$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$P_\lambda(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

$$B(k|p,n) = \binom{n}{k} p^k (1-p)^{n-k}$$

Functional Load Testing

GERALD MÜCKE

JAVALAND 2018

This talk is not about

- Load Testing a Function
- Loading a Function Test
- Loading a Test Function

- Testing a Load Function
- Test Loading a Function
- Testing a Function Load

It’s about using Functions to conduct Load Testing

Guess a Mathematician

Siméon Denis Poisson

Carl Friedrich Gauss

Jakob I. Bernoulli

What is Testing?

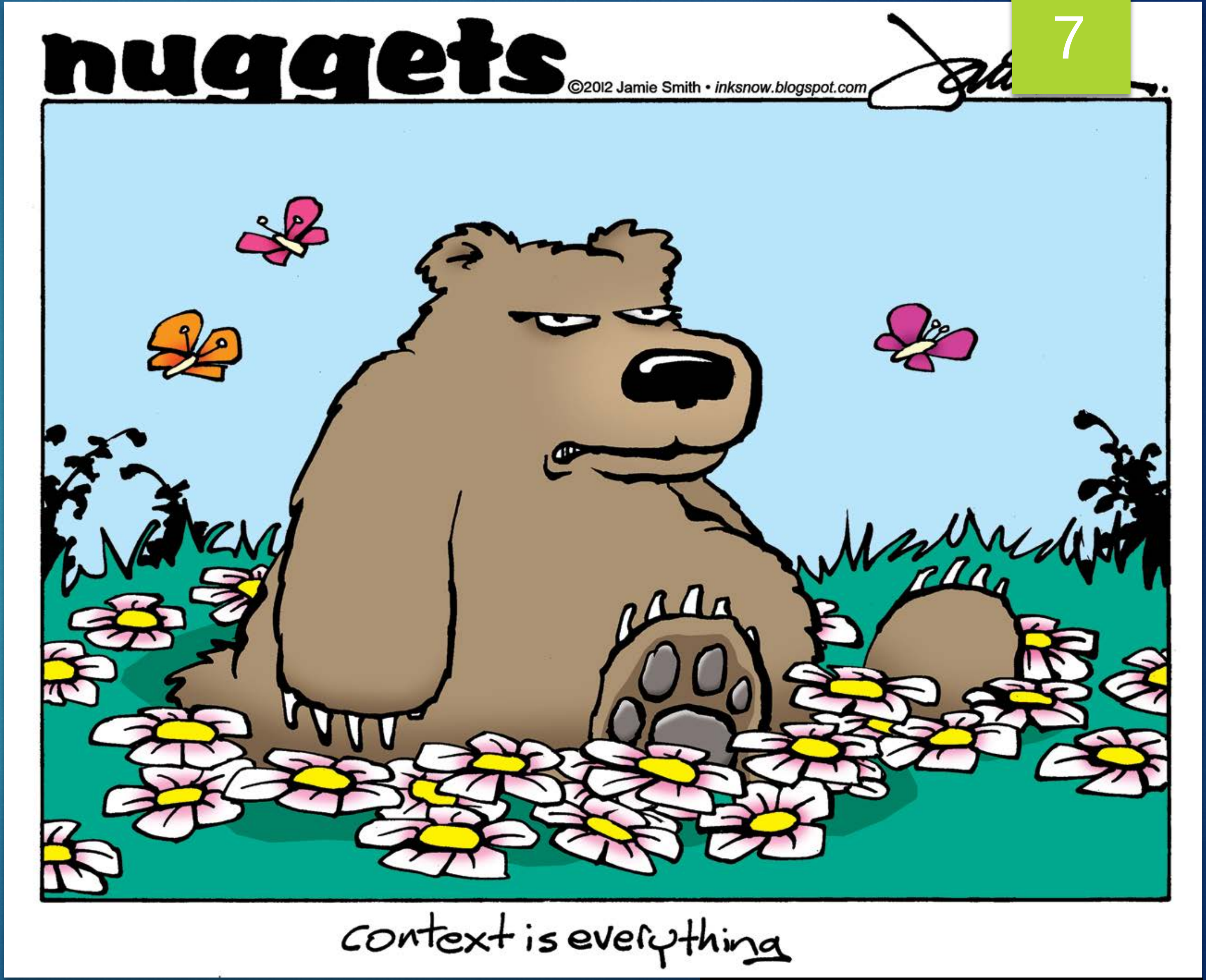
- ~~“Software testing is a process of executing a program or application with the intent of finding the software bugs.” (ISTQB)~~

- Testing is a information-providing service,
  not a "quality assurance" function.
- The value of testing is determined by whether it provides useful and timely information.
- A tester is a customer advocate.

Context is everything

- The value of any **practice** depends on its **context**.
- There are **good practices in context**, but **there are no best practices**.

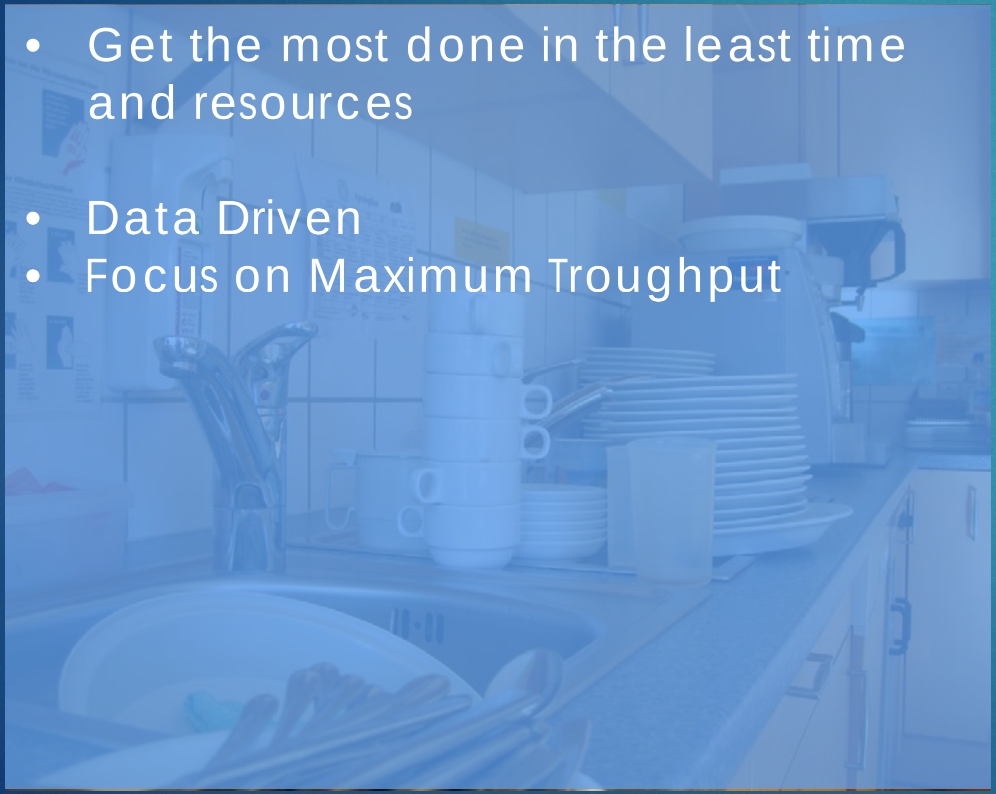
(From Seven Principles of Context Driven Testing)



How to do Performance Testing?

Type of Load Test

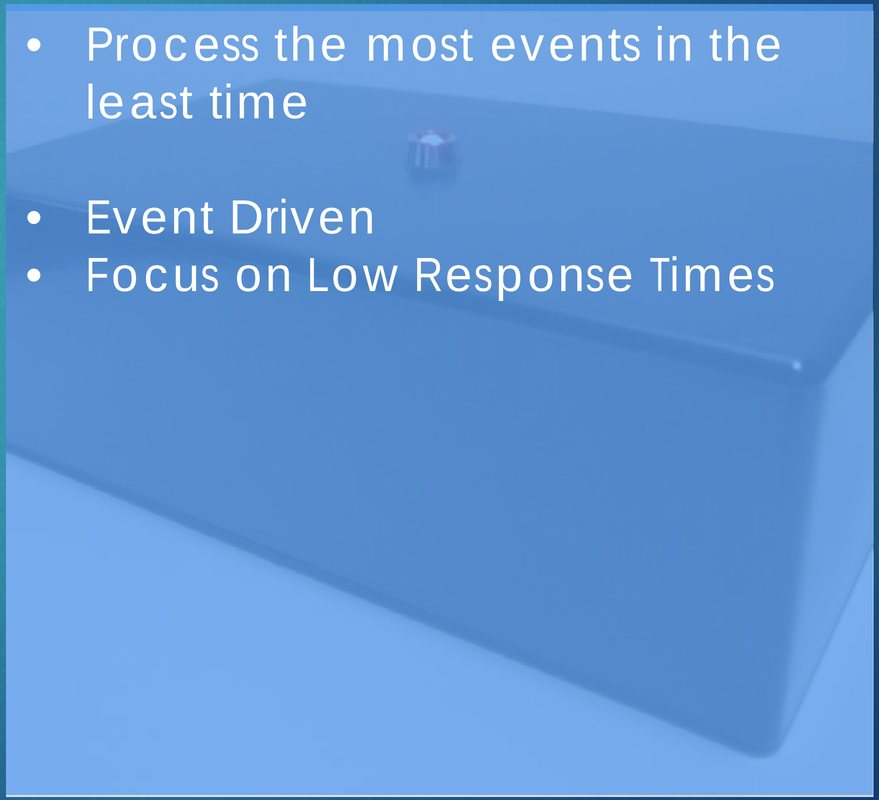
Batch



- Get the most done in the least time and resources

- Data Driven
- Focus on Maximum Troughput

Online



- Process the most events in the least time

- Event Driven
- Focus on Low Response Times

Input to Load Test Scenarios

- Performance Requirements
  - Target Users (Concurrent, per Duration, Total)
  - Response Time Targets (90%, 95%, 99%)
  - Throughput
- Historical Data
  - Number of Total Users per Duration
  - Number of Concurrent Users
  - Peak Loads (Peak Month/Day/Hour/Minute)
  - Request Logs
- Educated Guesses / Gut Feeling

Example Requirements

- The system is capable of
  - Serving 1000 concurrent users with an average Response Time of 1.5s
  - Source: The Project Manager

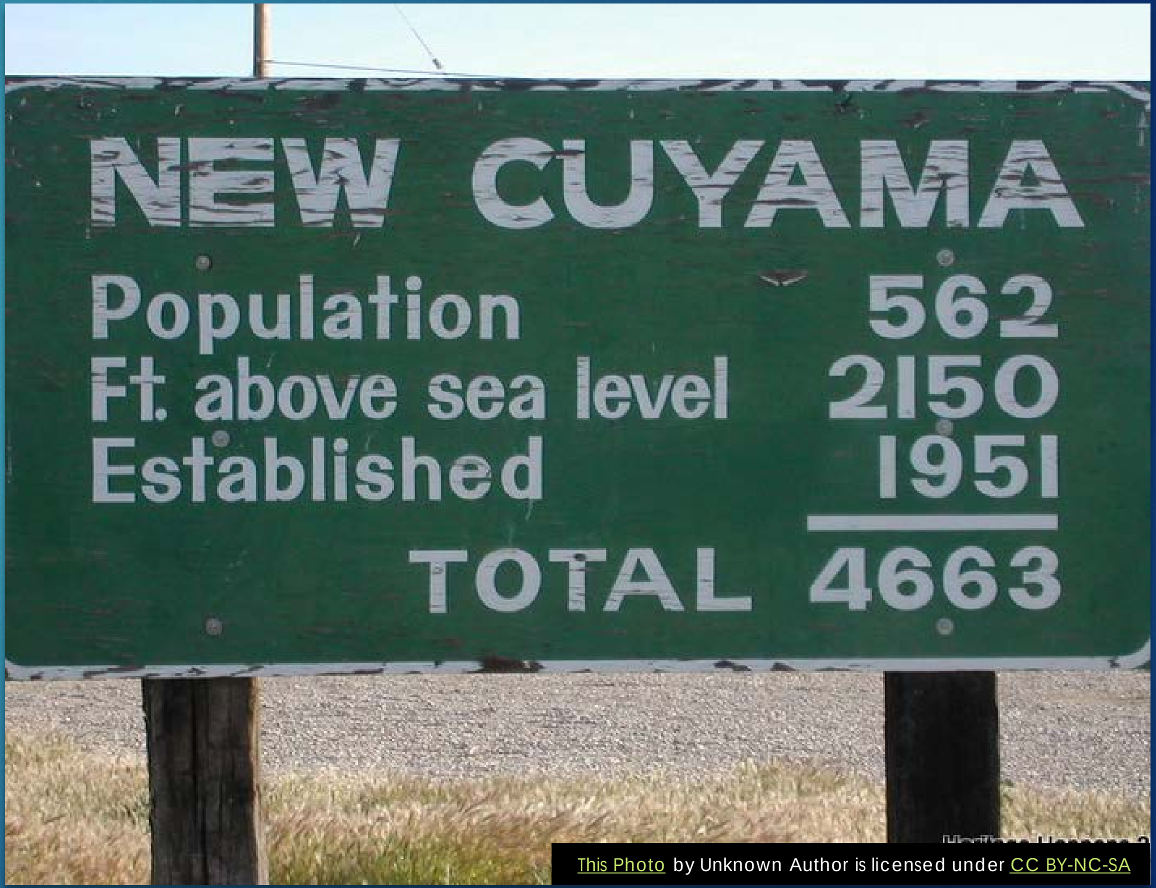
- What are the most relevant information?

