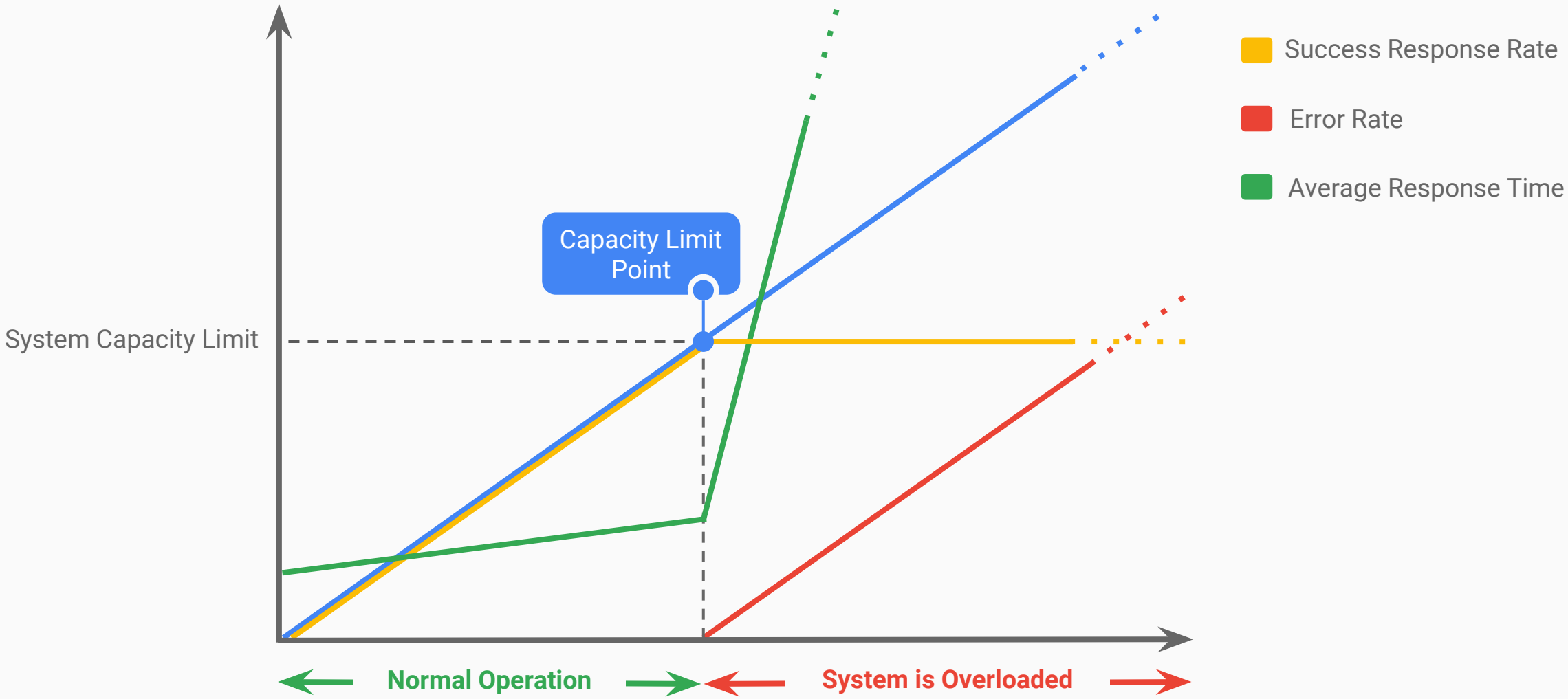
Google Cloud

Edge

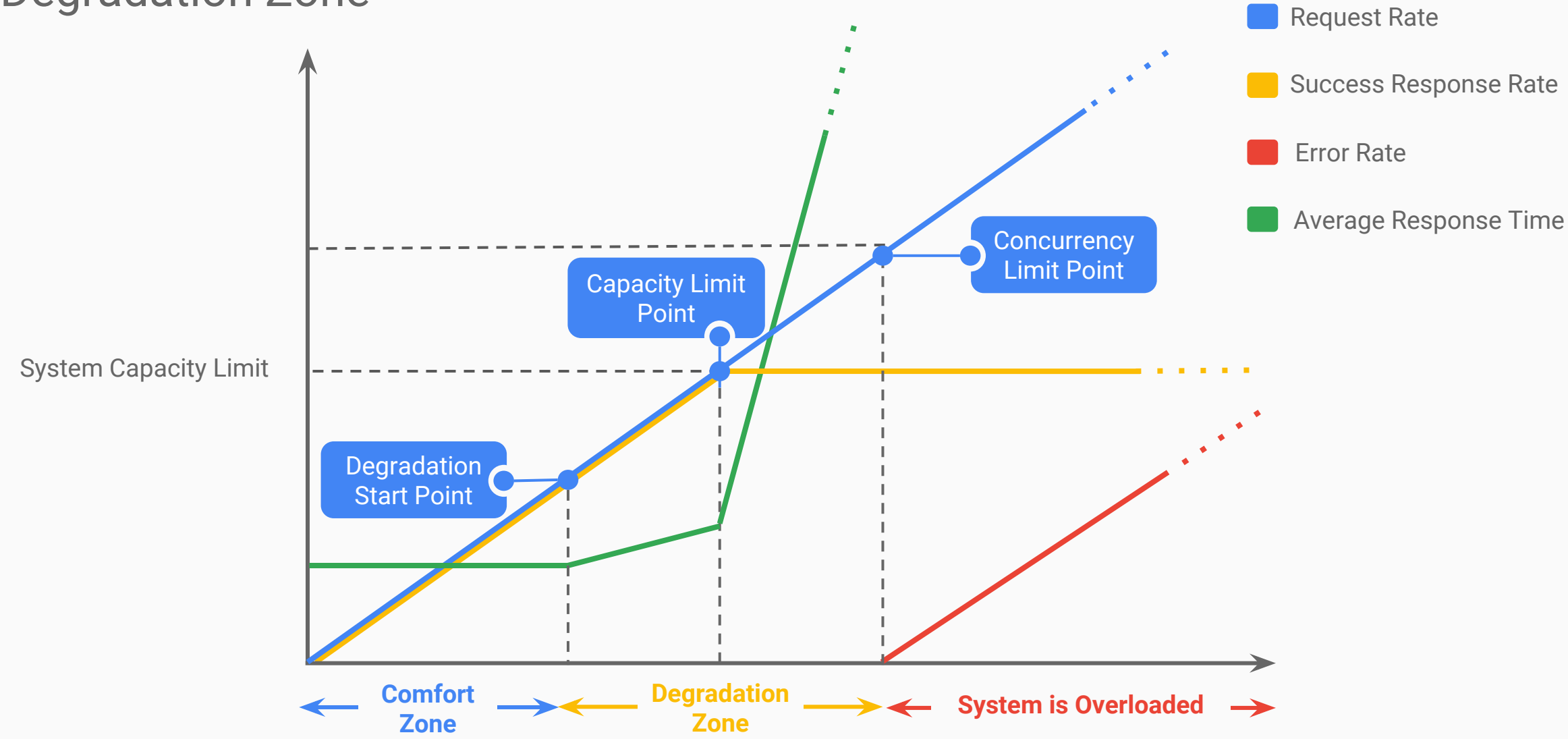
Performance Testing and Optimization

General Considerations

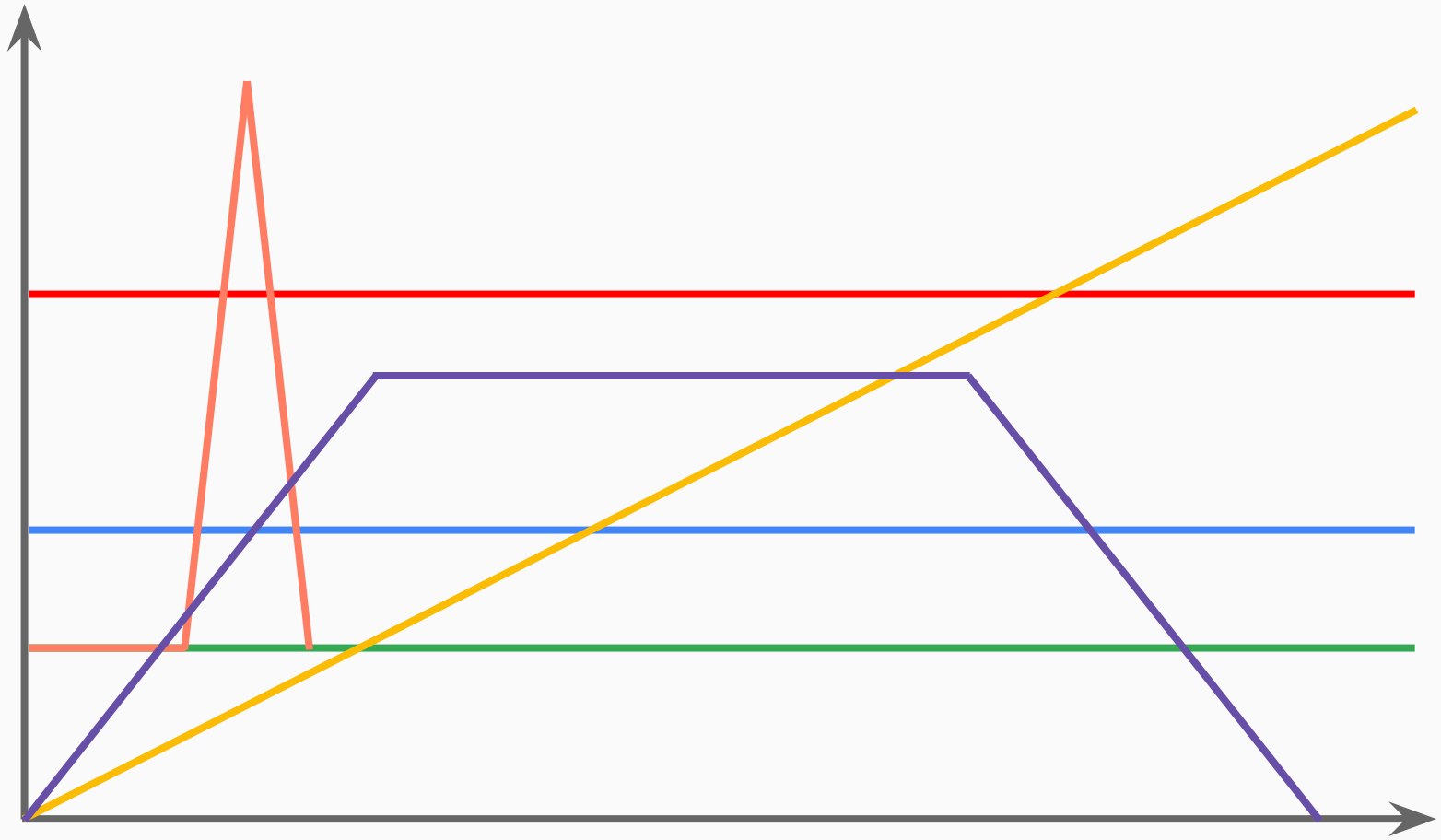
Capacity Limit Point



Degradation Zone



Performance Test Types (I)



Performance Test Types (II)

Stress Test

Progressive increase of the load applied until the limits of the system are found.

(Peak) Load Test

Conducted to make sure that a business objective (simulate busiest period, e.g. end of month payroll cut-off) in terms of performance are met. The load is generally ramped-up, kept running for a while at a fixed level, in order to verify that there is no significant degradation and finally ramped-down.

