Queues and You:
System Performance and Queueing Theory

Jeff Poole
Vivint Smart Home / NRG Energy

AKA

How an engineer can learn something useful in business school

Who am I?

Jeff Poole
Twitter/X @_JeffPoole
LinkedIn linkedin.com/in/jeffpoole0

Background:

- hardware (FPGAs)
- software (Java, Python, Go)
- operations (Kubernetes, datacenters, networking)

Hobbies:

- cycling
- skiing
- collecting certifications
- proposing talks on topics I don't understand

I currently manage teams that write our core back-end software functionality and manage our infrastructure at Vivint.

Why The Focus?

Queueing Theory

Queueing Theory

Queueing theory is all about the behavior of work getting done, when work may have to wait (be queued) before getting processed.

Queueing Theory

Queueing Theory

Queueing Theory

'TSA' by Thomas Hawk

RabbitMQ™

NATS

kafka

Queueing Theory

> **What Is Linux Load Average?**
> Linux load average is a metric that shows the number of tasks currently executed by the CPU and tasks waiting in the **queue**. [1]

1. from https://phoenixnap.com/kb/linux-average-load

Queueing Theory

`iostat` output

```
Device  r/s     w/s     rkB/s     wkB/s
rrqm/s   wrqm/s  %rrqm  %wrqm r_await
w_await aqu-sz rareq-sz wareq-sz  svctm
%util
sda      1.84   16.69     45.12    230.12
1.08     21.73  37.02  56.55   26.12
3.06   0.02     24.52    13.78   0.80
1.48
sdb      167.74   32.02   2061.01
575.14     0.39     0.90   0.23   2.73
0.14     1.58   0.05    12.29    17.96
0.12   2.47
```

***rqm/s** (and **rrqm/s** and **wrqm/s**)*
The number of I/O requests merged per second that were queued to the device.

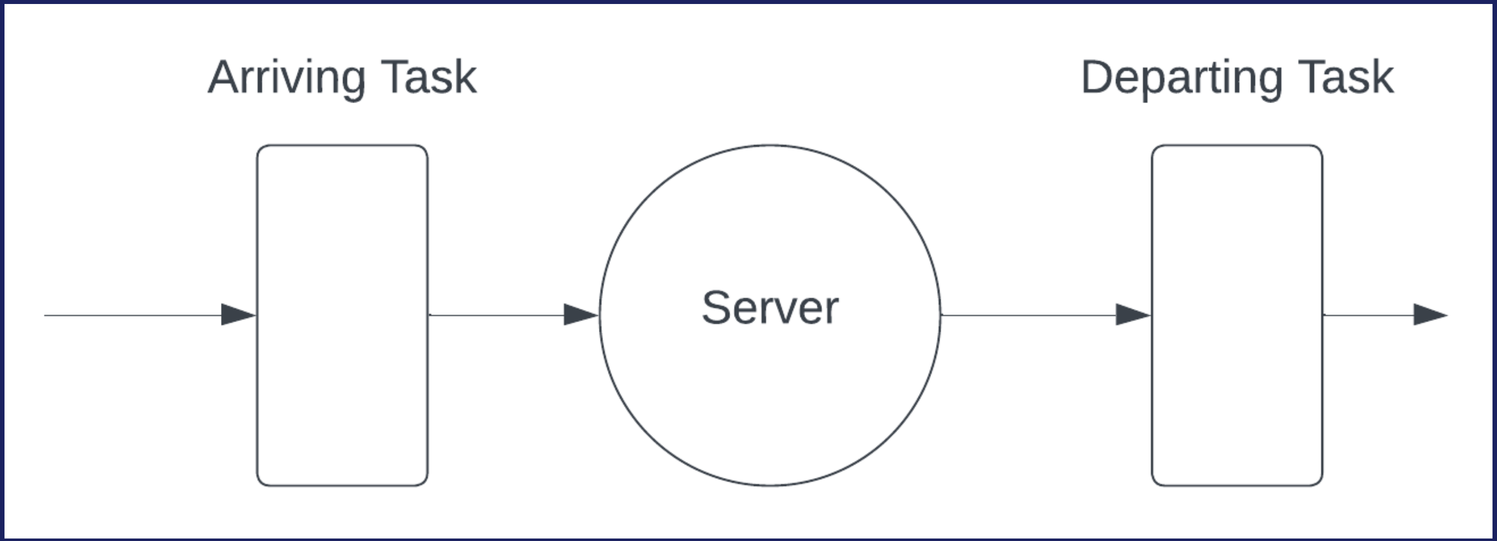
***await** (and **r_await** and **w_await**)*
The average time (in milliseconds) for I/O requests issued to the device to be served. This includes the time spent by the requests in queue and the time spent servicing them.

aqu-sz
The average queue length of the requests that were issued to the device.

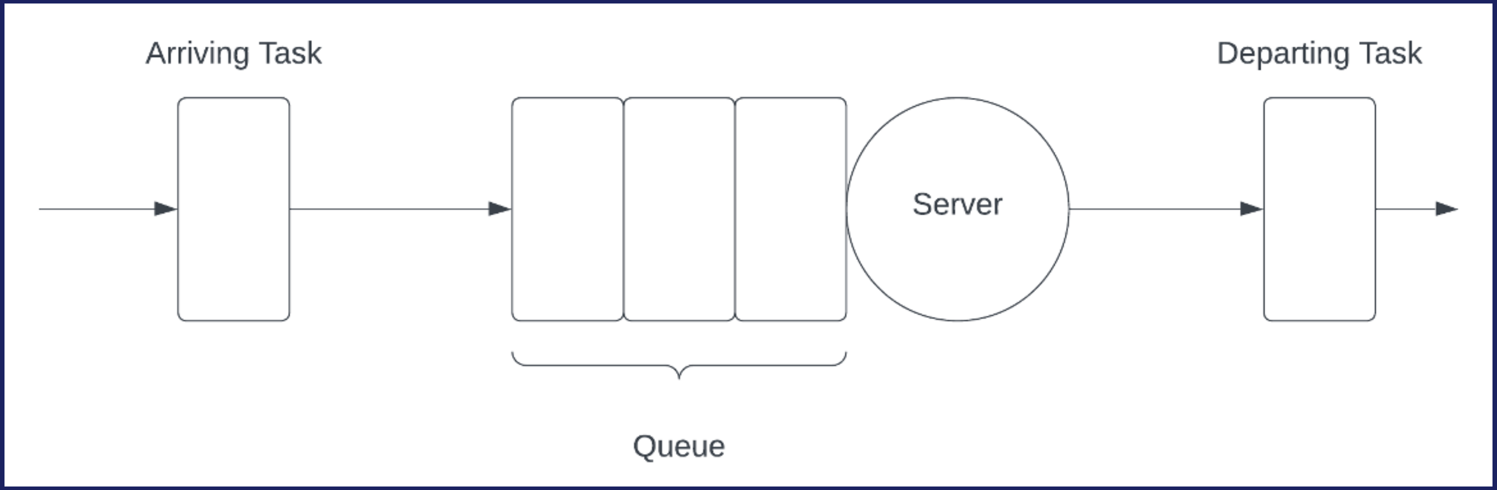
%util
Percentage of elapsed time during which I/O requests were issued to the device (bandwidth utilization for the device). Device saturation occurs when this value is close to 100% for devices serving requests serially.