Numbers need a Context

- Concurrent Users (CPU)
- vs. Concurrent Sessions (Memory)
- vs. Users per Period (Capacity)

- Average
- vs. perceived Average (90 %)
- vs. Percentiles (95%,99%,99.9%)

- 1000 Conc.Users, avg < 1.5s
- vs 1000 Users/h, 90% < 1.5s

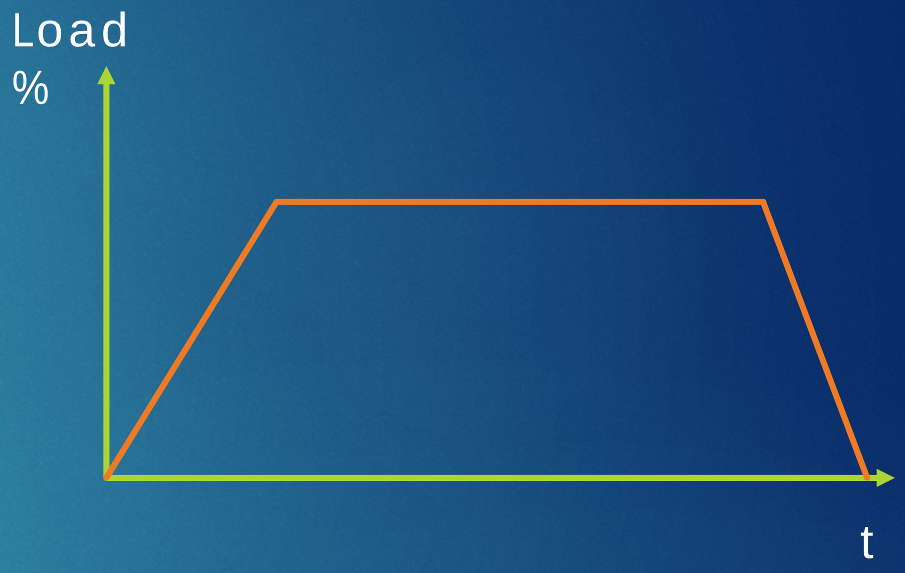


Load Testing Practices

- Soak Testing
  - Discover Leaks
  - SLA Regressions
- Stress Testing
  - Testing Stability
  - Overload / Recovery
- Benchmarking
  - Discover Regressions between different Versions or Configurations

A typical load test: Constant Load

- Constant Load
  - + Ramp up / down
    - Allowing the System to adapt to Load (warm up)
    - Distributes Load (virtual users)
- Good for:
  - Finding latent bugs, i.e.Memory or Resource Leaks
  - Precise Measurements
  - Regressions
  - Stability Issues
  - Statistical Response Times
  - Known Load Distributions

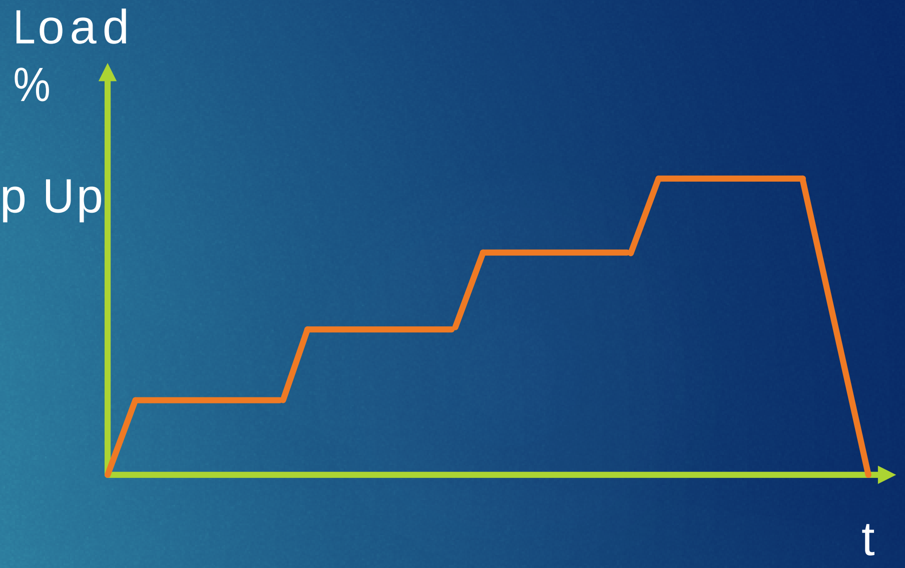


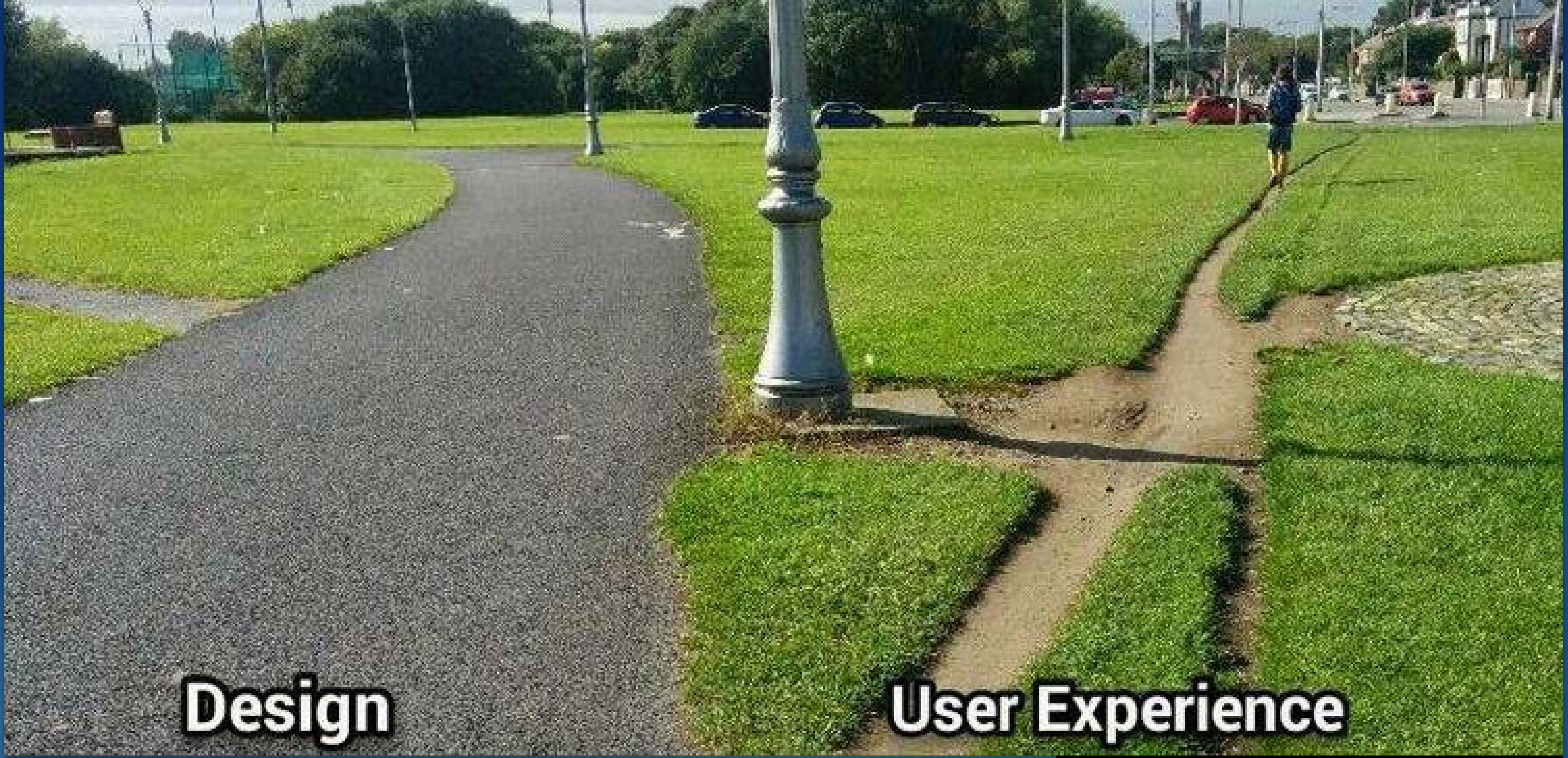
Why do a ramp up?

- System Warm-up
  - Allow JIT to optimize code
  - Allow Caches to be populated
  - Fetch or Initiate Resources (i.e. Database Connections)
  - Allow Queues and Buffers to fill to a stable level
  - Distribute Load evenly
- System's Performance Characteristics are non-linear during ramup

Synthetic Scenarios: Step Load

- Alternating Stable/Constant Load Phases and Ramp Up
- Good for:
  - Finding System Capacity
  - Finding Breaking Point





This Photo by Unknown Author is licensed under CC BY-NC-ND

But, from the user perspective …

What is the user value of...

- My car consumes 4.8 l per 100 km in the benchmark test

- Yet in the City, I constantly use more than 6.5 l per 100 km

What is the user value of...

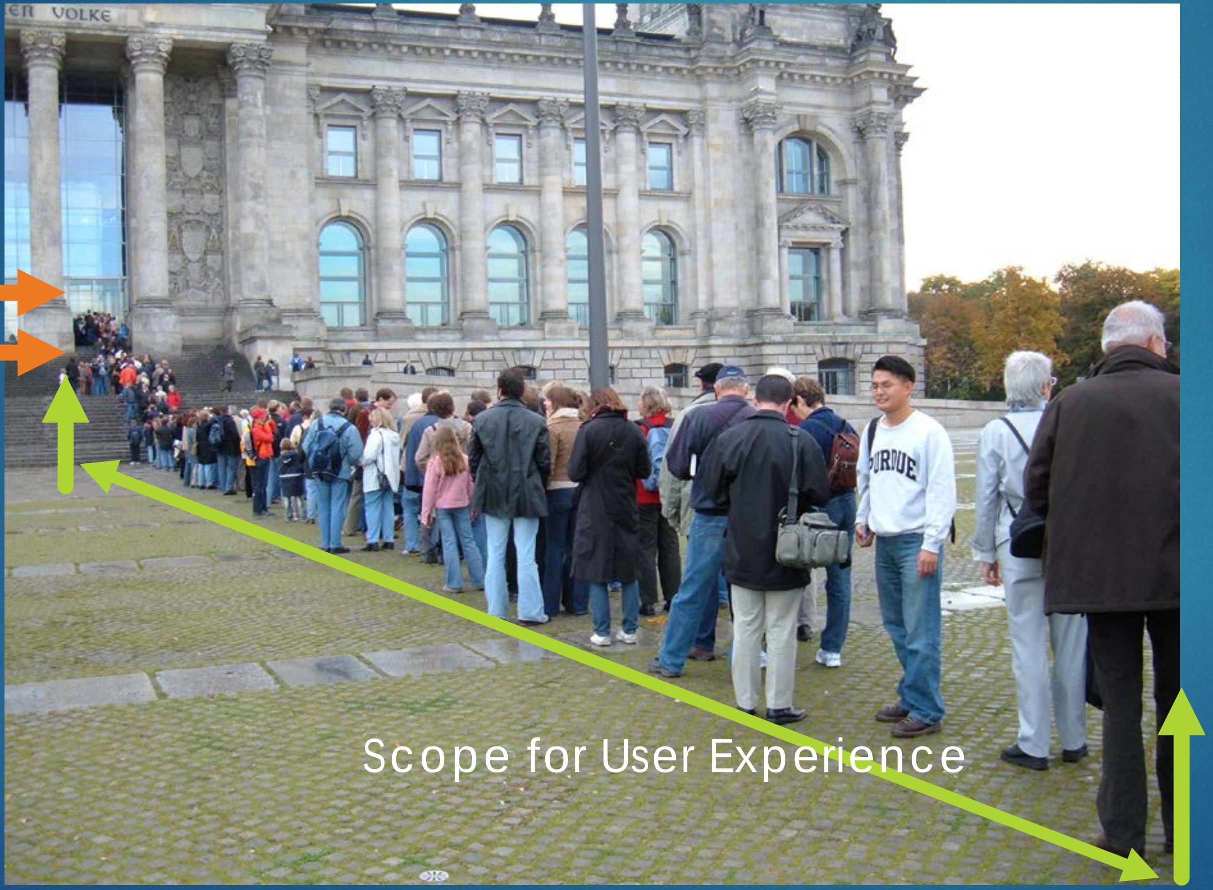
- 98% of all trains are on time
- Yet, I miss 1 in 10 connecting trains with a connection time of 3 mins
- Yet, I miss all connecting trains during peak hours (adding 30mins)