Spike Test

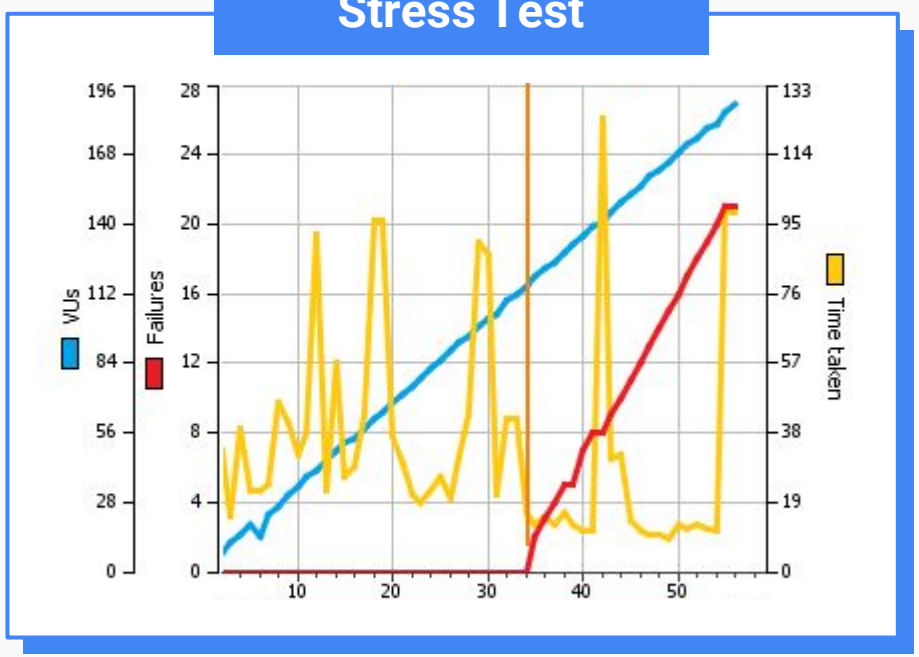
Sudden spike of load that goes over and above the maximum design capacity, just to see how the application can deal with a big spiking load.

Soak Test

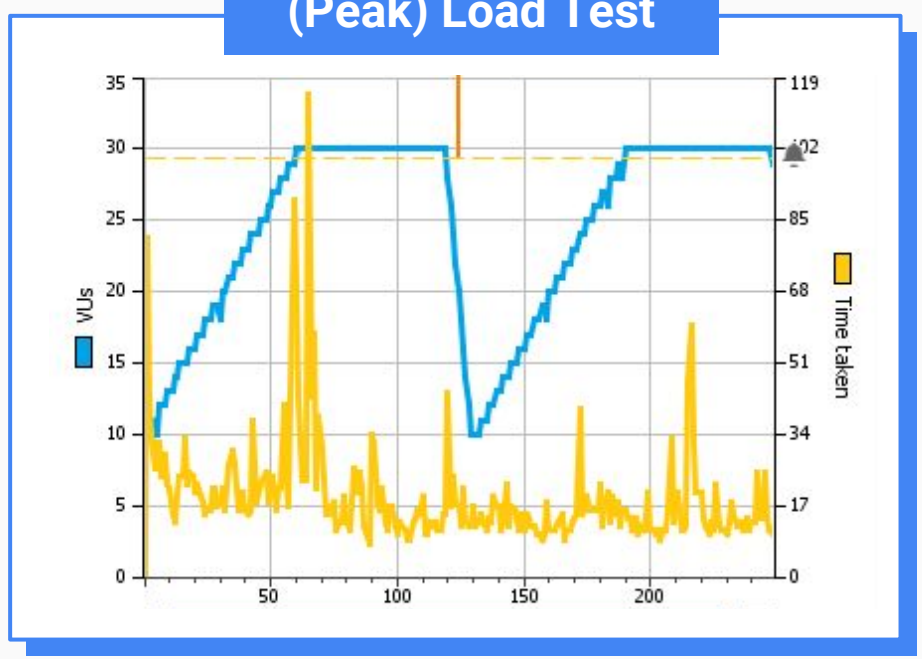
Constant load above the normal utilization rate run over a weekend or over a 24-hour period, while constantly monitoring the hardware resources and IO operations. The purpose of this test is generally looking for things such as memory leaks.