Queueing Theory Basics

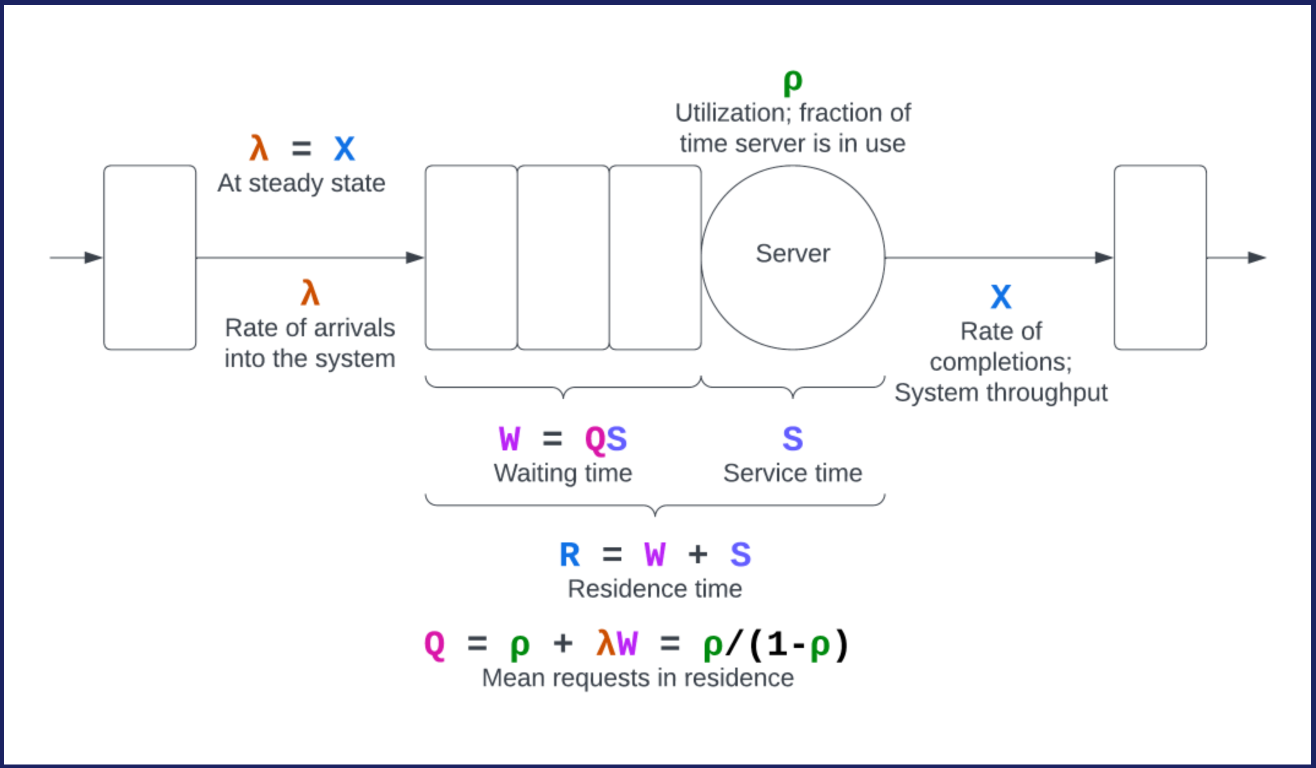
Queueing Theory Basics



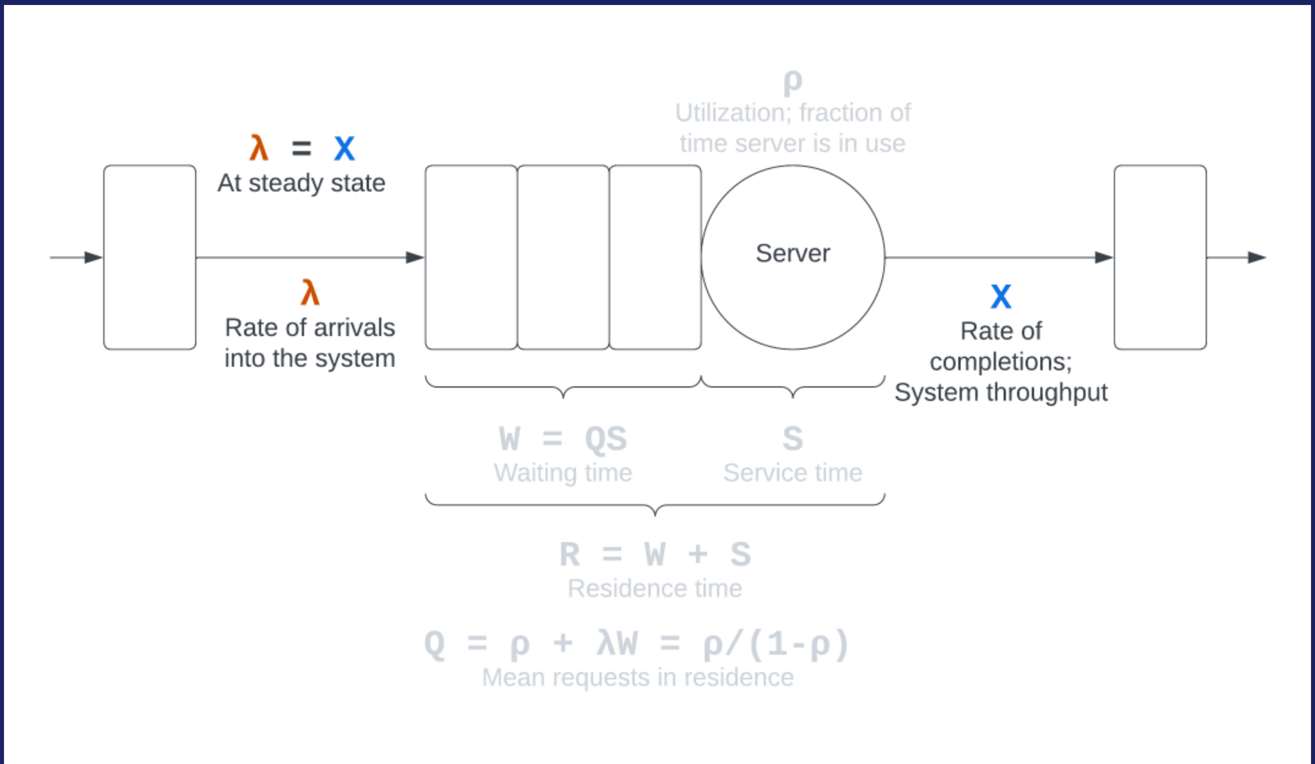
Queueing Theory Basics



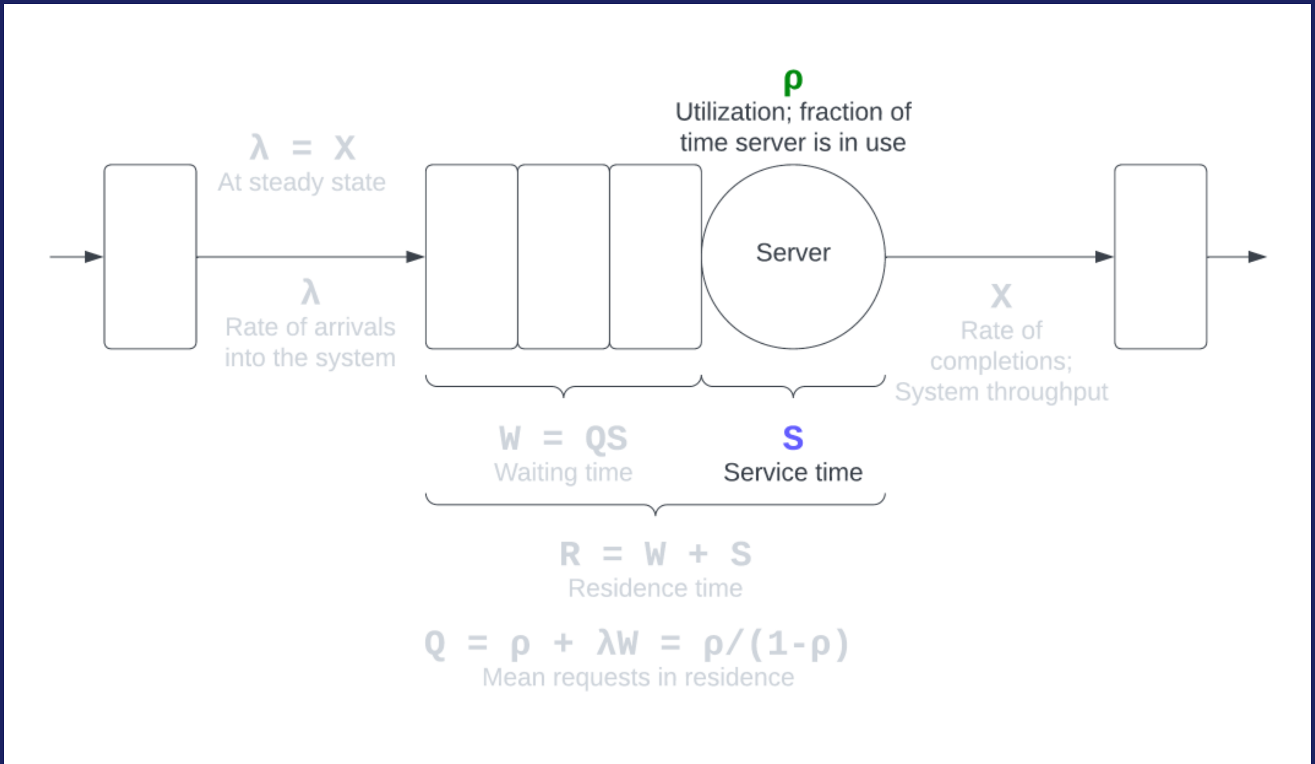
Queueing Theory Basics



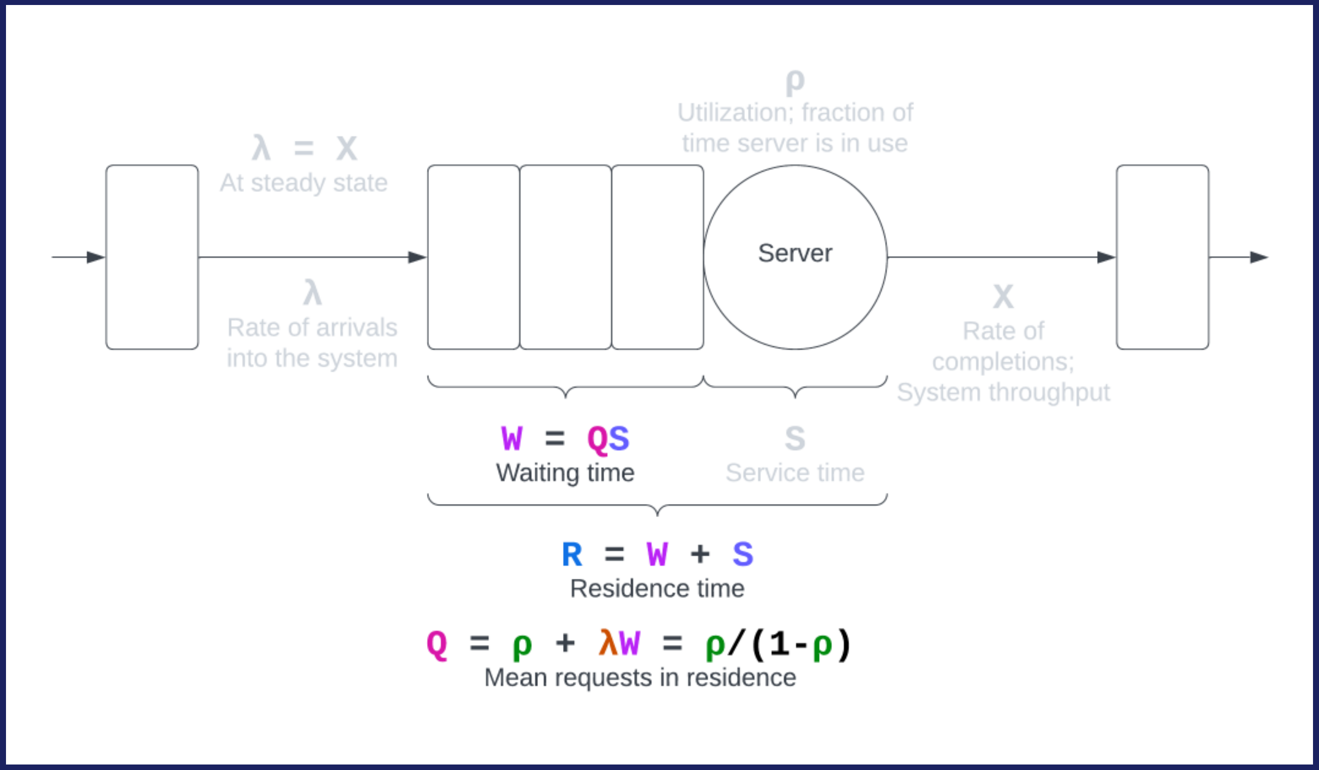
Queueing Theory Basics



Queueing Theory Basics



Queueing Theory Basics