What is the use value of…

- With my car
  I can drive faster than 200 km/h
- Yet, I’m stuck in traffic jams due to road work adding a third lane

The context of measurement

Scope for Static Load Test


Scope for User Experience

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-SA

A typical simplification of the system's users

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-NC

Today's Challenges

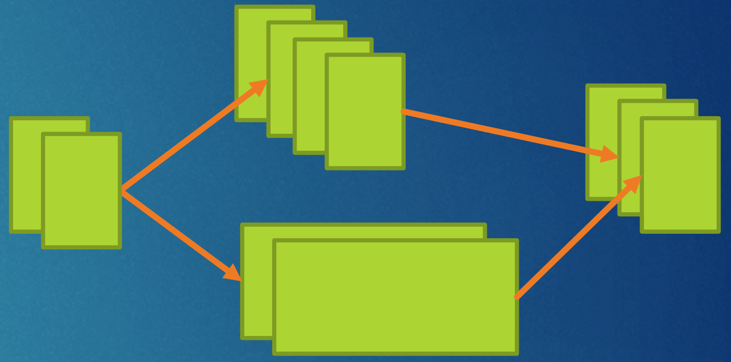
Real Life Scenarios

- Singular Events
  - Launch of Marketing Campaign / Black Friday Sales
  - Launch of new Service or WebSite
  - Users Logging in in the morning
  - Clients connect to a system on schedule
  - Automated Updates
- Daily Patterns
  - “McDonalds” M
  - asymmetric-M

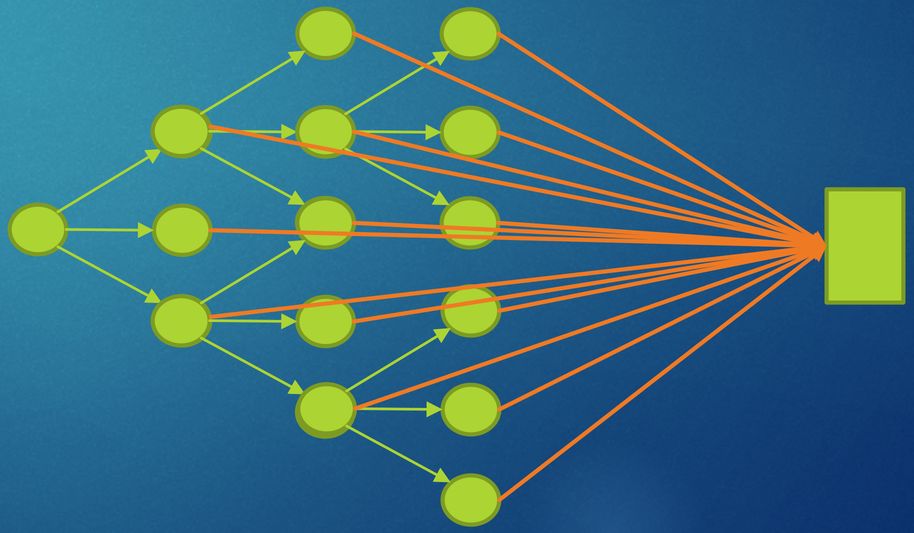
BLACK FRIDAY
The only day where you might get away with killing someone to grab some toy

The Challenges

- Dynamic Capacity
  - Systems automatically scale up or down
  - Systems are Virtualized (especially in the Cloud)



- Users come in Masses
  - A lot of people are online today
  - Social Media as Amplifier
  - Viral Marketing

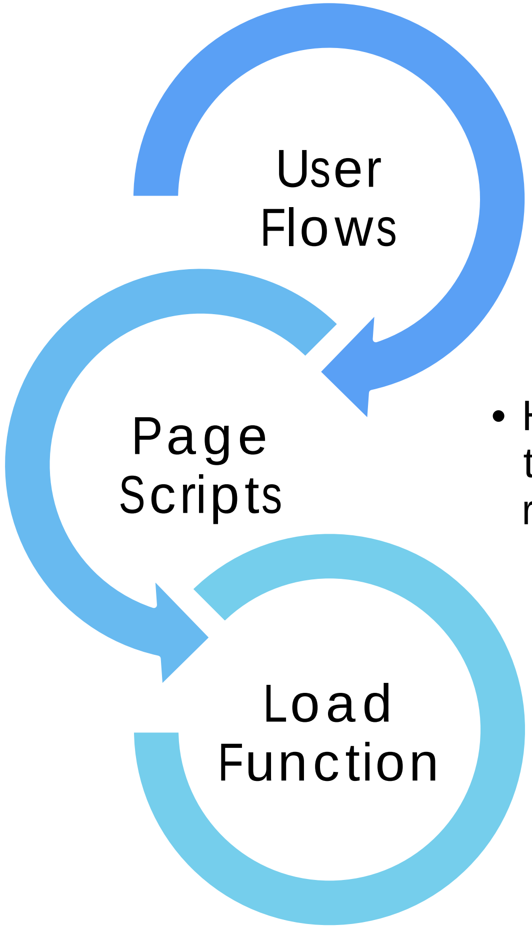


The information we need

- How are users affected during scaling
- How are the Response Times (with regard to SLAs)
  - During unexpected loads
  - During typical usage (which is not constant)

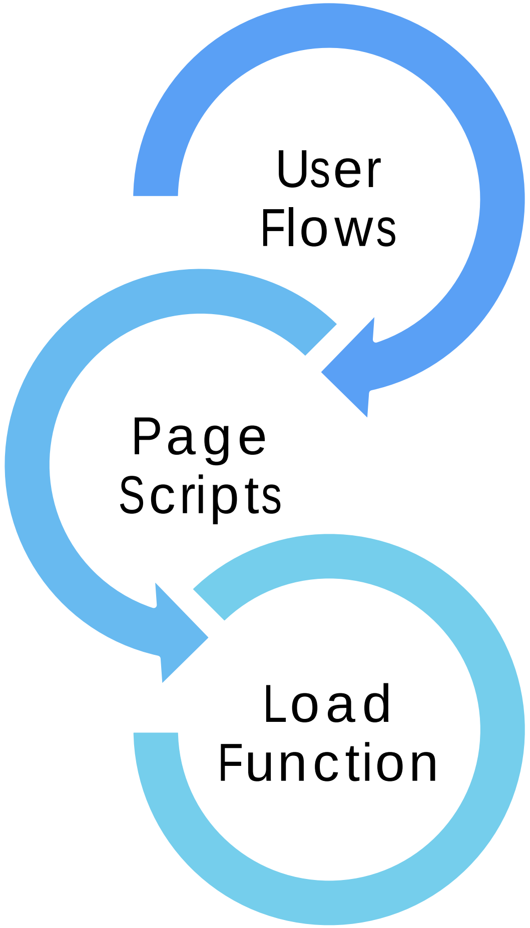
- In order to design a dynamic system that
  - Is cost-efficient (as small as possible during normal loads)
  - Can deal with (unexpected) peak loads
  - Fulfills the SLAs (Reliability, Response Times, Throughput, ...)

Three elements of a Load Test



- How groups of users navigate through your application
- How to fetch a typical page and it’s resources
- How the users arrive at your page

Three elements of a Load Test



User Flows

Page Scripts

Load Function

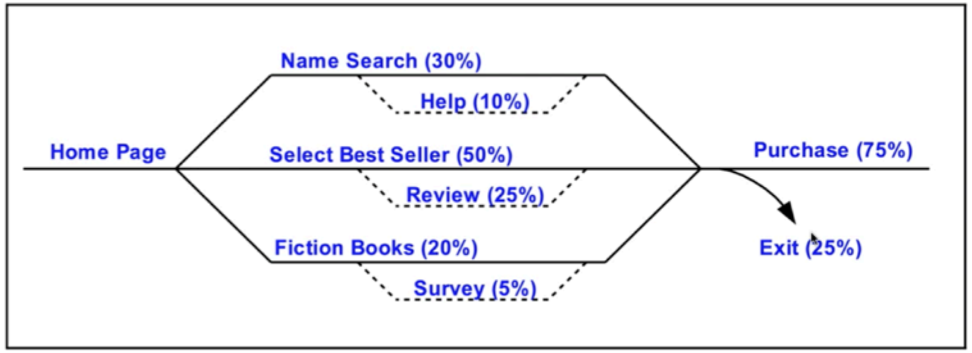
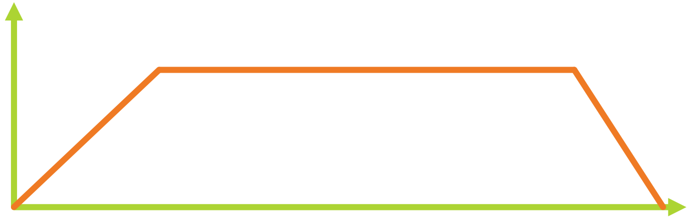


Image from User Experience Not Metrics, **Scott Barber** 2006

```
GET /index.html
GET /favicon.ico
GET /someScript.js
GET /img/background.jpg
```



Page Scripts

- A sequence of requests sent to the server
- The page itself (via GET)
  - Resources are inferred (implicit)
  - Resources are defined (explicit)
  - Resource Caching might be
    - activated (returning visits)
    - deactivated (first visit)
  - No external resources (doesn't generate load on server)
- A POST request (or any other non-GET request)
- Dynamic Sequence (i.e. on-type search queries)

User Flow Model

- Entry Points
- Split Points With %
- Exit Points With %
- Each Path is a User Scenario

- % indicates how many of the users follow that path

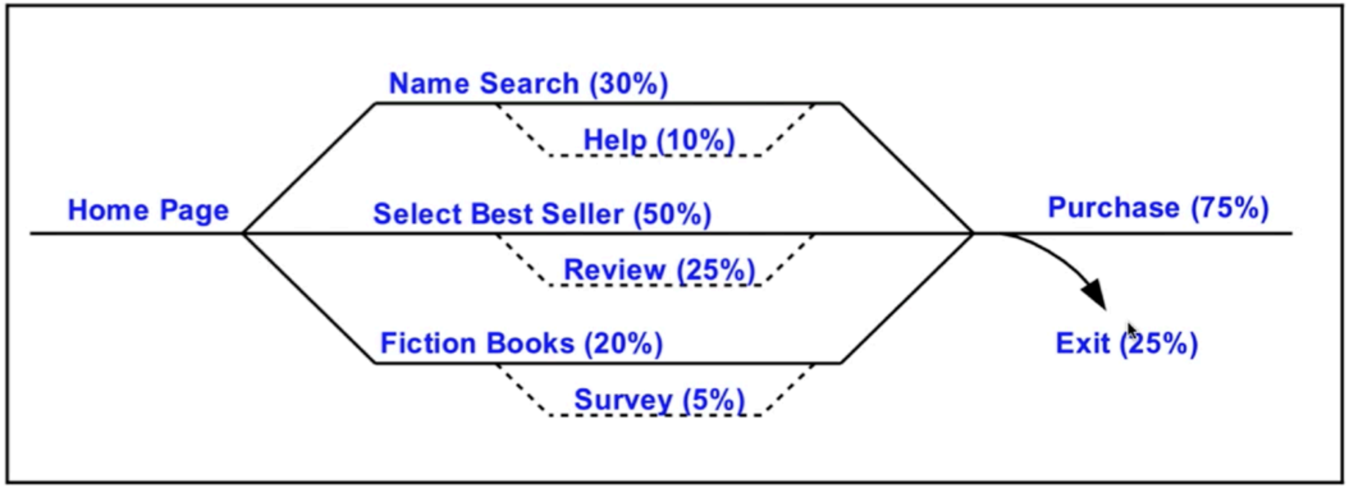


Image from User Experience Not Metrics, **Scott Barber** 2006

Load Function

What is a load function?