Performance Test Types (III)

Stress Test

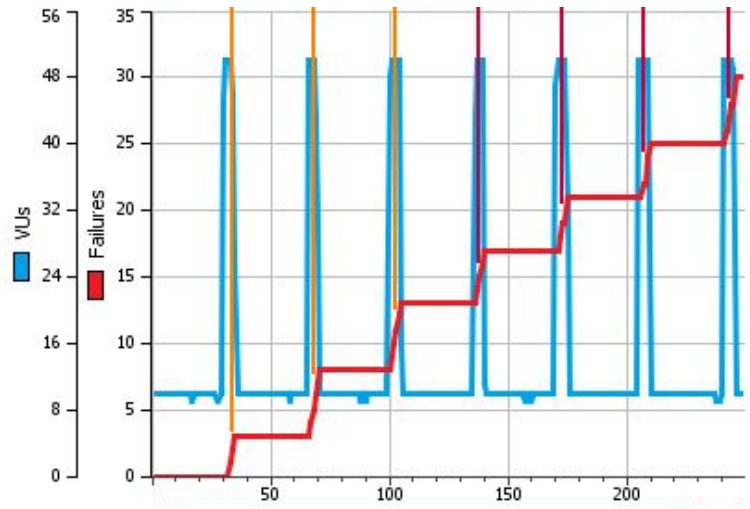


(Peak) Load Test

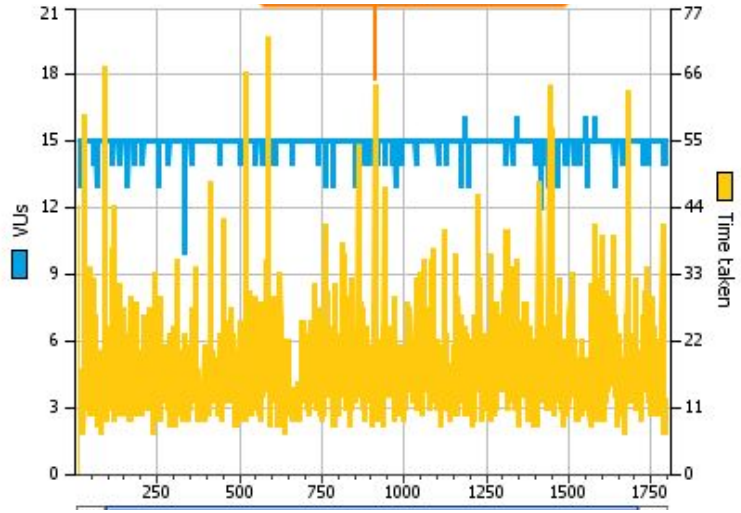


Performance Test Examples (IV)

Spike Test

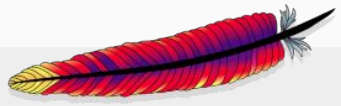


Soak Test



Tools
Apache

ApacheBench



Pros

- Zero-configuration required.
- Easy to learn and to run first test.
- Small CPU / memory footprint.

Cons

- Tests only one URL at a time.
- Has a hard set limit the maximum concurrency.
- Can't run distributed tests.

APACHE JMeter

Pros

- Can test multiple URLs in a single load test.
- Can run distributed tests, even in cloud.
- Can be easily integrated into CI.

Cons

- Hard to configure (Java tuning required)
- High memory consumption GUI mode (use command line and minimize the number of reports to achieve better results)
- Difficult to run complex scenarios.