Little's Law - two forms

If you want to find the relationship between queue length and residence time:

$$\text{Average Queue Length} = \text{Average Arrival Rate} * \text{Average Residence Time}$$

$$Q = \lambda R$$

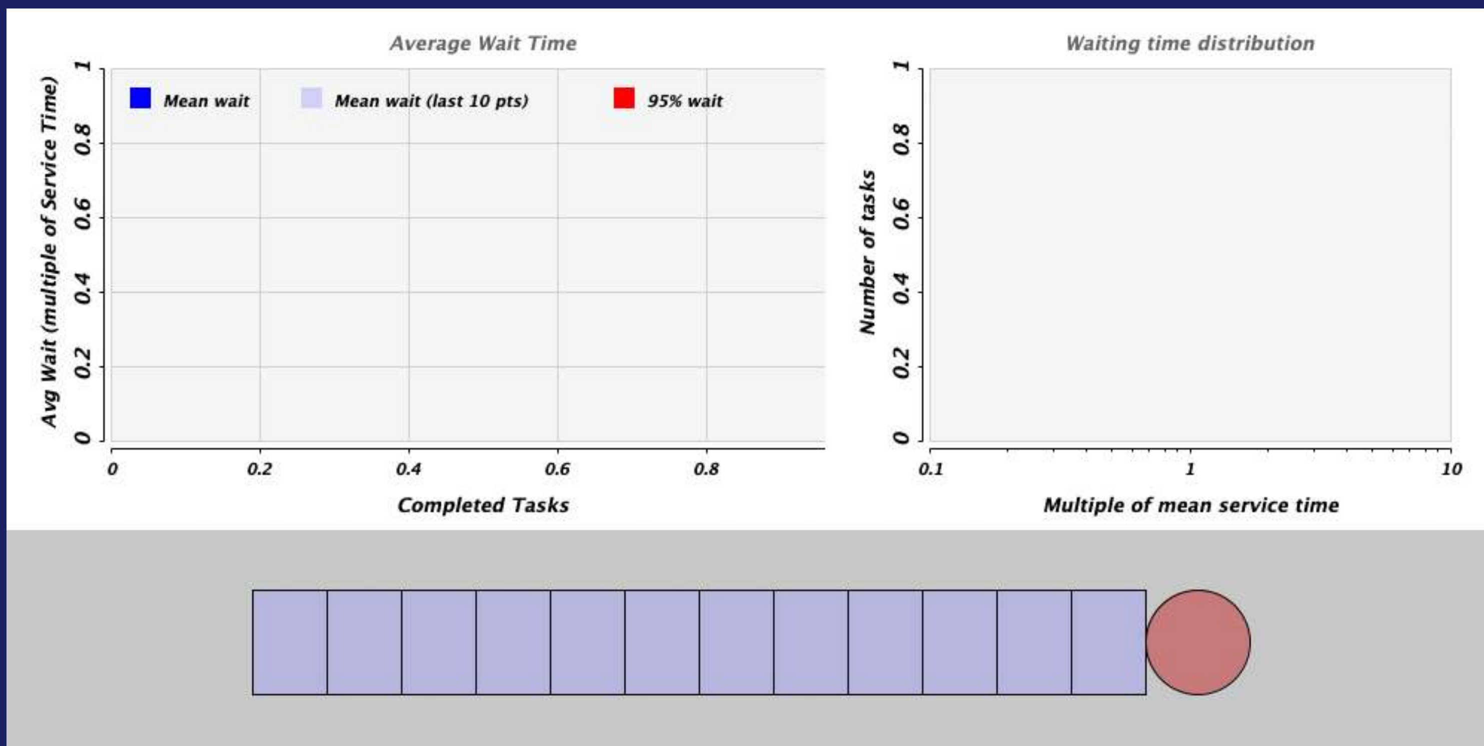
If you want to find the relationship between utilization and service time:

$$\text{Utilization} = \text{Average Arrival Rate} * \text{Average Service Time}$$

$$\rho = \lambda S$$

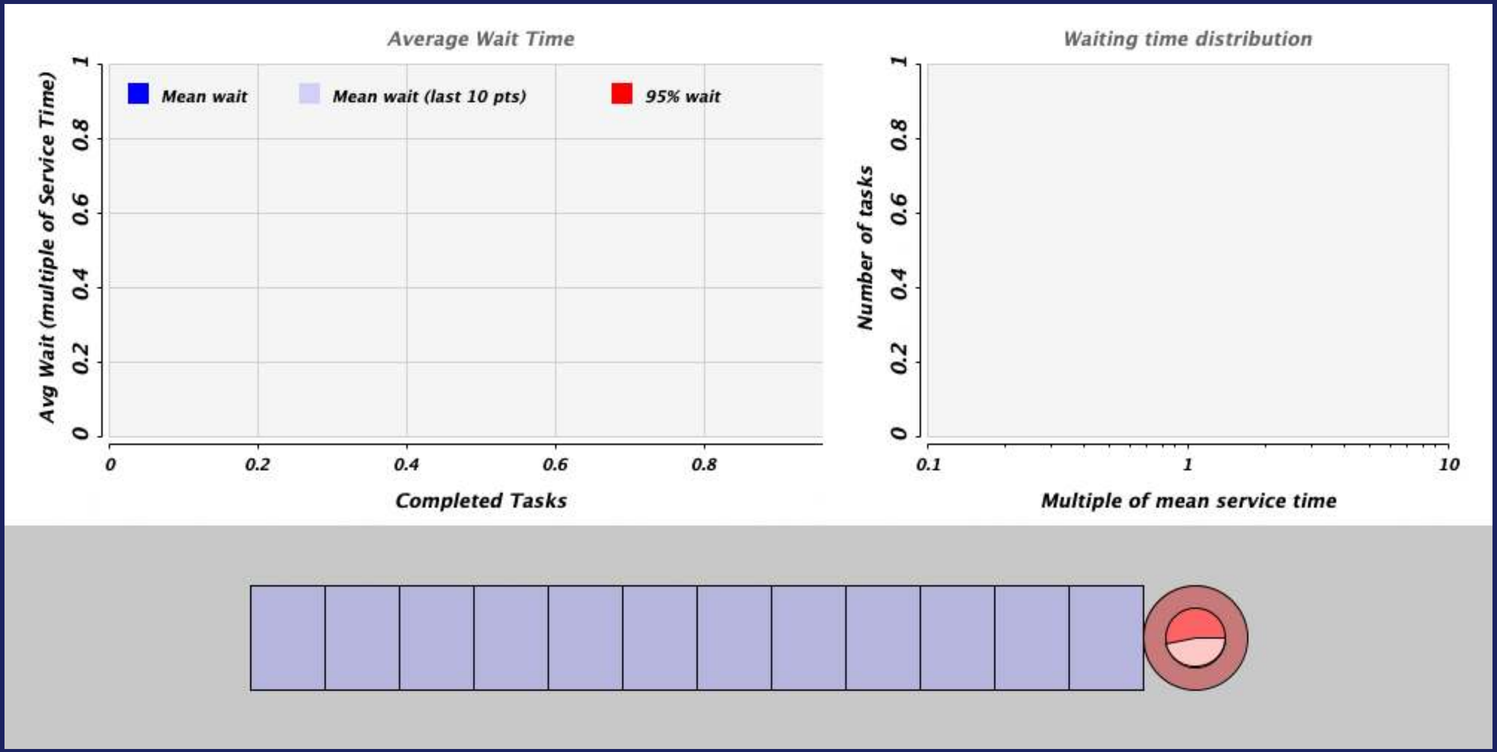
Capacity and Utilization

A Simple Model - 80% utilization



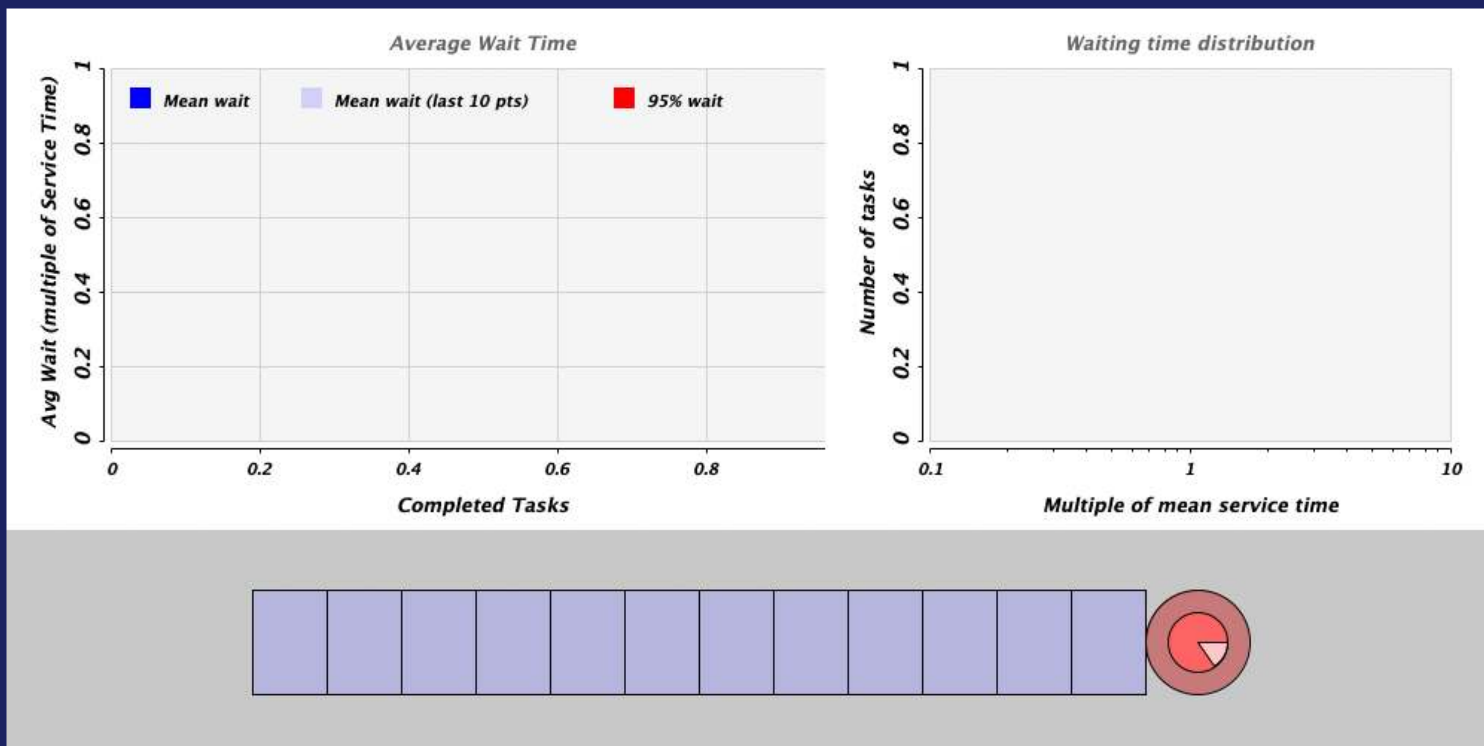
(link to gif)

Oh no, variability in arrivals!