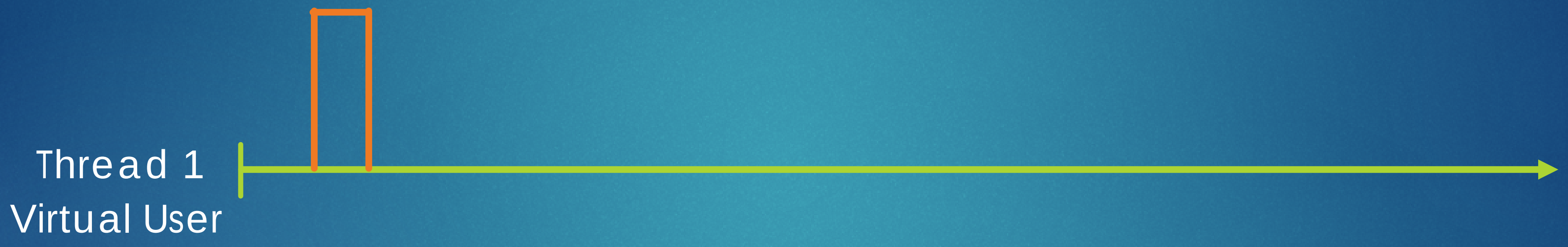
- How Load is distributed over time
- Defined by a rate of events in a period
  - **Users arriving at the application**
  - Requests
  - Actions / Clicks
  - Bytes
- Load Functions define when the Load Generator sends a request

Arrival Rate

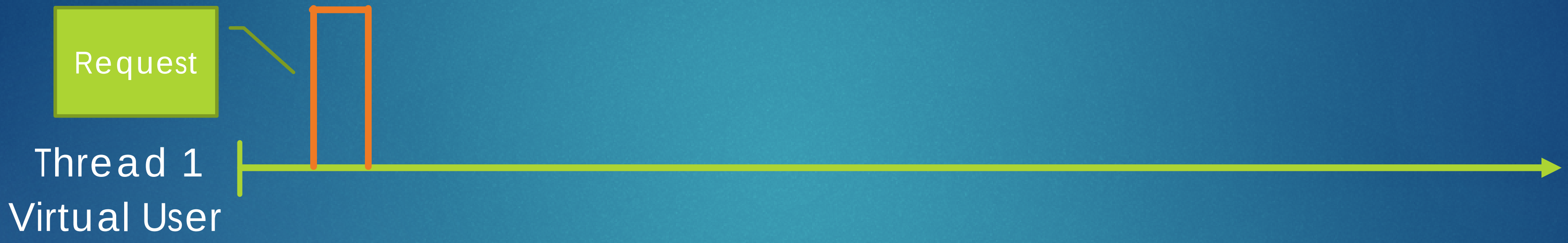
- Each User acts as an individuum, independent from all other users
- Unknown arrival rates can be best approximated, using a statistical distribution function

- We need a load generator that can generate
  - indepent user requests
  - user requests following an arrival rate defined by *f(t)*