Tools

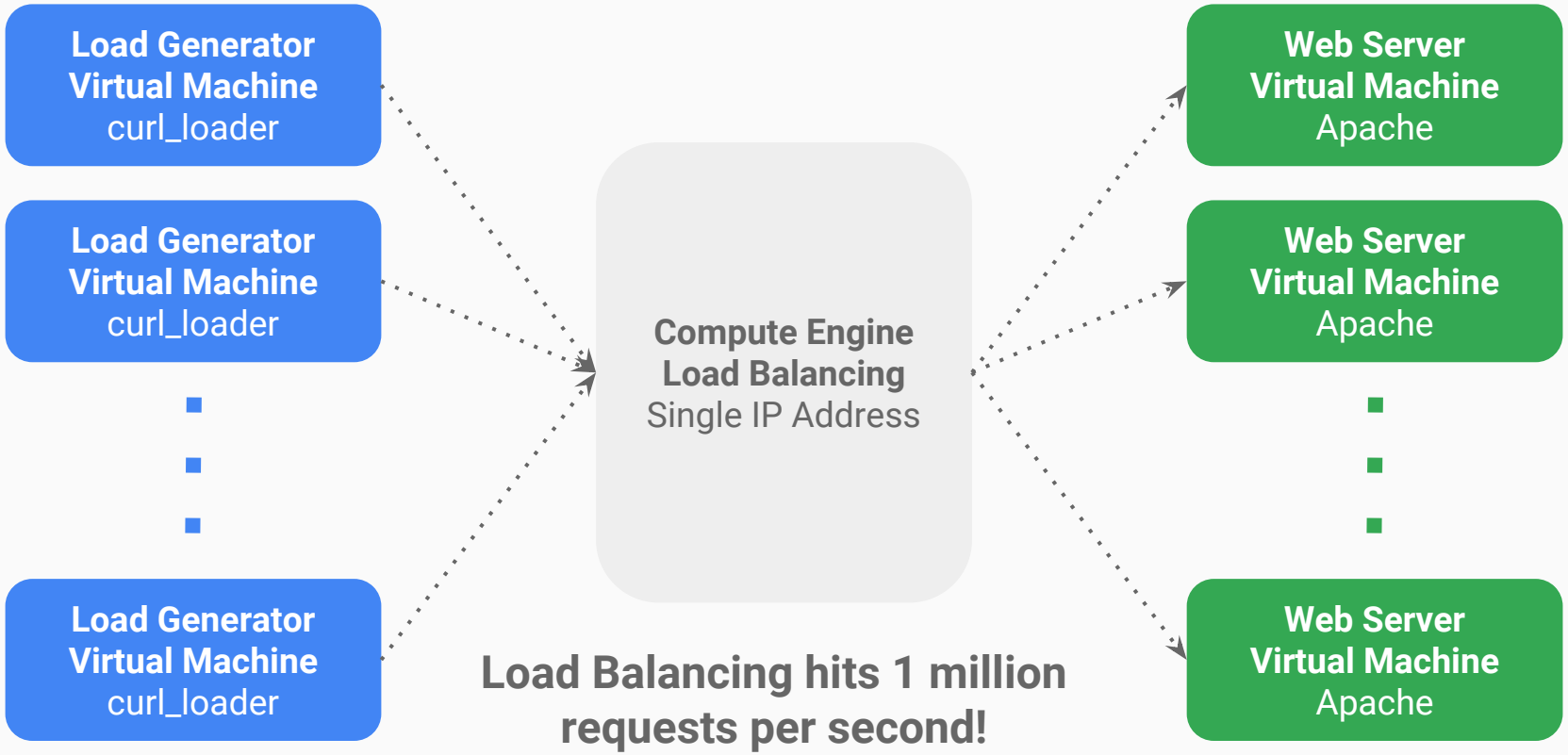
curl-loader (I)

- Open source, C language
- Fundamental study: the curl loader README file
- Very high loads 100-150K HTTP 1.1 clients from a single PC, quad-core modern Intel / AMD CPU, 8 GB RAM
- Used in real world (256 VM cores, 16 threads and 1000 connections):

  Compute Engine Load Balancing hits 1 million requests per second! (GCP Blog)

Tools

`curl-loader` (II)



Load Balancing hits 1 million requests per second!

