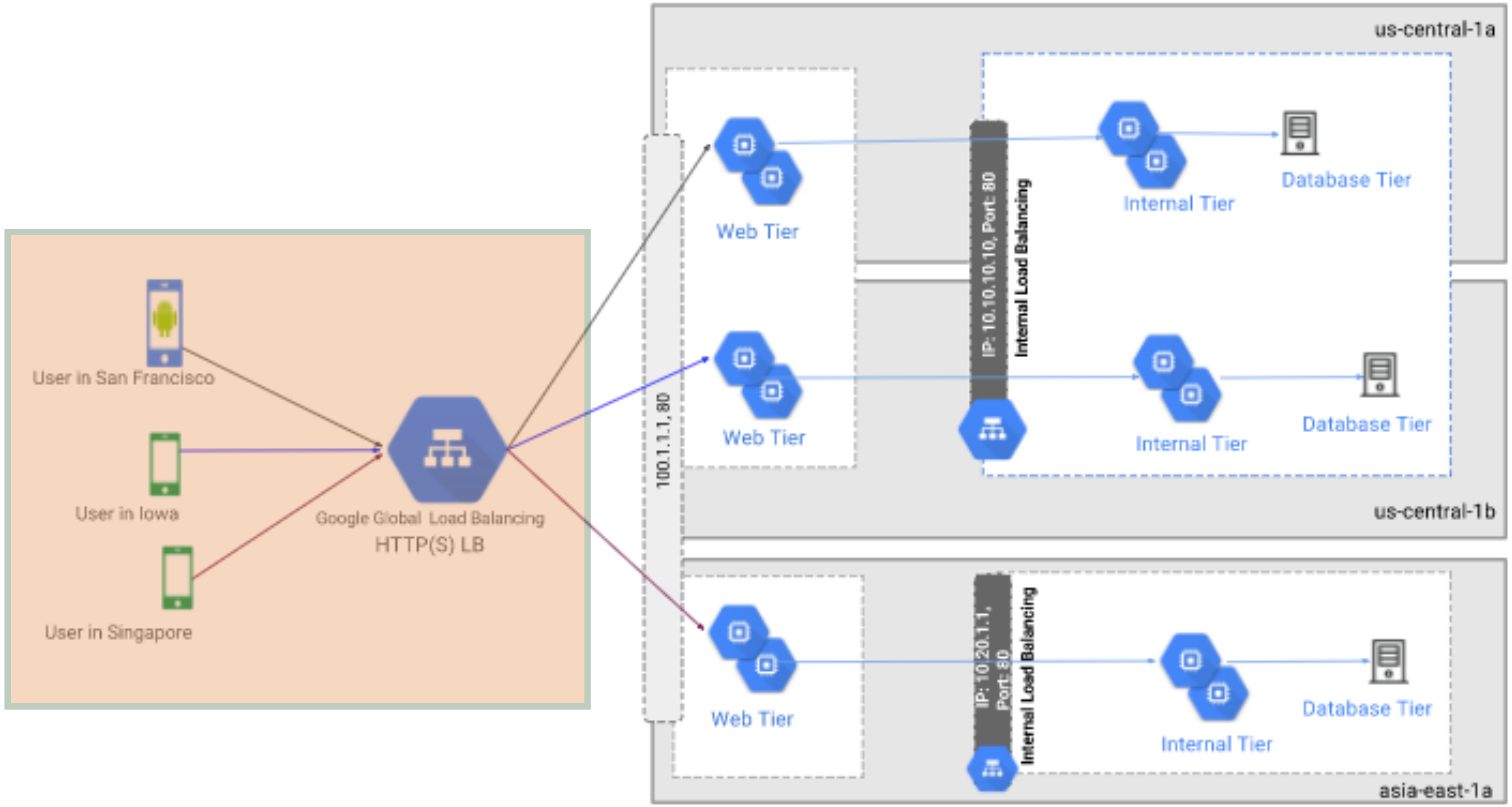


2. Google Cloud Internal Load Balancing

Use Case: 3-tier Web App

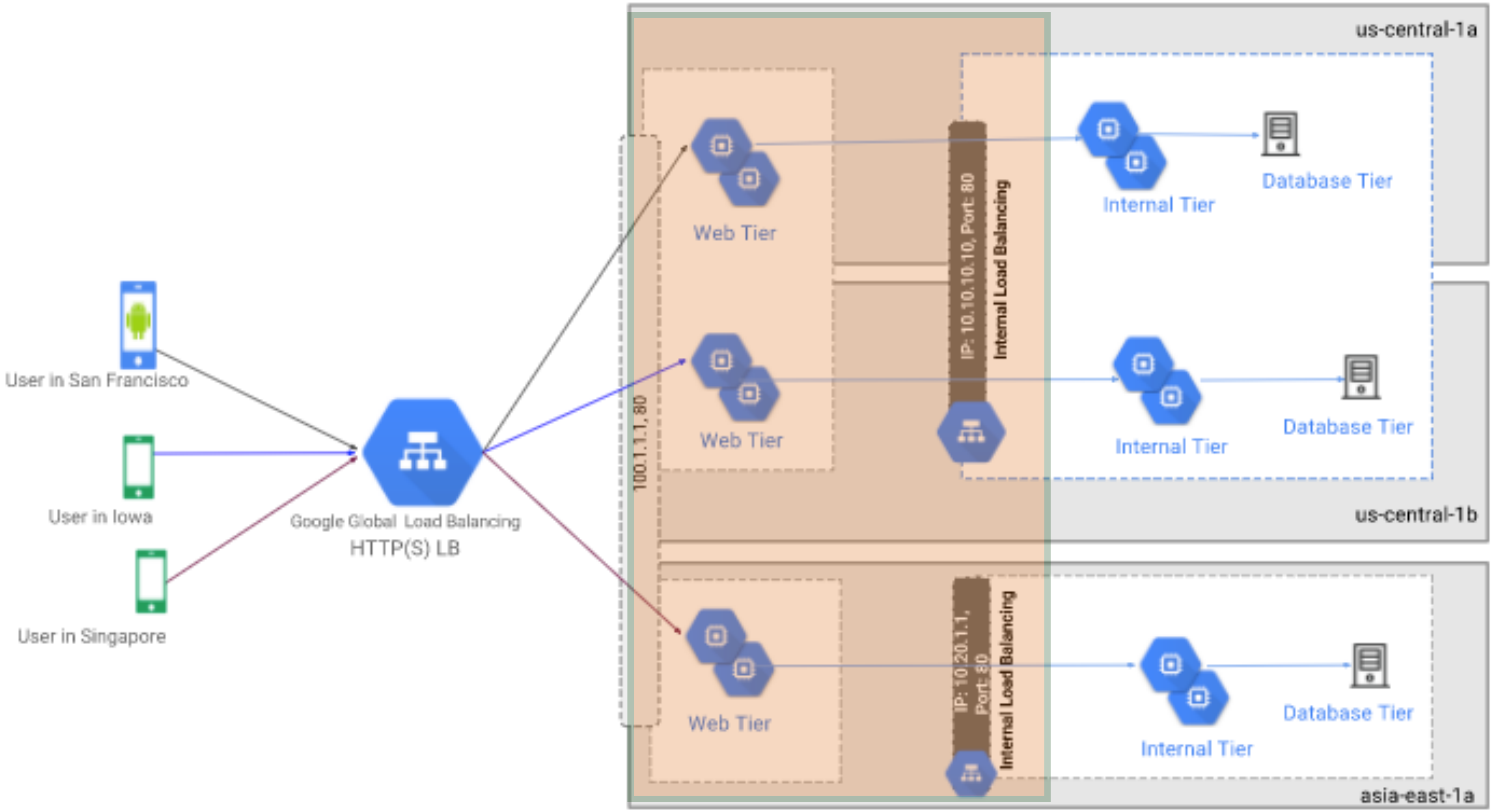


Use Case: 3-tier Web App



External HTTP(S) load balancer to manage client traffic to frontend instance groups