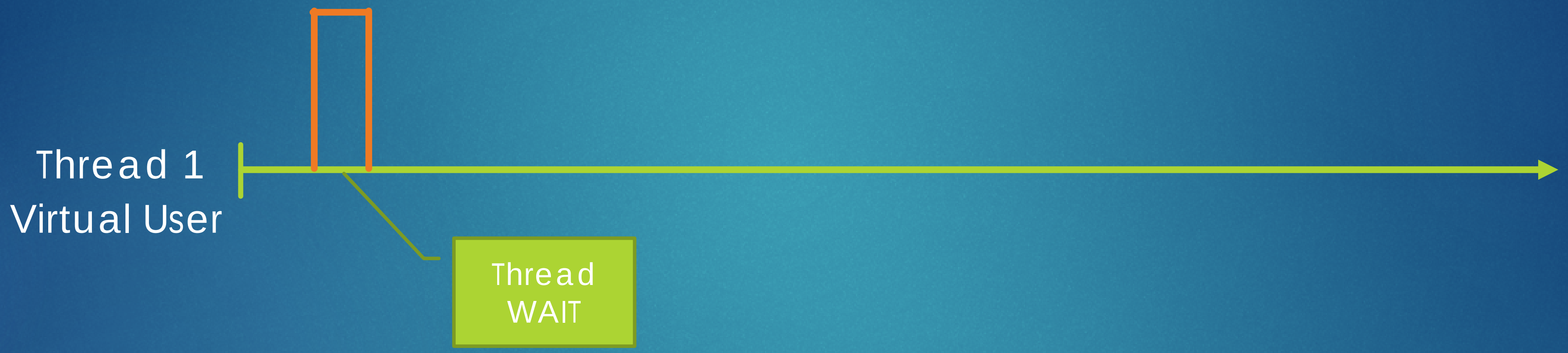
Thread based Load Generation



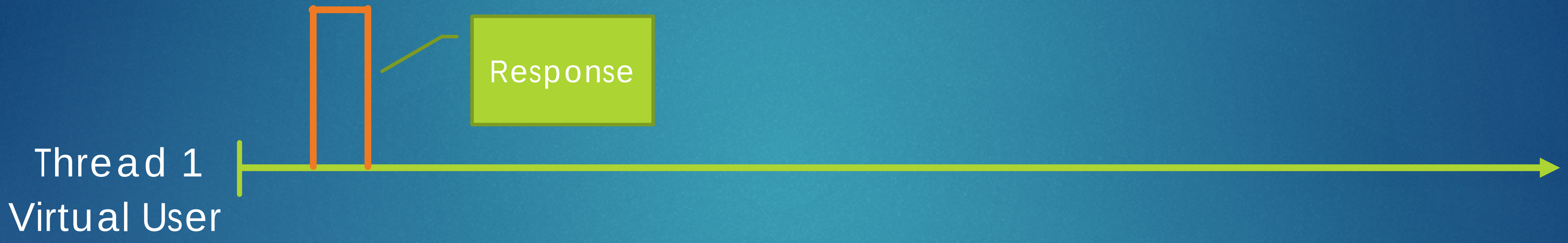
Thread based Load Generation



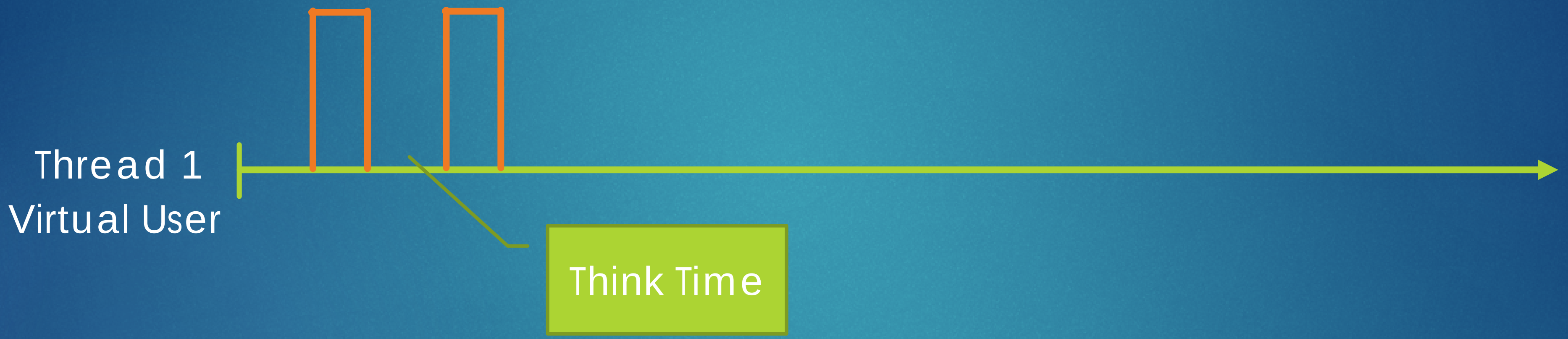
Thread based Load Generation



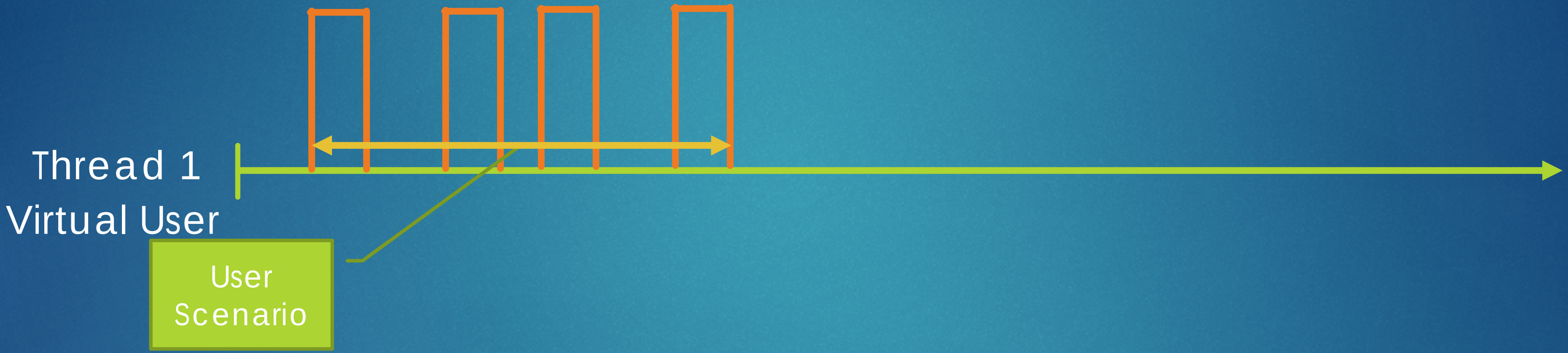
Thread based Load Generation



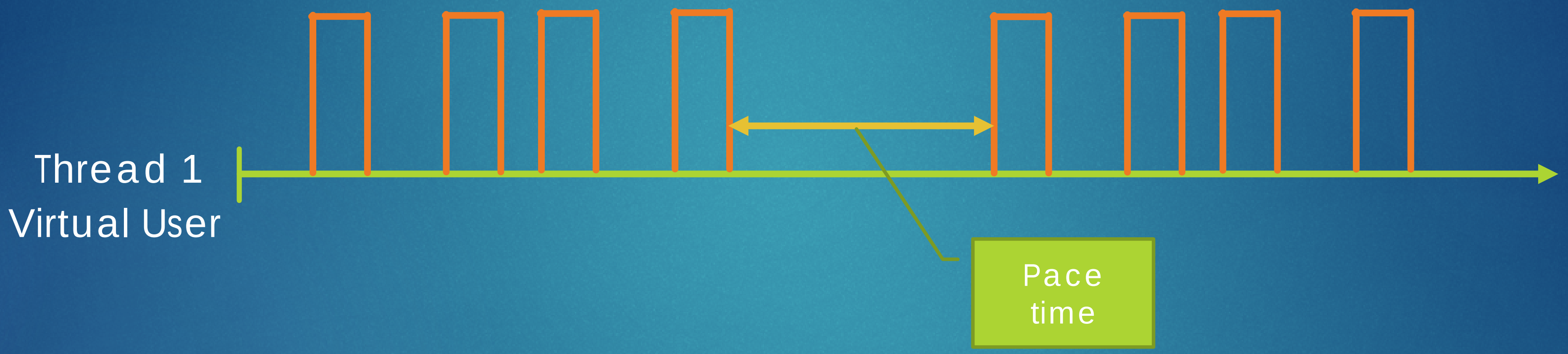
Thread based Load Generation



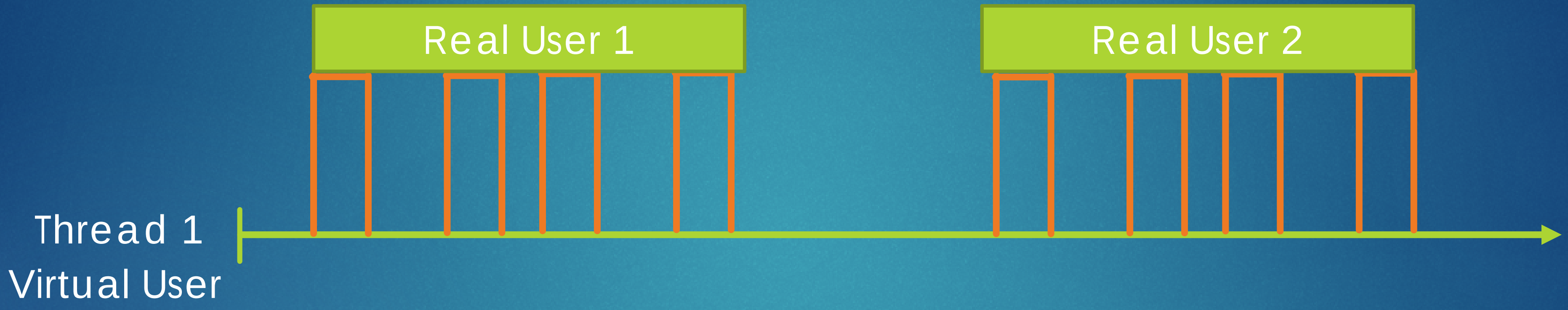
Thread based Load Generation



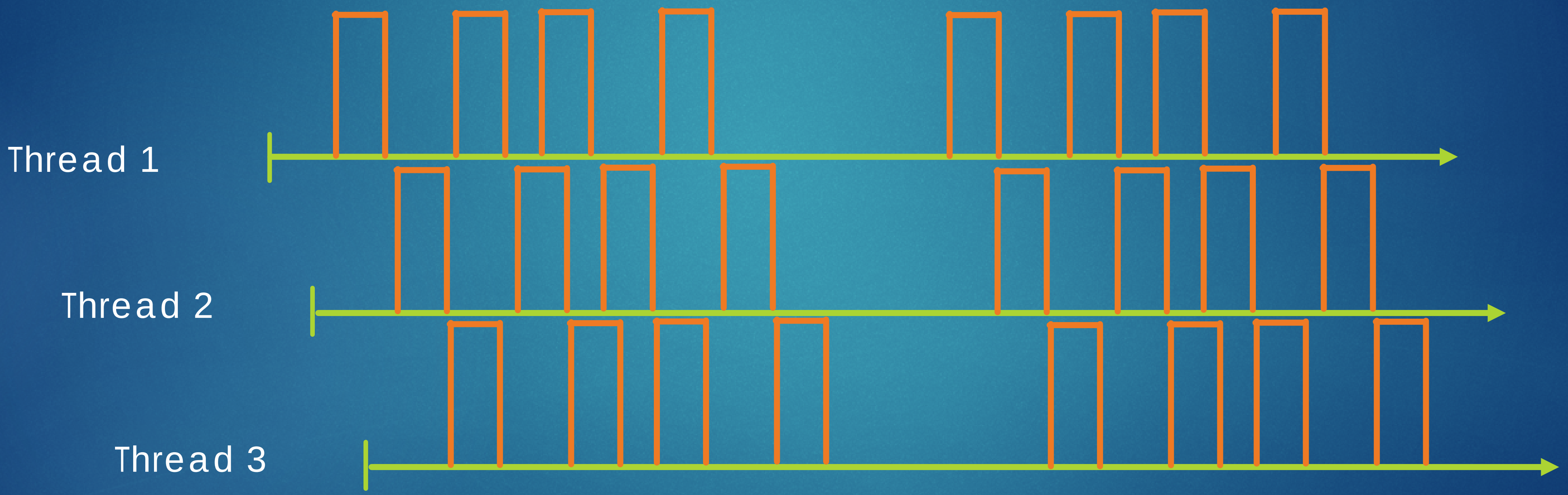
Thread based Load Generation



Thread based Load Generation



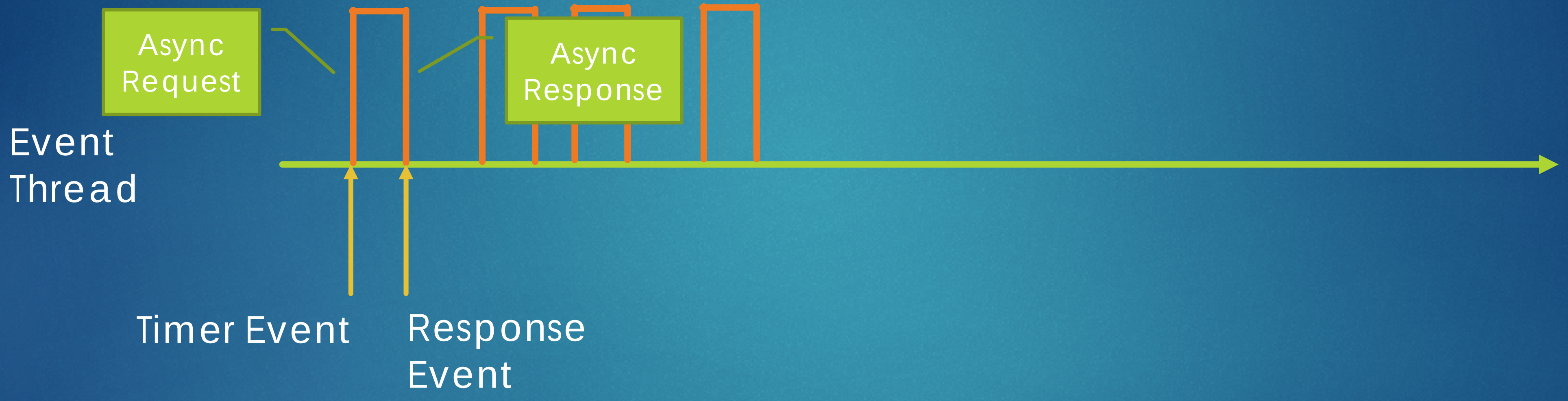
Thread based Load Generation



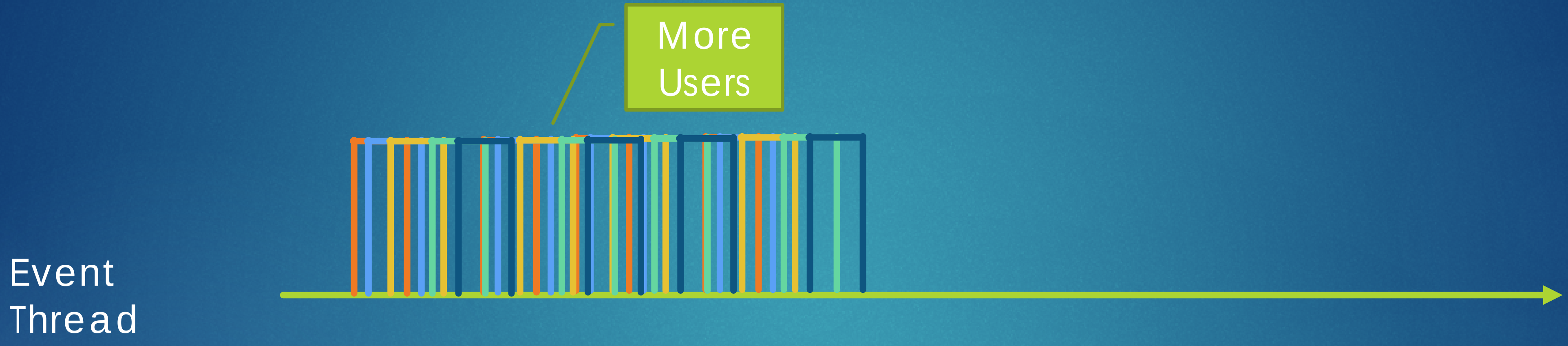
Thread based Load Generation

- Characteristics
  - Each Simulated Real User depends on it’s previous user
  - Each consecutive Request depends on Response for previous Request
  - Requires lot of Resource
  - Suffers from Coordinated Omission (SUT throttles Load Generator)
  - Rampup ensures that generated load is evenly distributed
  - Pacing ensures that load distribution remains stable
- Good for
  - Closed User Groups (i.e. Employees, Named Users), Dependant Users
  - Determine Capacity
- Tool Example: Apache JMeter

Event Based Load Generation



Event Based Load Generation



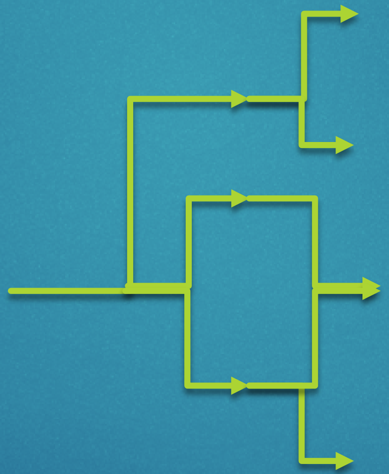
Event Based Load Generation

- Characteristics
  - Simulated Users are Independent
  - Sent Requests are independent from previous Requests
  - Requests/Responses are handled asynchronously
  - Requires less resources for same load as thread based systems
  - No Coordinated Omission
  - Limited by the Processing Capacity of the Event-Thread
  - Ramp up / down is defined by change in User Rate
- Good for
  - **Independent Users**
  - Open User Groups (i.e. for public web sites)
  - User Experience Rating
- Tool-Example: Gatling

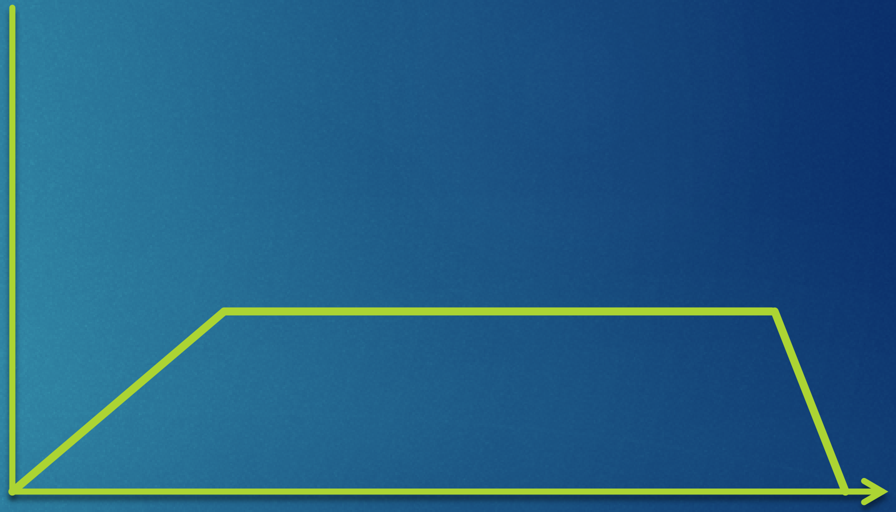
Gatling

Test Development with Gatling

Pages



Scenarios



Load Models

Test Development with Gatling

```
val page1 = exec(
 http("request_0")
  .get("/")
  .headers(acceptHtml)
  .resources(...))
```

Pages

```
val scn =
scenario("Simple")
 .exec(page1)
```

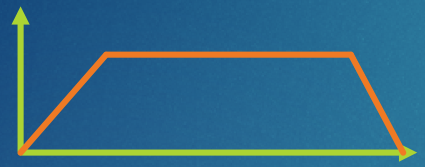
Scenarios

```
scn.inject(
  constantUsersPerSec(500)
    during (1 minute)
  )
```

Load Models

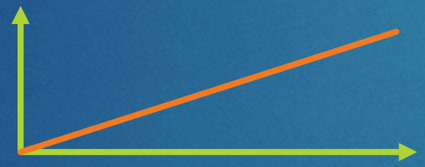
Typical Load Distribution Functions

- Equi-Distribution



```
scn.inject(
  rampUsersPerSec(10) to 20 during(10 minutes),
  constantUsersPerSec(20) during (20 minutes)
)
```

- Linear-Distribution



```
scn.inject(
   rampUsers(10) over(10 seconds)
)
```

Typical Load Models

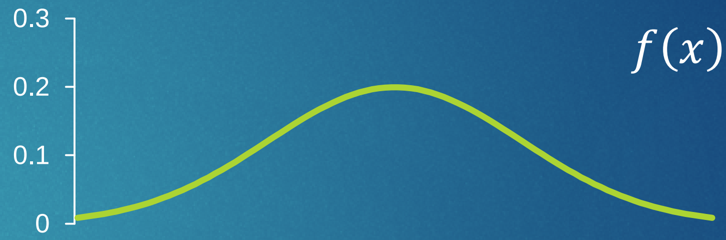
```
scn.inject(
   nothingFor(5 seconds),
   rampUsersPerSec(1) to 200 during(10 minutes),
   constantUsersPerSec(200) during (30 minute),
   rampUsersPerSec(200) to 400 during(10 minutes),
   constantUsersPerSec(400) during (30 minute),
)
```

List of Injection Steps

$$f(t) = \begin{cases} 0, & t < 5\,s \\ \frac{t}{3}, & 5\,s \le t < 600\,s \\ 200, & 600s \le t < 2400s \\ \frac{t}{3} + 200, & 2400s \le t < 3000s \\ 400, & t \ge 3000s \end{cases}$$