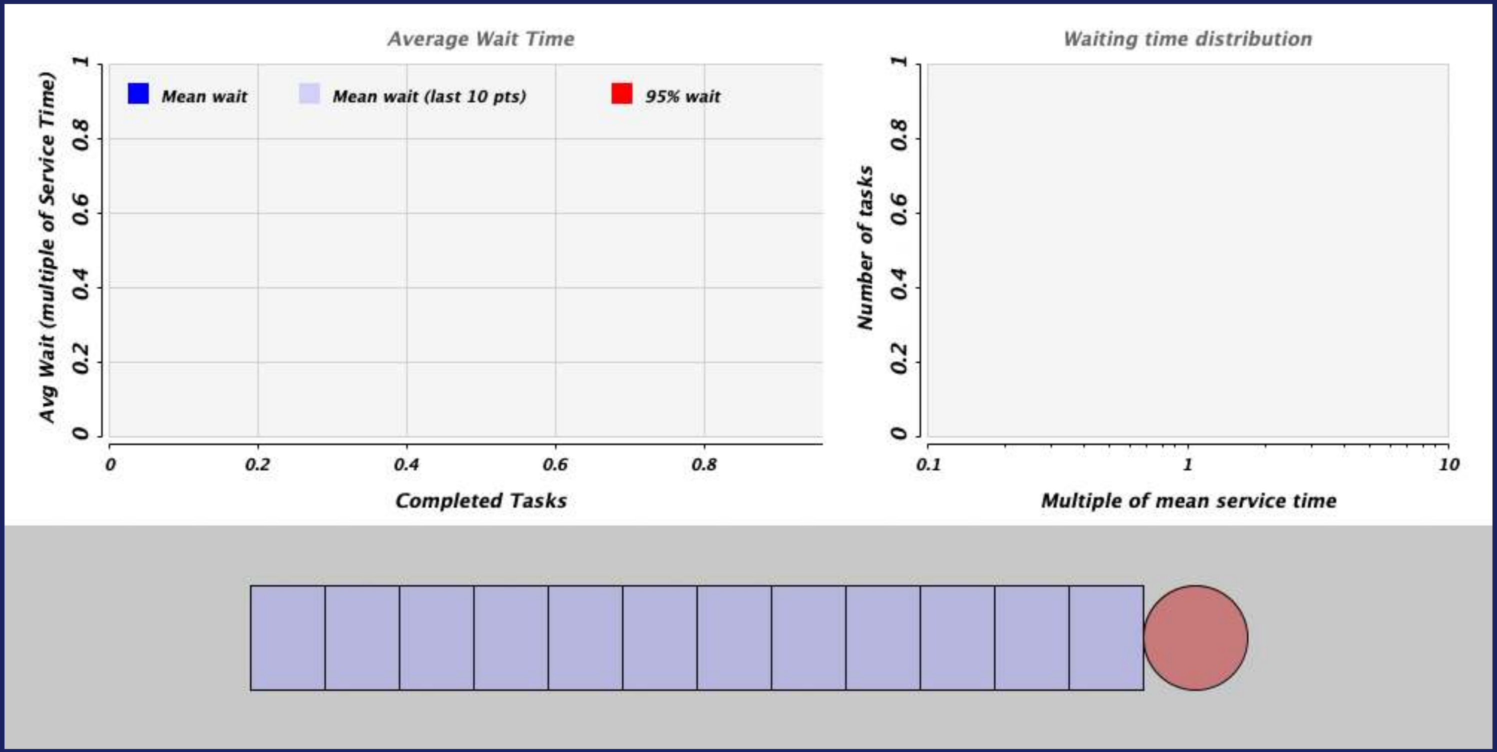
(link to gif)

Let's try variable service time!



(link to gif)

Variable service AND arrival time!



(link to gif)

Variability in either arrival time *OR* in service time can cause queueing in a system that has enough capacity for the work

...and most systems have variability in both.

Equation for time in queue for a single resource

$$\text{Time in Queue} = \text{Service Time} * \left(\frac{\text{Utilization}}{1 - \text{Utilization}}\right) * \left(\frac{\text{CV}_{\text{arrivals}}^2 + \text{CV}_{\text{service}}^2}{2}\right)$$

$$W = S * \left(\frac{\rho}{1-\rho}\right) * \left(\frac{CV_{arrival}^2 + CV_{service}^2}{2}\right)$$

Note: Coefficient of Variation (CV) of a distribution is the standard deviation divided by the mean ($CV = \sigma/\mu$).

Equation for time in queue for a single resource

$$\text{Time in Queue} = \text{Service Time} * \underbrace{\left(\frac{\text{Utilization}}{1 - \text{Utilization}}\right)}_{\text{How busy}} * \underbrace{\left(\frac{\text{CV}^2_{\text{arrivals}} + \text{CV}^2_{\text{service}}}{2}\right)}_{\text{How variable}}$$

$$W = S * \underbrace{\left(\frac{\rho}{1 - \rho}\right)}_{\text{How busy}} * \underbrace{\left(\frac{CV^2_{arrival} + CV^2_{service}}{2}\right)}_{\text{How variable}}$$

Note: Coefficient of Variation (CV) of a distribution is the standard deviation divided by the mean ($CV = \sigma/\mu$).

Equation for time in queue (NO variability)

$$\text{Time in Queue} = \text{Service Time} * \left(\frac{\text{Utilization}}{1 - \text{Utilization}}\right) * \left(\frac{0^2 + 0^2}{2}\right) = 0$$

$$W = S * \left(\frac{\rho}{1 - \rho}\right) * \left(\frac{0^2 + 0^2}{2}\right) = 0$$