Use Case: 3-tier Web App



Frontend instances are connected to the backend instances using an **internal load balancer**

Autoscaling

Autoscaling

- **Managed instance groups** automatically add or remove instances based on increases and decreases in load
- Helps your applications **gracefully handle increases in traffic**
- **Reduces cost** when load is lower
- Define autoscaling policy, the autoscaler takes care of the rest

Autoscaling is a feature of managed instance groups

Unmanaged instance groups are not supported

Autoscaling is a feature of managed instance groups

For GKE groups autoscaling is different, called Cluster Autoscaling

Autoscaling

- Autoscaling Policy
- Target Utilization Level

Autoscaling Policy

Average CPU utilization

Stackdriver monitoring metrics

HTTP(S) load balancing server capacity (utilization or RPS)

Pub/Sub queueing workload (alpha)

Target Utilization Level

- The level at which you want to maintain your VMs
- Interpreted differently based on the autoscaling policy that you've chosen

Autoscaling Policy

Average CPU utilization

Stackdriver monitoring metrics

HTTP(S) load balancing server capacity (utilization or RPS)

Pub/Sub queueing workload (alpha)

Average CPU Utilization

- Target utilization level of 0.75 maintains average CPU utilization at 75% **across all instances**
- If utilization exceed the target, **more CPUs will be added**
- If utilization reaches 100% during times of heavy usage the autoscaler might increase the number of CPUs by
  - 50%
  - 4 instances
- whichever is **larger**

Autoscaling Policy

Average CPU utilization

Stackdriver monitoring metrics

HTTP(S) load balancing server capacity (utilization or RPS)

Pub/Sub queueing workload (alpha)

Stackdriver monitoring metrics

- Can configure the autoscaler to use **standard** or **custom** metrics
- Not all standard metrics are valid utilization metrics that the autoscaler can use
  - the metric must contain data for a VM instance
  - the metric must define **how busy the resource is**, the metric value increases or decreases proportional to the number of instances in the group