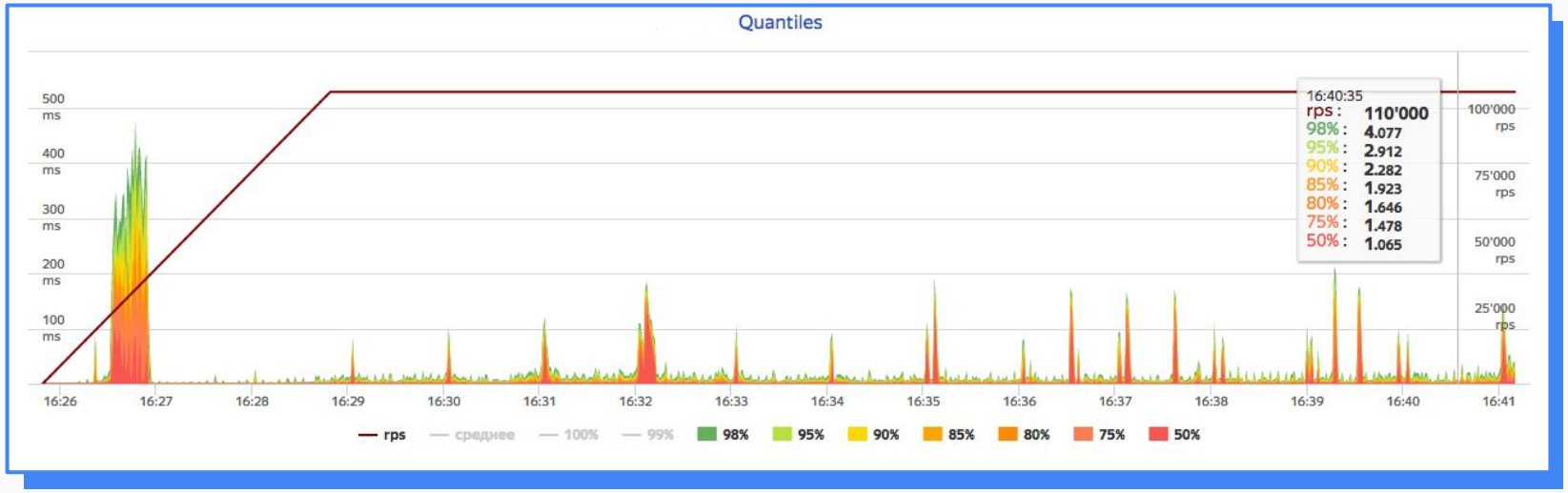
64 VMs each with **4 cores**

200 VMs each with **1 core**

Tools

Yandex Tank

- Open-source, written in Python.
- Extensible, supports different load generators: phantom, JMeter, BFG. Default load generator: phantom (up to 100,000TPS, written in C++).
- Console and HTML reporting.
- Graphite (enterprise monitoring tool) support.



Tools

JMeter EC2

- Open-source, bash shell wrapper.
- Run local `JMeter` files either using `Amazon EC2` service or any comma-delimited list of hosts.
- Summary results are printed to the console as the script runs and then all result data is downloaded and concatenated to one file when the test completes, ready for more detailed analysis offline.
- By default, it will launch required hardware using `Amazon EC2` using `Ubuntu AMIs`, dynamically installing `Java` and `JMeter`.