No variability $\rightarrow CV = 0 \rightarrow \text{time in queue (W)} = 0$

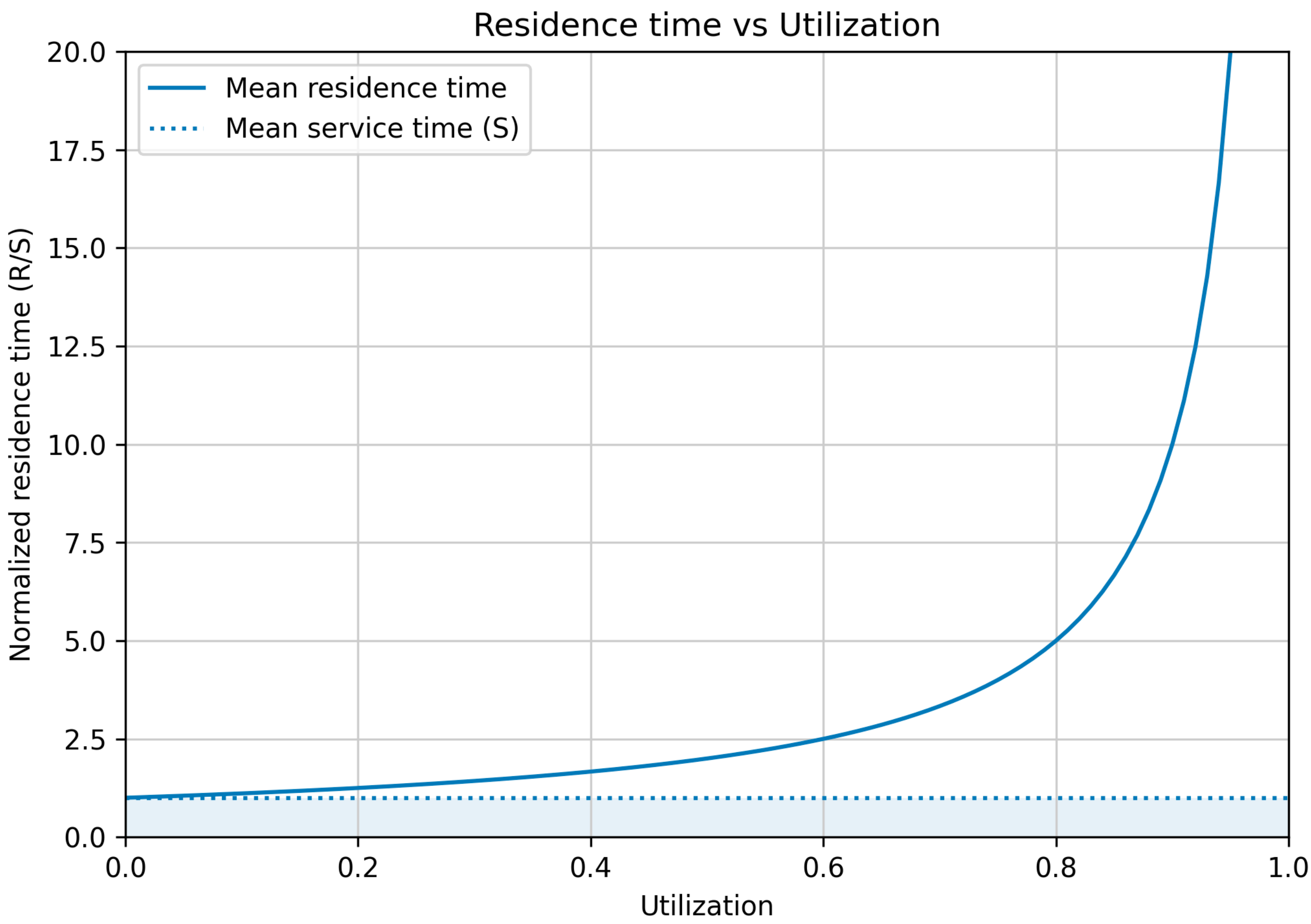
Equation for time in queue (exponential distributions)

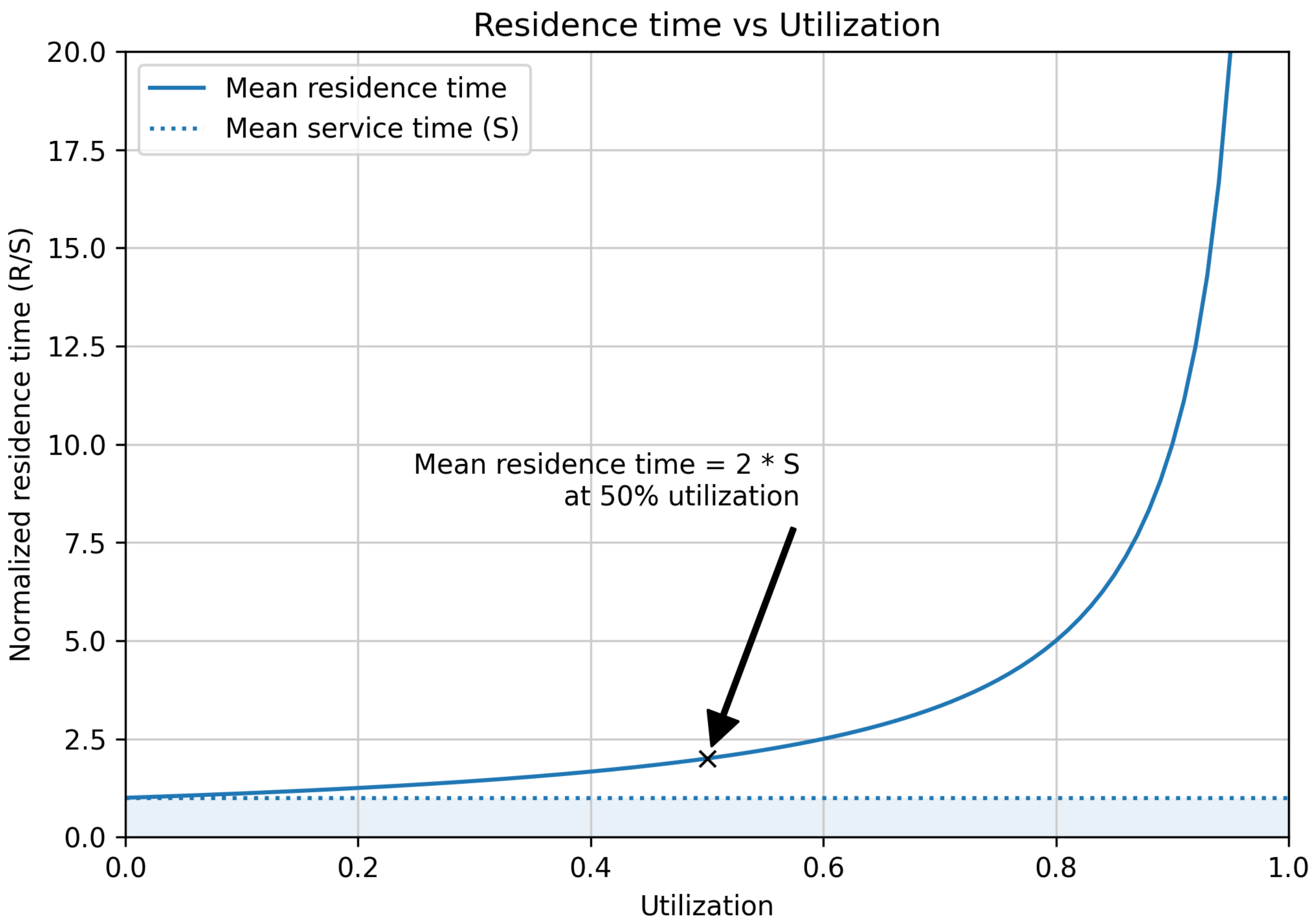
$$\text{Time in Queue} = \text{Service Time} * \left(\frac{\text{Utilization}}{1 - \text{Utilization}}\right)$$