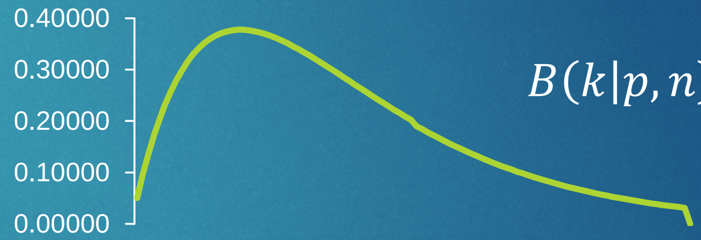
Statistical Distribution Functions

- Normal-Distribution (Gauss)
  - Daily usage patterns



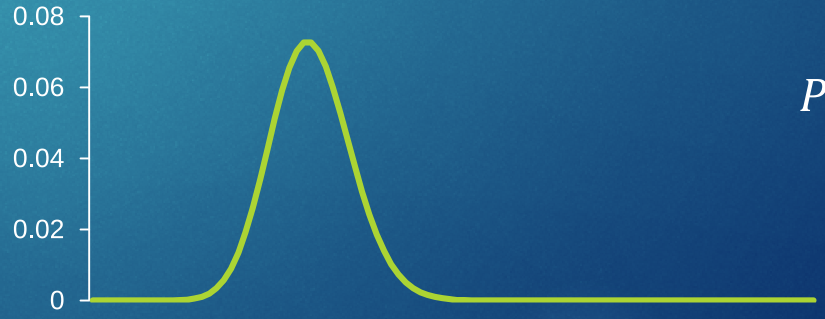
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

- Binomial Distribution (Bernoulli)



$$B(k|p,n) = \binom{n}{k} p^k (1-p)^{n-k}$$

- Poisson Distribution
  - Coordinated
  - i.e. Scheduled events



$$P_\lambda(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Combined Functions

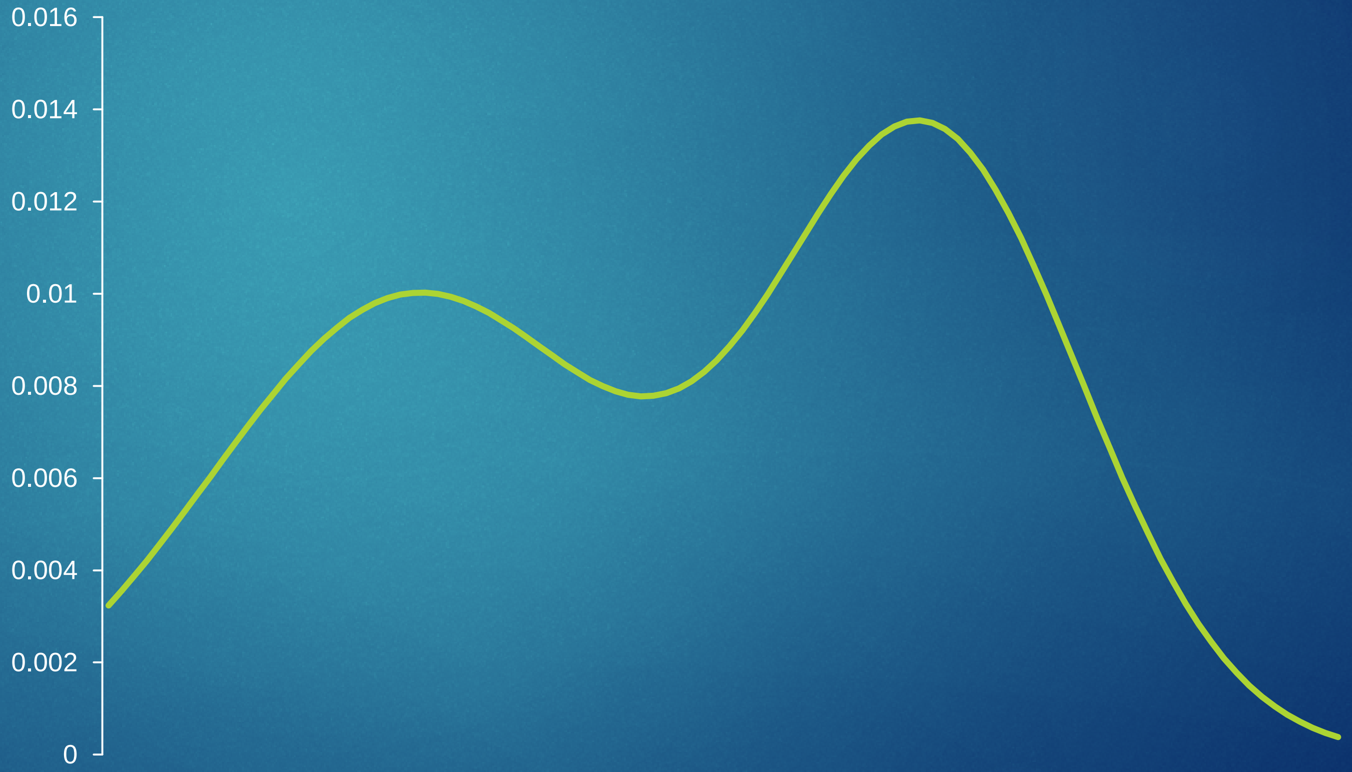
- Overlaying two Gauss Functions with different Parameters

$\sigma_1 = 20$

$\sigma_2 = 15$

$\mu_1 = 30$

$\mu_2 = 80$

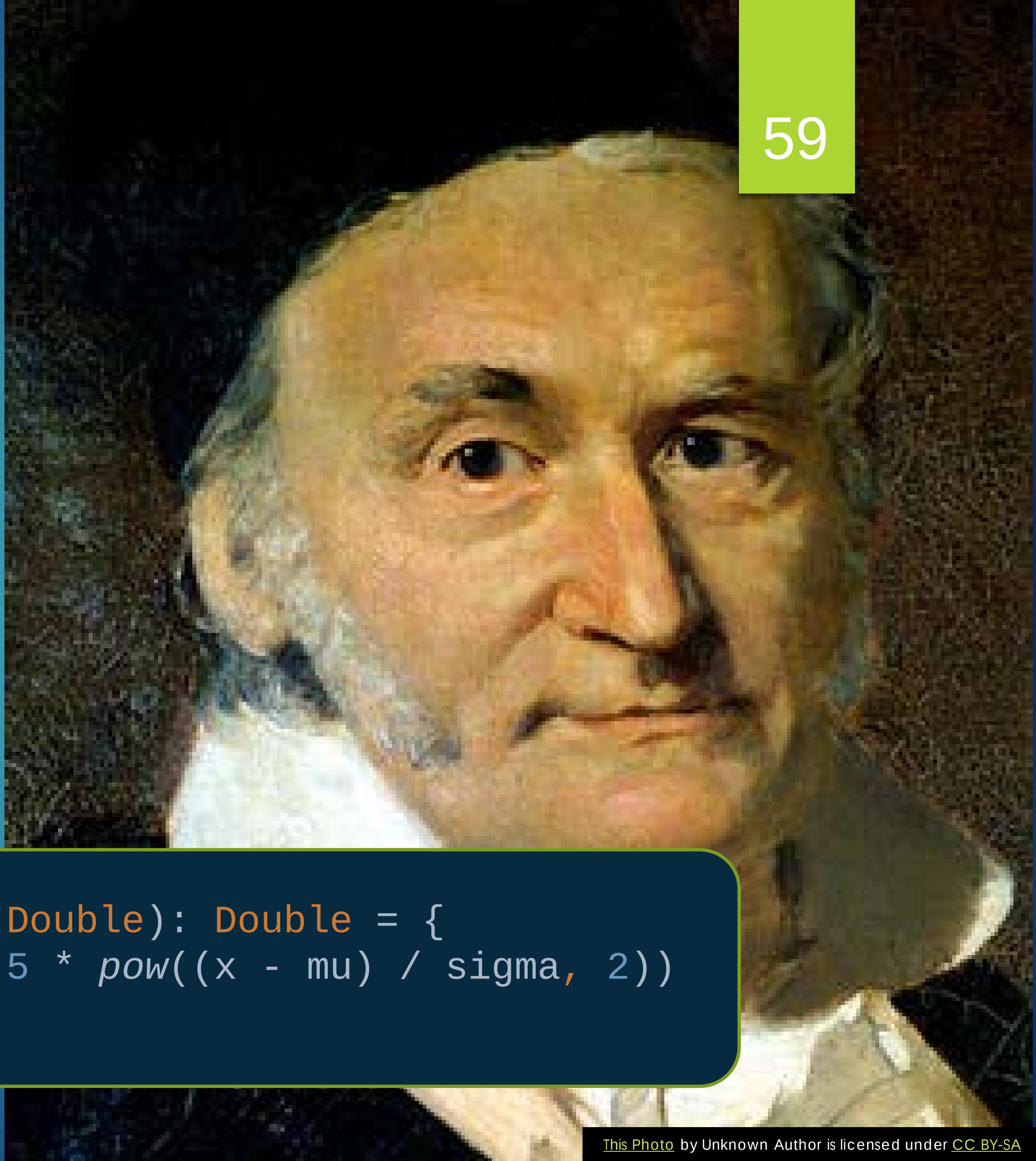


- Daily Loads

Gauss

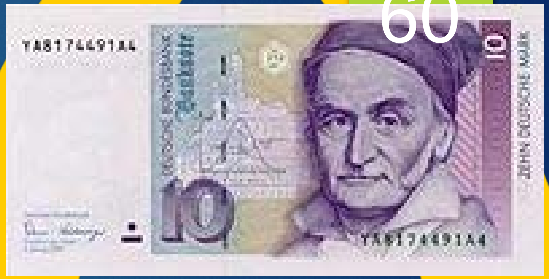
- $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

```
def gauss(sigma: Double, mu: Double)(x: Double): Double = {
  (1 / (sigma * sqrt(2 * Pi))) * exp(-0.5 * pow((x - mu) / sigma, 2))
}
```



M-Shape

This Photo by Unknown Author is licensed under CC BY-SA



- $f(x) = \frac{f_{\sigma_1 \mu_1}(x) + f_{\sigma_2 \mu_2}(x)}{2}$

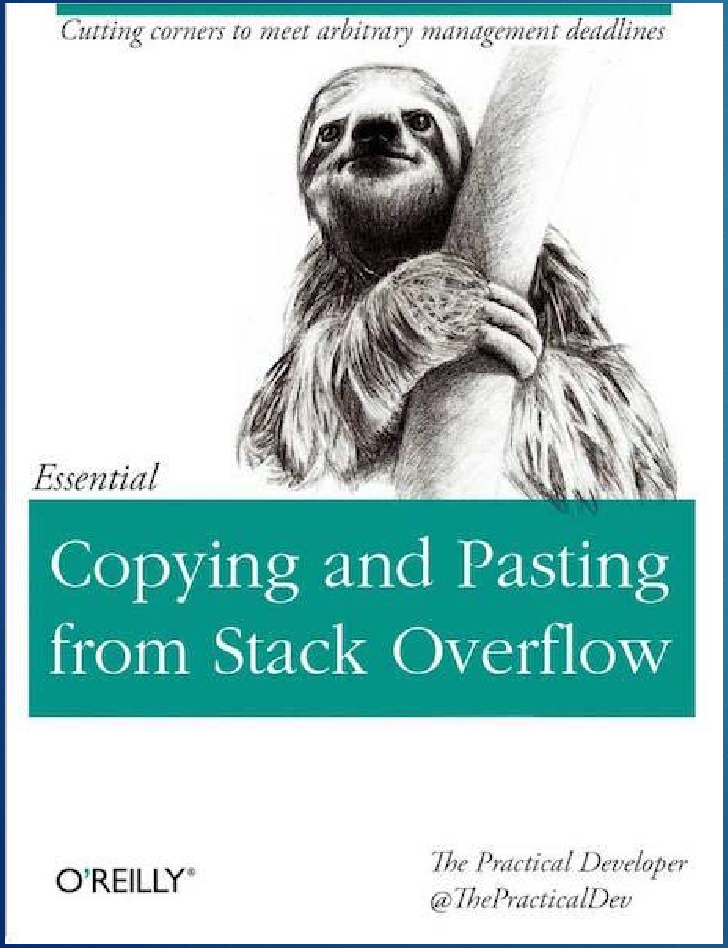
```
def mshape(sigma1: Double, mu1: Double,
           sigma2: Double, mu2: Double)(x: Double): Double = {
  (gauss(sigma1, mu1)(x) + gauss(sigma2, mu2)(x)) / 2
}
```