Edge Specifics

Synchronous Data Access

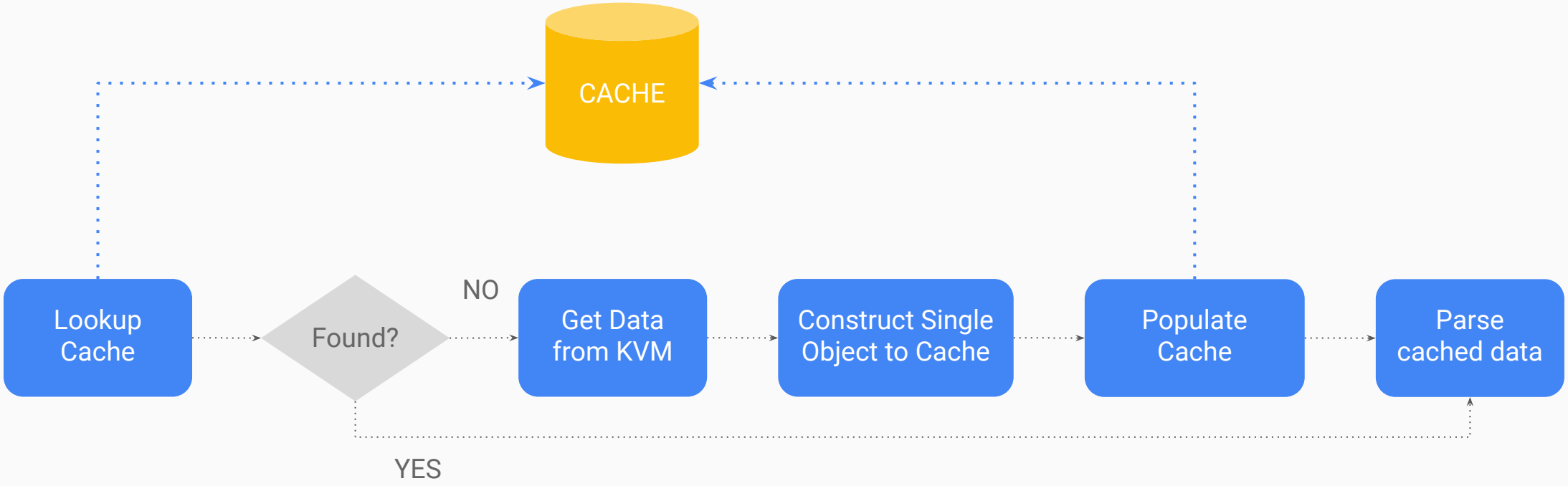
- Policies waiting for data from remote / external sources (e.g., Service Callout)
- Policies querying internal data - `C*` access (e.g., KVM, Quota)

Caching (I)

Try to cache data as much as you can in Edge layer:

- Use `LookupCache` / `PopulateCache` policies to cache arbitrary purpose data caching
- Use `ResponseCache` policy to cache target responses - supporting `HTTP/1.1` caching headers

Caching (II)



https://community.apigee.com/articles/24906/a-pattern-for-caching-kvm-values.html

Asynchronous Quota

- Synchronous `Quota` goes to `C*` for every single request in order to provide 100% accuracy – this is a blocking event.
- Use async `Quota` as much as you can.

```
<Quota name="DailyAsyncQuota" async="true">
    <Interval>1</Interval>
    <TimeUnit>day</TimeUnit>
    <Allow count="100" />
    <Distributed>true</Distributed>
    <Synchronous>false</Synchronous>
    <Identifier ref="request.queryparam.apikey" />
    <AsynchronousConfiguration>
      <SyncIntervalInSeconds>20</SyncIntervalInSeconds>
      <SyncMessageCount>5</SyncMessageCount>
    </AsynchronousConfiguration>
</Quota>
```

Asynchronous HTTP Callout

- Edge Service Callout policy performs synchronous HTTP callout to a single target. It's a blocking operation.
- Use custom code for asynchronous operations, e.g. `JavaScript`, `Node.js`

```
// SYNCHRONOUS

var logglyRequest = new Request(url, "POST", headers, logData);
var logglyResponse = httpClient.send(logglyRequest);
```

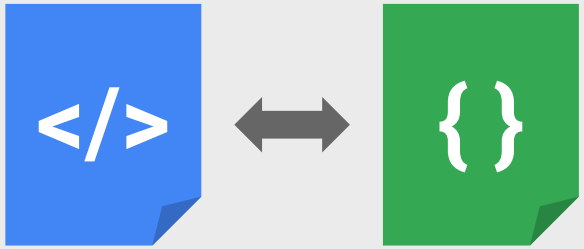
```
// ASYNCHRONOUS

var london = httpClient.get('http://weather.yahooapis.com/forecastrss?w=2467861');
var newyork = httpClient.get('http://weather.yahooapis.com/forecastrss?w=2459115');

// Thread is paused until the response is returned or error
// or step time limit has been reached.
london.waitForComplete();

// Thread is paused for a maximum of 100 ms
newyork.waitForComplete(100);
if (london.isSuccess() && newyork.isSuccess() { ... }
```

Data Transformation



- Be mindful of the data sizes you are transforming.
- Try to combine data extraction logic into single policy (e.g. Extract Variables, Xpath, Java code to load XML into DOM, etc).