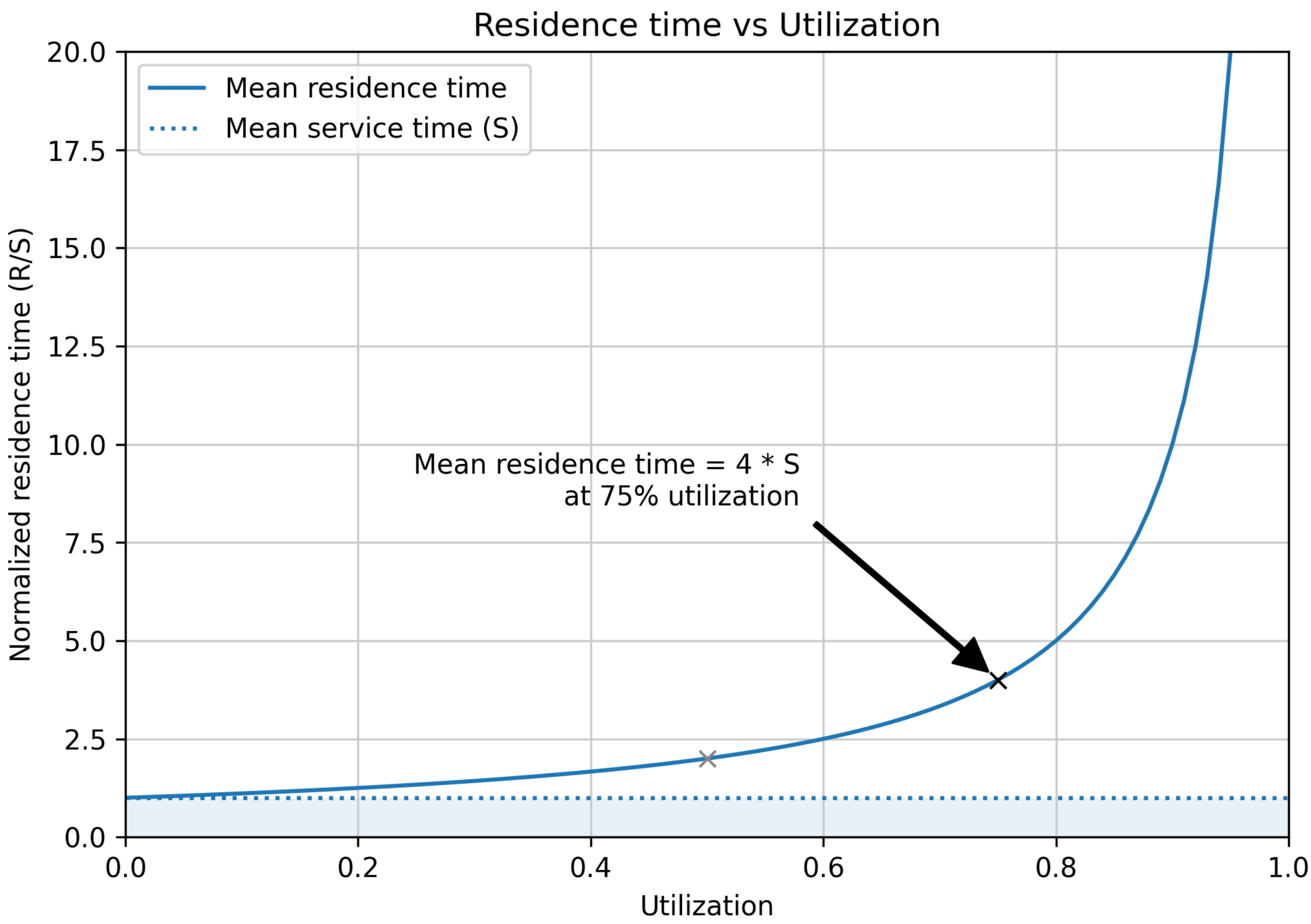
$$W = S * \left(\frac{\rho}{1 - \rho}\right)$$

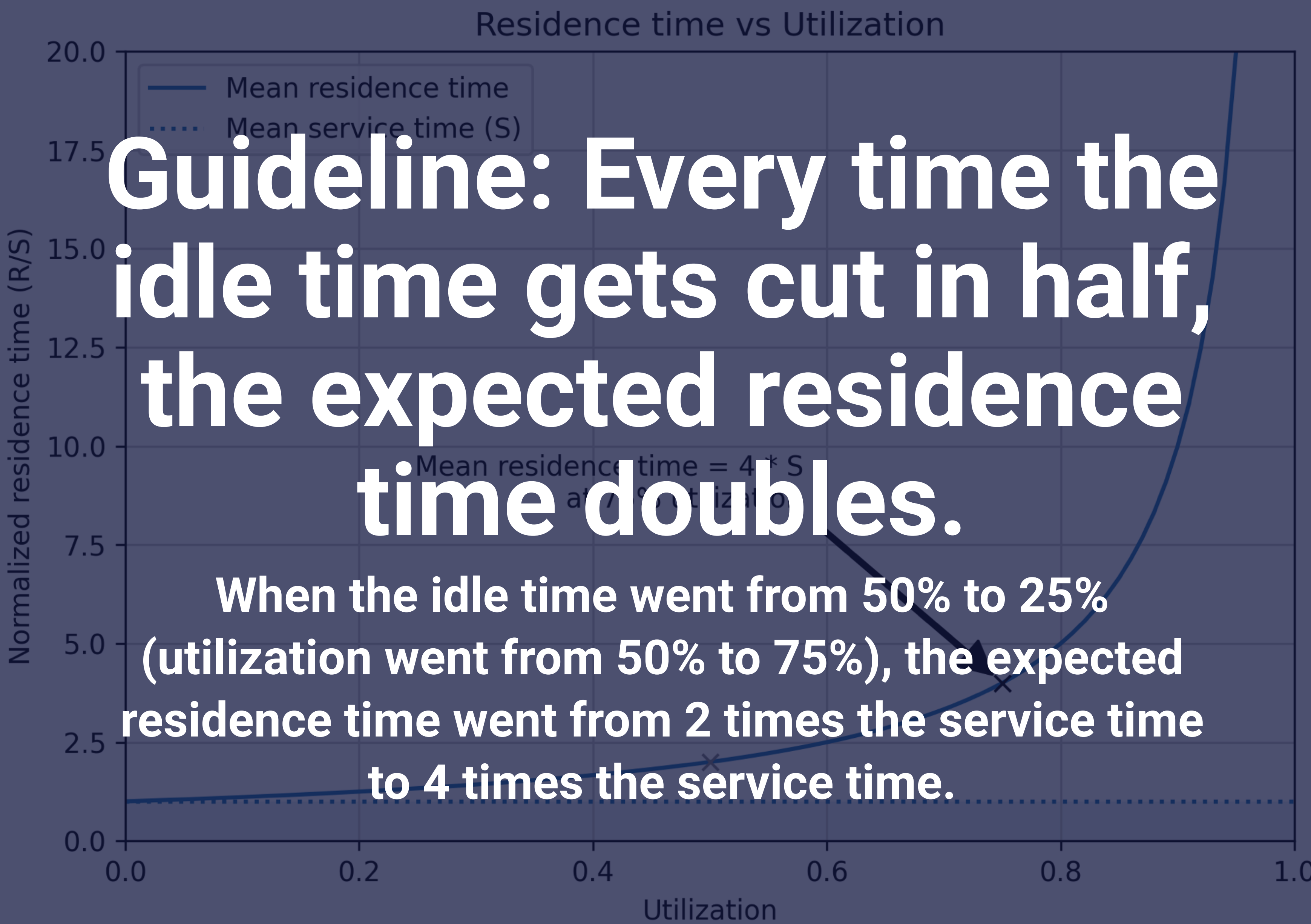
Exponential distribution $\rightarrow CV = 1 \rightarrow$
$\text{time in queue (W)} = S * \frac{\rho}{1-\rho}$

How does utilization affect latency?