Binomial

- $B(k|p,n) = \binom{n}{k} p^k (1-p)^{n-k}$
- $B(k|p,n) = \frac{n!}{(n-k)!k!}\left(\frac{np}{n}\right)^k \left(1-\frac{np}{n}\right)^{n-k}$

```
def binomial(k: Int, p: Double)(n: Double): Double = {

  if (k > n) 0
  else ((n.toInt !) / ((k !) * (n.toInt - k) !))
       * pow(p, k)
       * pow(1 - p, n - k)
}
```

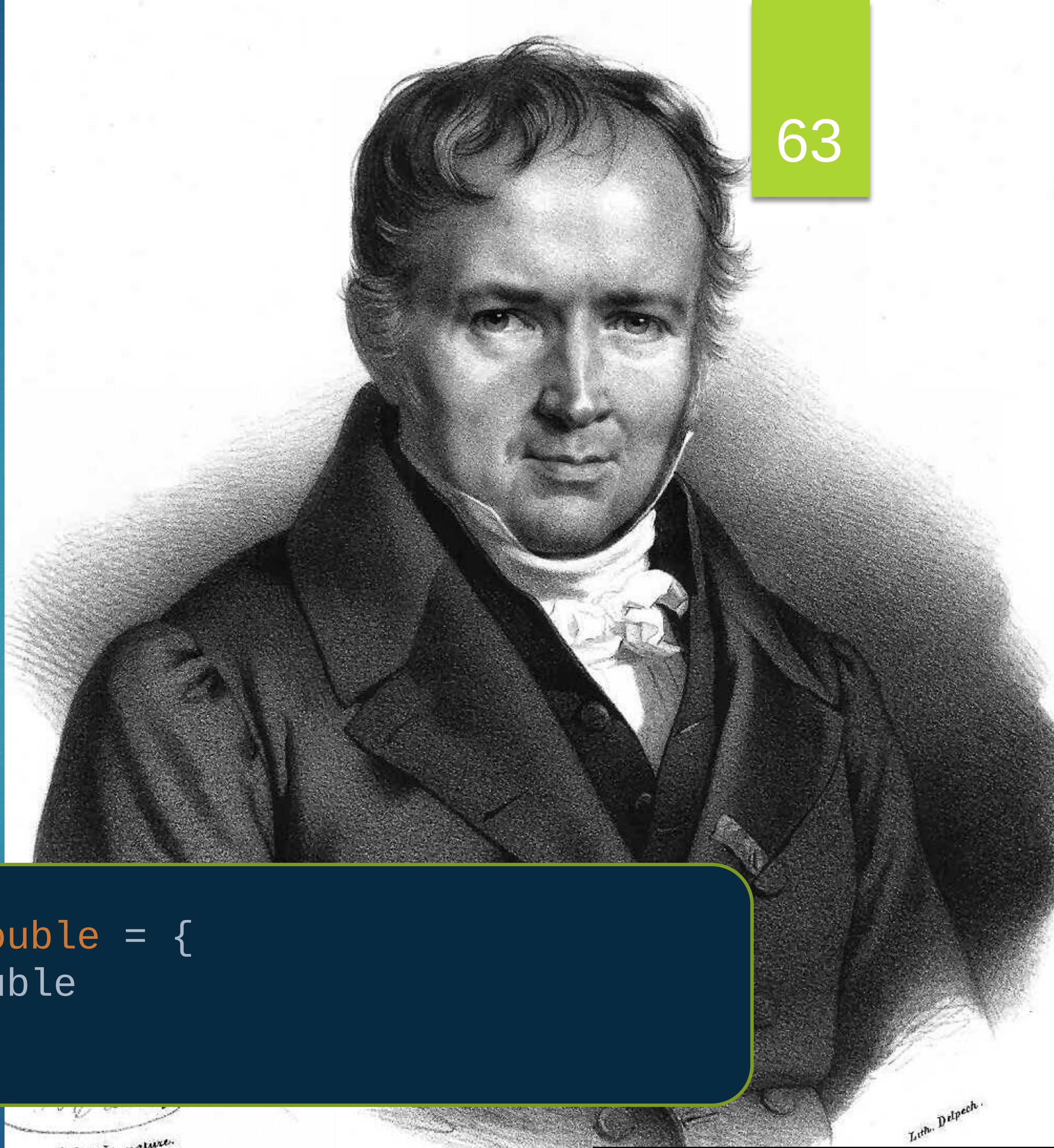
Wait!
Factorial ? => ! … !?!


Cutting corners to meet arbitrary management deadlines
Essential
Copying and Pasting from Stack Overflow
O'REILLY®
The Practical Developer
@ThePracticalDev

```scala
import scala.language.{implicitConversions, postfixOps}

…

def fac(n: Int): Int = (1 /: (1 to n)) (_ * _)


private implicit def factorial(n: Int) = new {
  def ! : Int = fac(n)
}
```

Poisson

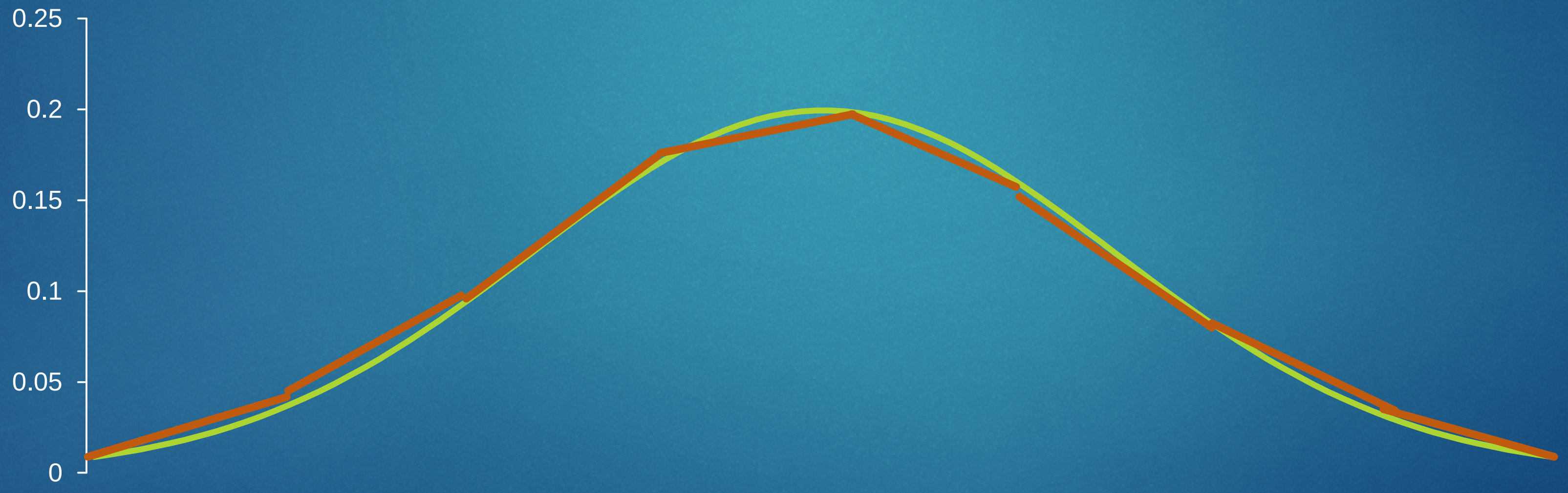
- $P_\lambda(k) = \frac{\lambda^k}{k!} e^{-\lambda}$



```
def poisson(l : Double)(n : Double) : Double = {
  exp(-l) * pow(l, n) / (n.toInt!).toDouble
}
```

Generating the Injection Steps

- Approximation of a continuous function through a series of linear functions (ramps)



Generating the Injection Steps

```
def f()= _ => List[InjectionStep]
```

Generating the Injection Steps

```
def continuousUserRate(duration: FiniteDuration,
                       totalUsers: Int,
                       distrFun: (Int) => (Double => Double),
                       stepFun: (Duration) => (Int) = (d) => d.toMinutes.toInt):
List[InjectionStep] = {
  val steps = stepFun(duration)
  val stepDuration = duration / steps
  def fun(x: Double) : Double =
       totalUsers * distrFun(steps)(x) / stepDuration.toSeconds
  List.range(0, steps).map(
       step => rampUsersPerSec(fun(step)) to fun(step + 1) during stepDuration)
}
```

Generating the Injection Steps

```
def continuousUserRate(duration: FiniteDuration,
                       totalUsers: Int,
                       distrFun: (Int) => (Double => Double),
                       stepFun: (Duration) => (Int) = (d) => d.toMinutes.toInt):
List[InjectionStep] = {
  val steps = stepFun(duration)
  val stepDuration = duration / steps
  def fun(x: Double) : Double =
       totalUsers * distrFun(steps)(x) / stepDuration.toSeconds
  List.range(0, steps).map(
       step => rampUsersPerSec(fun(step)) to fun(step + 1) during stepDuration)
}
```

Number of Injection Steps

Generating the Injection Steps

```
def continuousUserRate(duration: FiniteDuration,
                       totalUsers: Int,
                       distrFun: (Int) => (Double => Double),
                       stepFun: (Duration) => (Int) = (d) => d.toMinutes.toInt):
List[InjectionStep] = {
  val steps = stepFun(duratio
  val stepDuration = duration / steps
  def fun(x: Double) : Double =
      totalUsers * distrFun(steps)(x) / stepDuration.toSeconds
  List.range(0, steps).map(
      step => rampUsersPerSec(fun(step)) to fun(step + 1) during stepDuration)
}
```

Duration of a single step (1 minute)

Generating the Injection Steps

```
def continuousUserRate(duration: FiniteDuration,
                       totalUsers: Int,
                       distrFun: (Int) => (Double => Double),
                       stepFun: (Duration) =>                            nt):
List[InjectionStep] = {
  val steps = stepFun(duration)
  val stepDuration = duration / steps
  def fun(x: Double) : Double =
      totalUsers * distrFun(steps)(x) / stepDuration.toSeconds
  List.range(0, steps).map(
      step => rampUsersPerSec(fun(step)) to fun(step + 1) during stepDuration)
}
```

Calculate the user/sec rate for a specific step using the distribution function

Calculate the linear change in user/sec rate between two steps

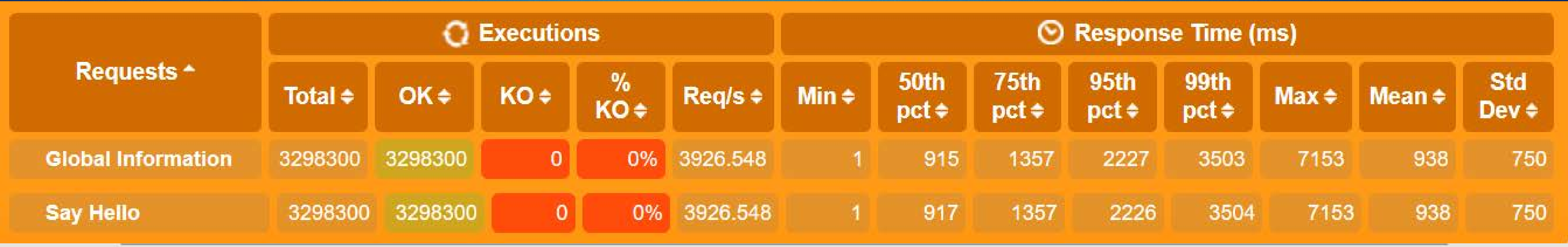
Generating the Injection Steps

```
 def gaussDistr(sigma: Double = 4, muPercent: Double = 0.5)
               (duration: FiniteDuration, totalUsers: Int) =
  continuousUserRate(duration, totalUsers,
                     steps => gauss(sigma, steps * muPercent))



setUp(
  ExampleScenario.helloWorld.inject(
    gaussDistr(2)(10 minutes, 30000)
  )
).protocols(httpServer)
```