Autoscaling Policy

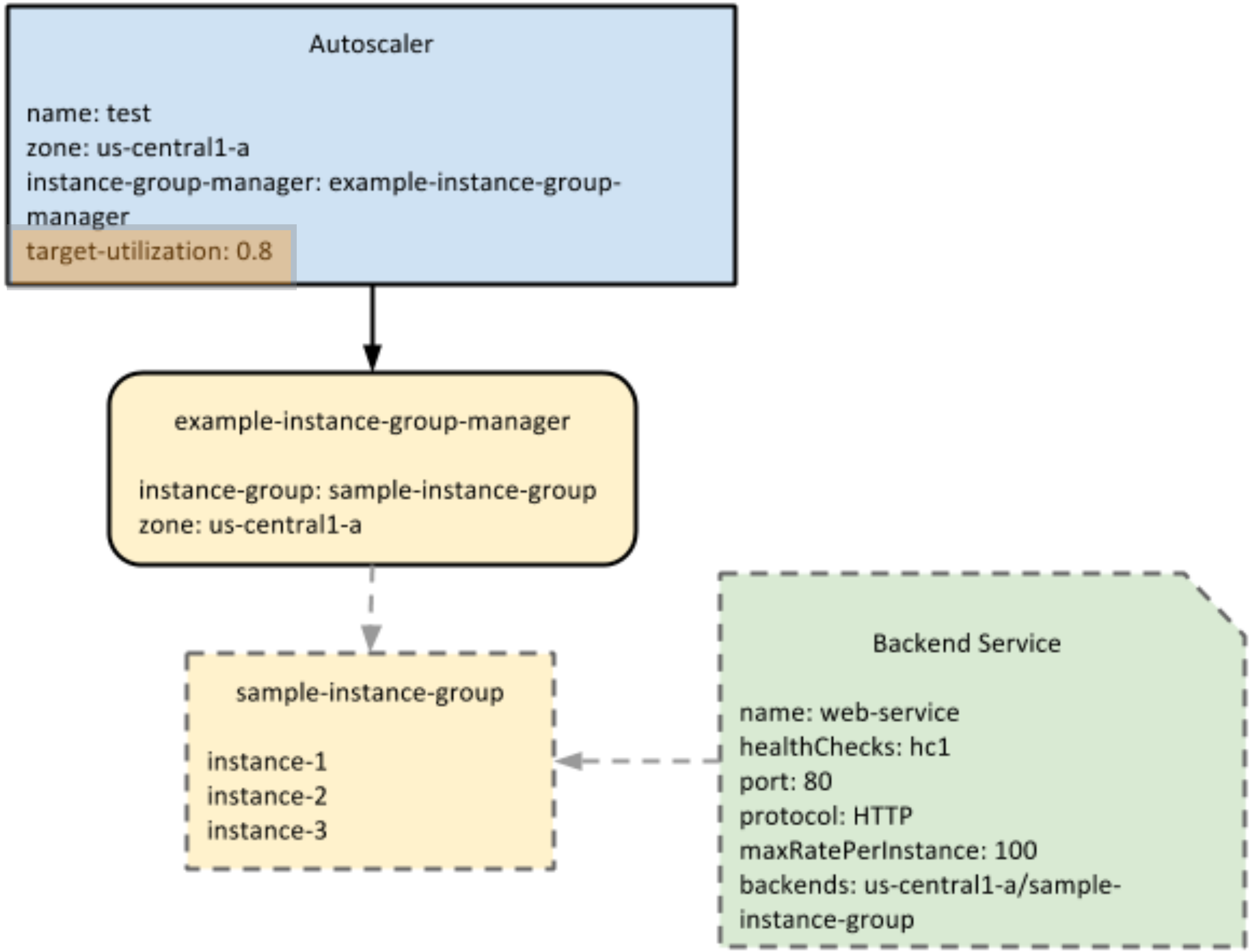
Average CPU utilization

Stackdriver monitoring metrics

HTTP(S) load balancing server capacity (utilization or RPS)

Pub/Sub queueing workload (alpha)

HTTP(S) Load Balancing Server Capacity



Autoscaler

name: test
zone: us-central1-a
instance-group-manager: example-instance-group-manager
target-utilization: 0.8