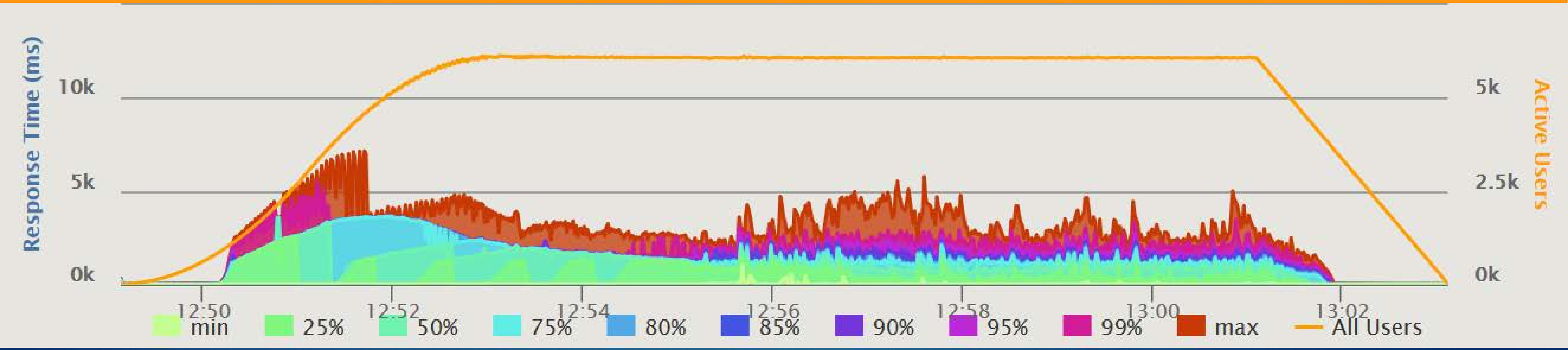
Examples

- Elastic System that dynamically scales up / down depending on load
- Parameters define the inertia of responding to load

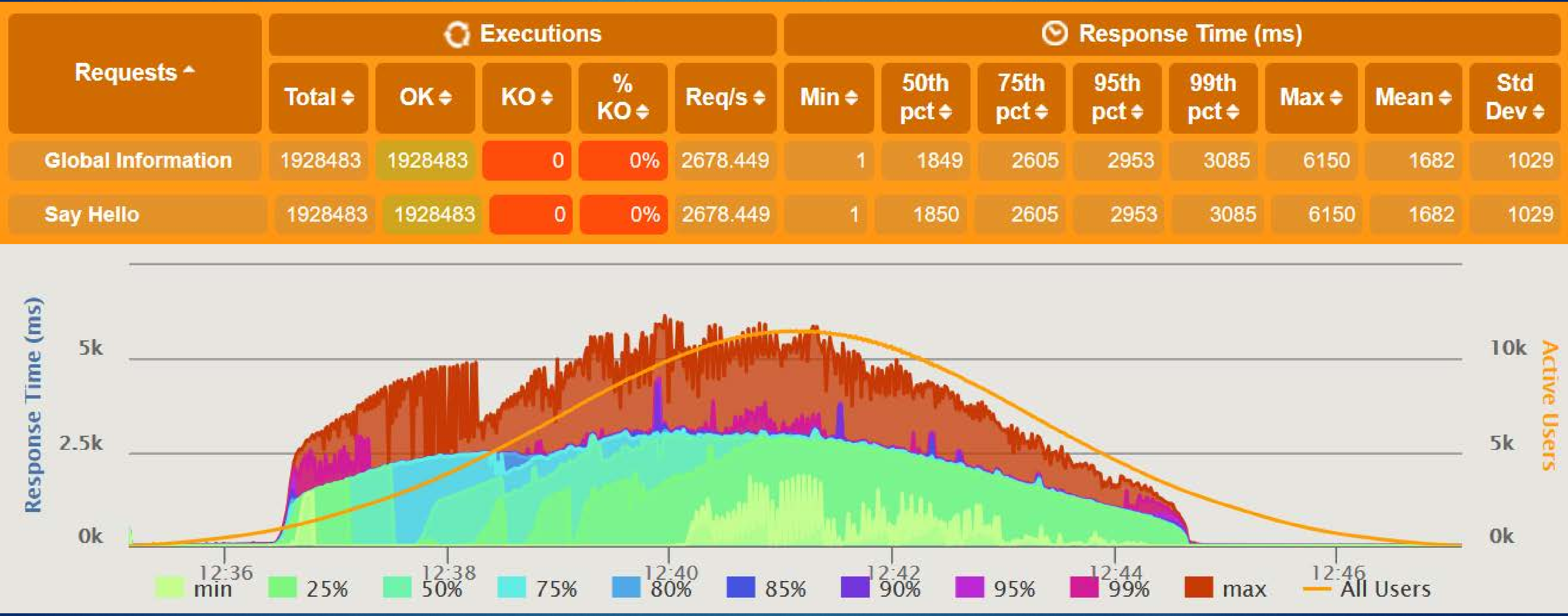
Constant Load



Requests ▲	Executions					Response Time (ms)							
	Total ♦	OK ♦	KO ♦	% KO ♦	Req/s ♦	Min ♦	50th pct ♦	75th pct ♦	95th pct ♦	99th pct ♦	Max ♦	Mean ♦	Std Dev ♦
Global Information	3298300	3298300	0	0%	3926.548	1	915	1357	2227	3503	7153	938	750
Say Hello	3298300	3298300	0	0%	3926.548	1	917	1357	2226	3504	7153	938	750

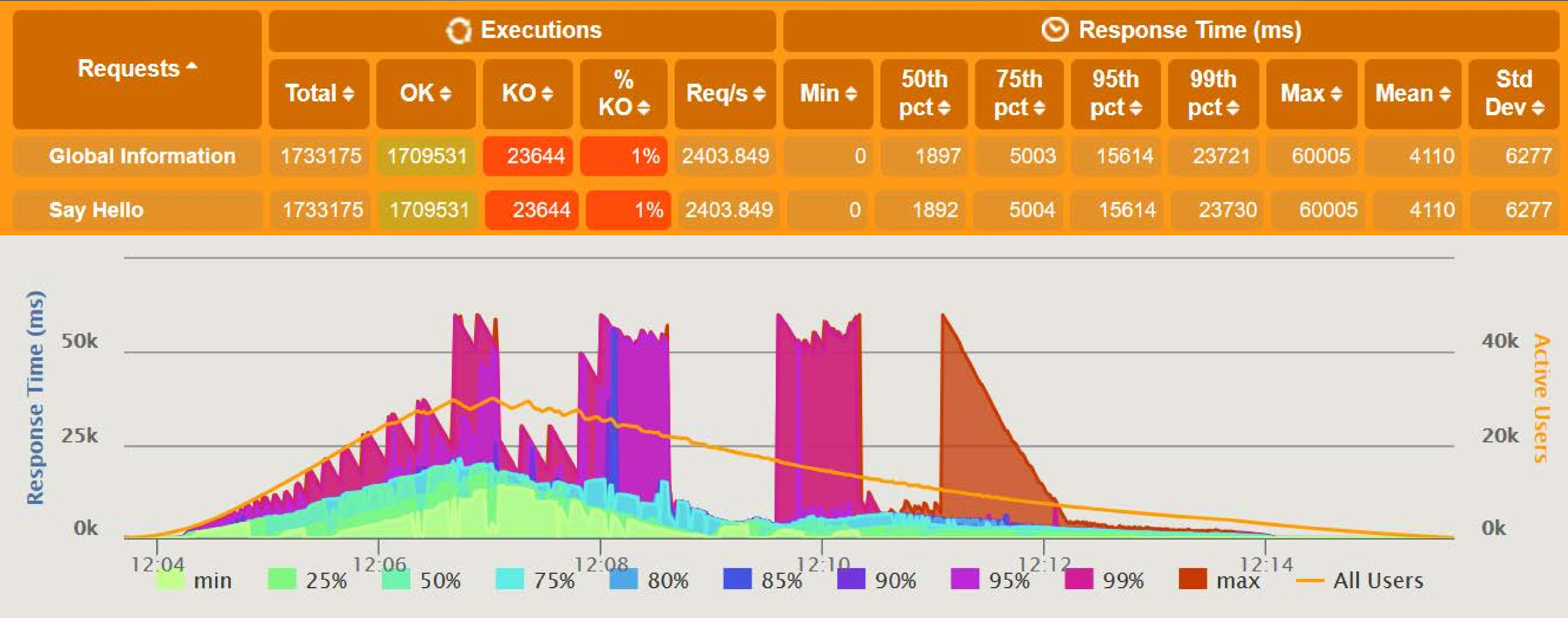


Gauss Distribution



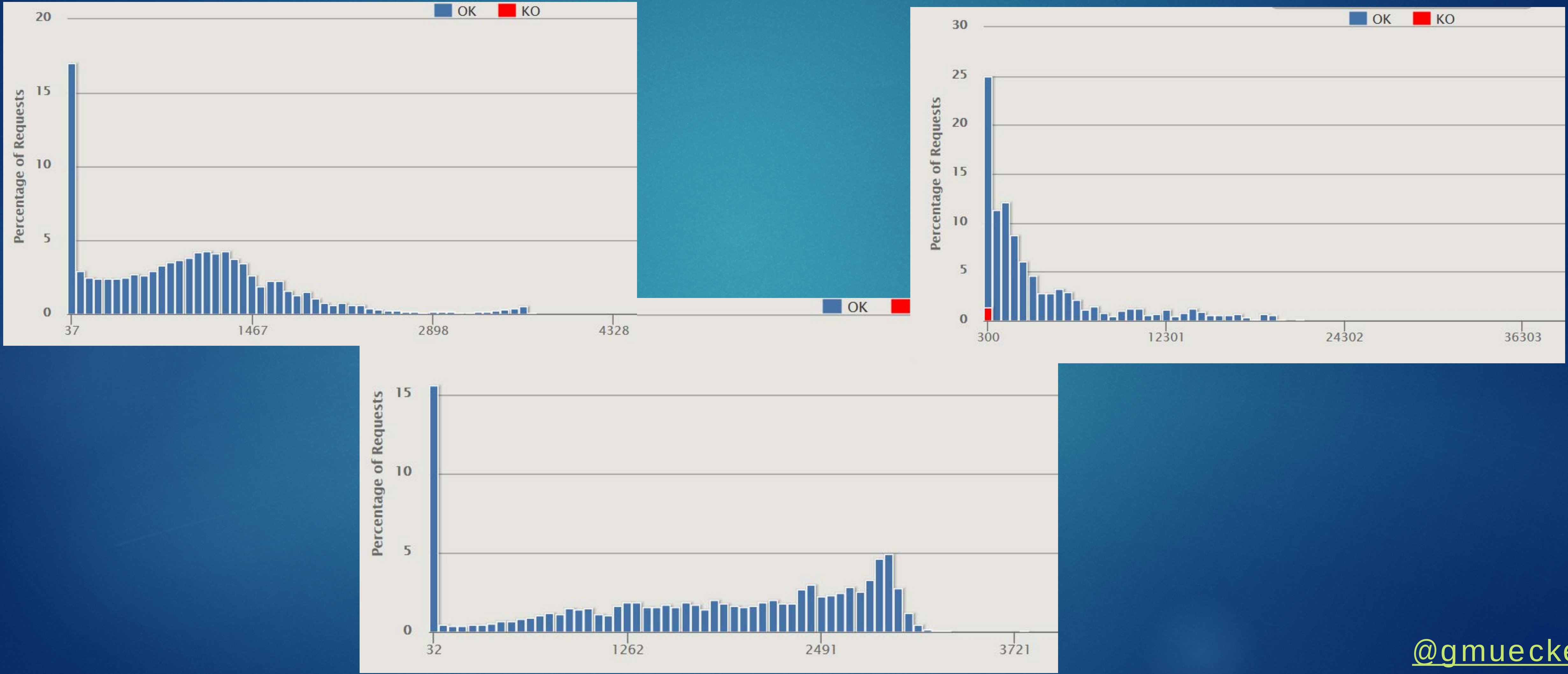
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	1928483	1928483	0	0%	2678.449	1	1849	2605	2953	3085	6150	1682	1029
Say Hello	1928483	1928483	0	0%	2678.449	1	1850	2605	2953	3085	6150	1682	1029

Binomial Distribution



Requests ▴	Executions					Response Time (ms)							
	Total ⬍	OK ⬍	KO ⬍	% KO ⬍	Req/s ⬍	Min ⬍	50th pct ⬍	75th pct ⬍	95th pct ⬍	99th pct ⬍	Max ⬍	Mean ⬍	Std Dev ⬍
Global Information	1733175	1709531	23644	1%	2403.849	0	1897	5003	15614	23721	60005	4110	6277
Say Hello	1733175	1709531	23644	1%	2403.849	0	1892	5004	15614	23730	60005	4110	6277

Response Time Historgrams



Functional Load Testing

- Uses Statistical Distribution Functions to Shape the Load Profile
- Generate more realistic load patterns
  - Response Time Percentiles better reflect User Experience
  - Test dynamic behavior of the system
  - Facilitates more realistic Sizing & Capacity Forecast with respect to User Experience
- Does not replace other forms of load testing
  - Is a supplement that provides additional value in certain contexts

- Example source code available at
- https://github.com/gmuecke/flt-example

Thank you!

QUESTIONS & FEEDBACK APPRECIATED!