example-instance-group-manager

instance-group: sample-instance-group
zone: us-central1-a

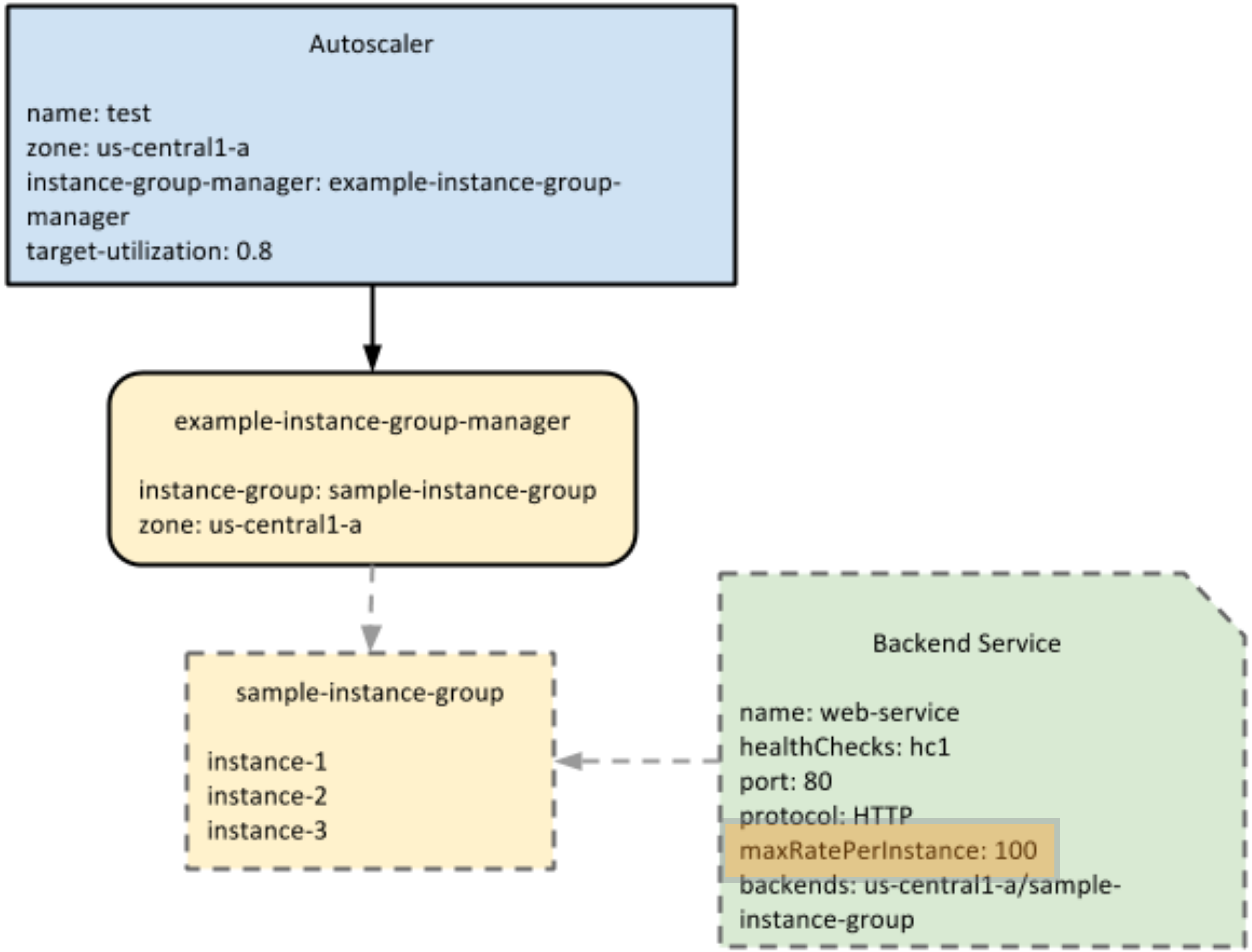
sample-instance-group

instance-1
instance-2
instance-3

Backend Service

name: web-service
healthChecks: hc1
port: 80
protocol: HTTP
maxRatePerInstance: 100
backends: us-central1-a/sample-instance-group

HTTP(S) Load Balancing Server Capacity



Autoscaler

name: test
zone: us-central1-a
instance-group-manager: example-instance-group-manager
target-utilization: 0.8

example-instance-group-manager

instance-group: sample-instance-group
zone: us-central1-a

sample-instance-group

instance-1
instance-2
instance-3

Backend Service

name: web-service
healthChecks: hc1
port: 80
protocol: HTTP
maxRatePerInstance: 100
backends: us-central1-a/sample-instance-group

HTTP(S) Load Balancing Server Capacity

- Only works with
  - **CPU utilization**
  - **maximum requests per second/instance**
- These are the only settings that can be controlled by adding and removing instances

Autoscaling does not work with maximum requests per group

This setting is **independent** of the number of instances in a group

Autoscaler with Multiple Policies

The autoscaler will scale based on the policy which provides the **largest number of VMs** in the group

This ensures that you always have enough machines to handle your workload

Can handle a maximum of **5 policies** at a time

Example Policies

cpuUtilization with target of 0.8

loadBalancingUtilization with target of 0.6

customMetricUtilization for metric1 with target of 1000

customMetricUtilization for metric2 with target of 2000

Example Utilization

cpuUtilization 0.5

loadBalancingUtilization 0.4

customMetricUtilization 1100

customMetricUtilization 2700

Additional Machines Recommended

cpuUtilization 7

loadBalancingUtilization 7

customMetricUtilization 11

customMetricUtilization 14