Residence time vs Utilization
Mean residence time
Mean service time (S)
Normalized residence time (R/S)
Utilization
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0
0.2
0.4
0.6
0.8
1.0

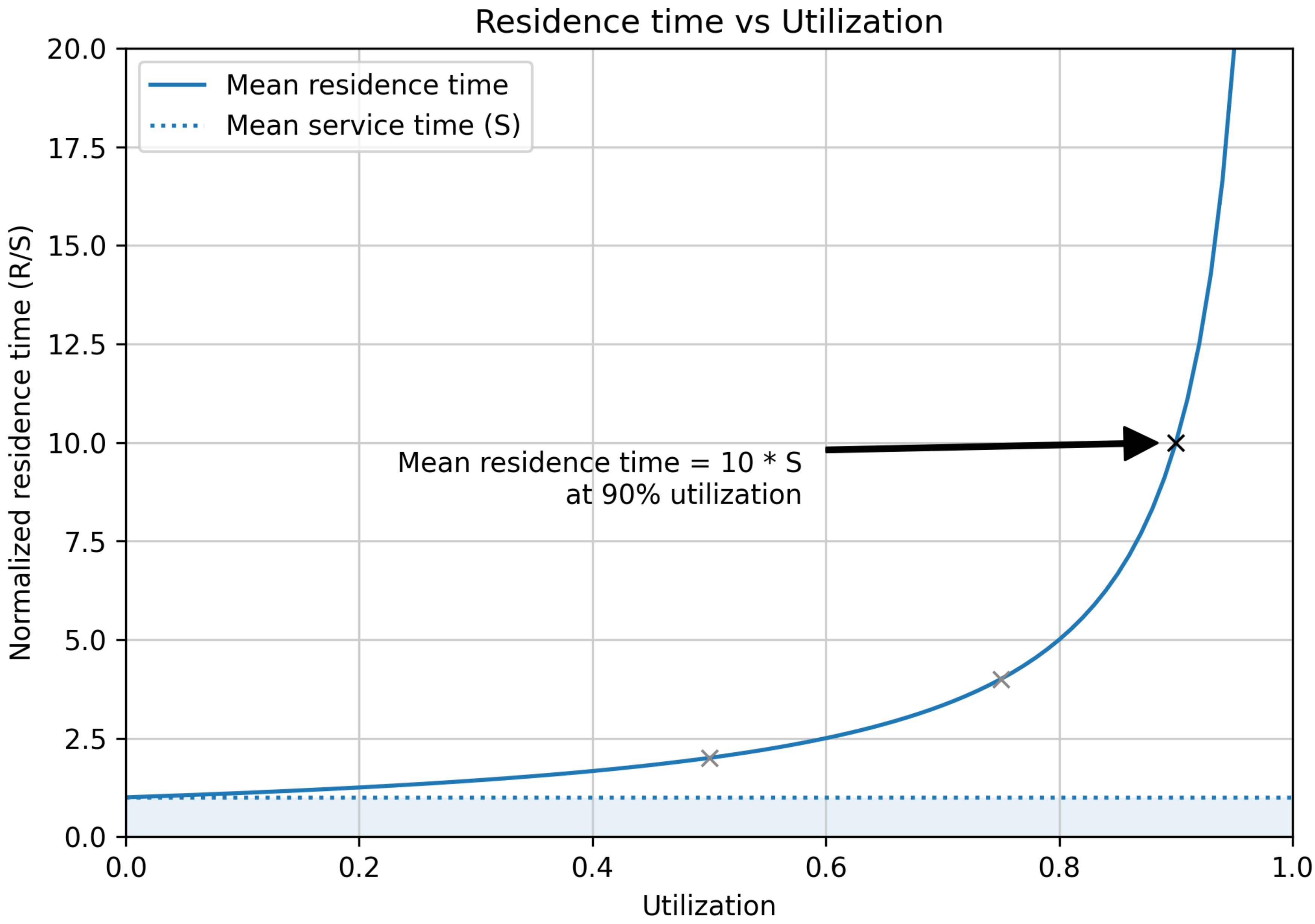

Residence time vs Utilization
Mean residence time
Mean service time (S)
Mean residence time = 2 * S
at 50% utilization
Normalized residence time (R/S)
Utilization
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

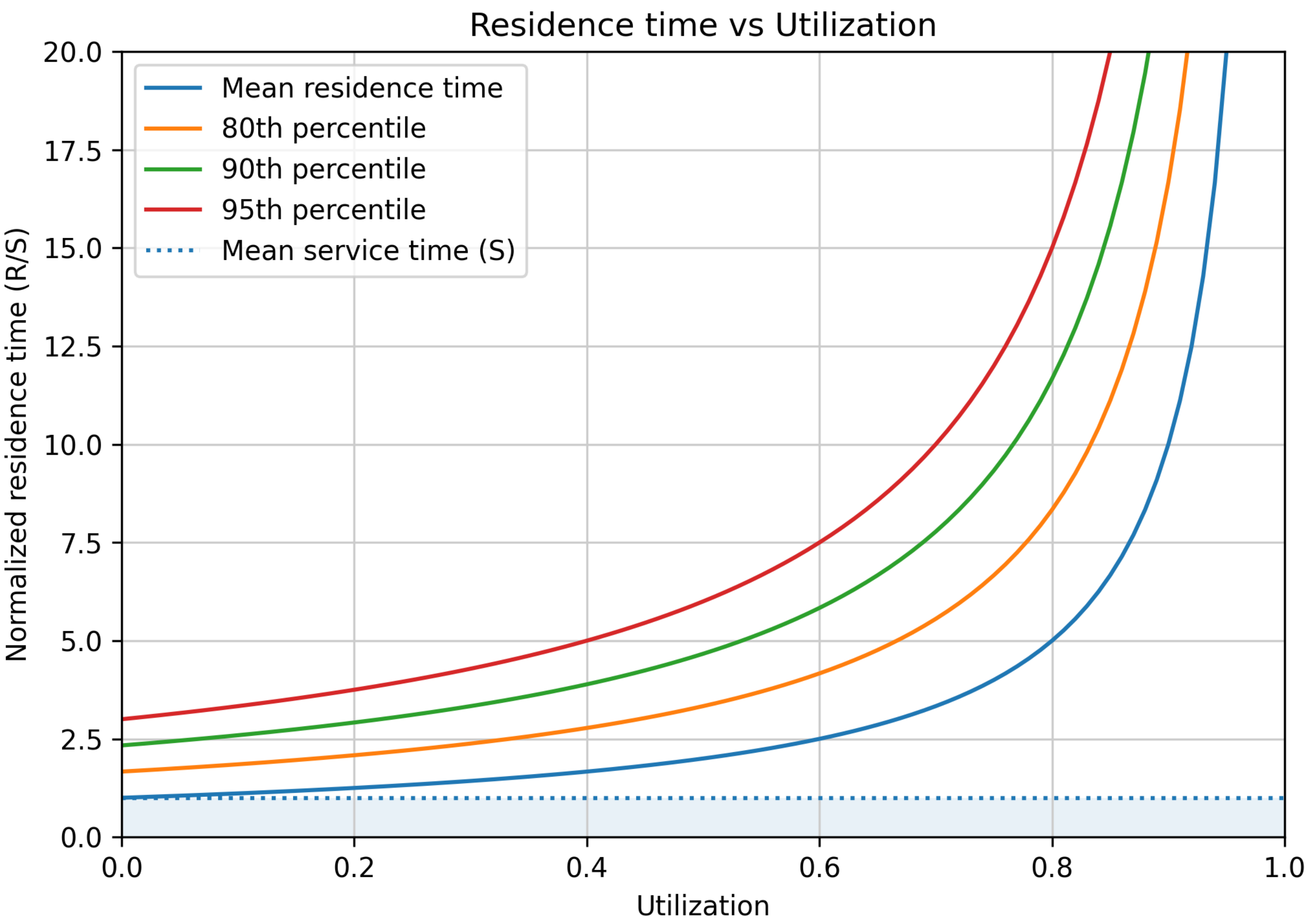

Residence time vs Utilization
Mean residence time
Mean service time (S)
Mean residence time = 4 * S
at 75% utilization
Normalized residence time (R/S)
Utilization
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0
0.2
0.4
0.6
0.8
1.0



Guideline: Every time the idle time gets cut in half, the expected residence time doubles.

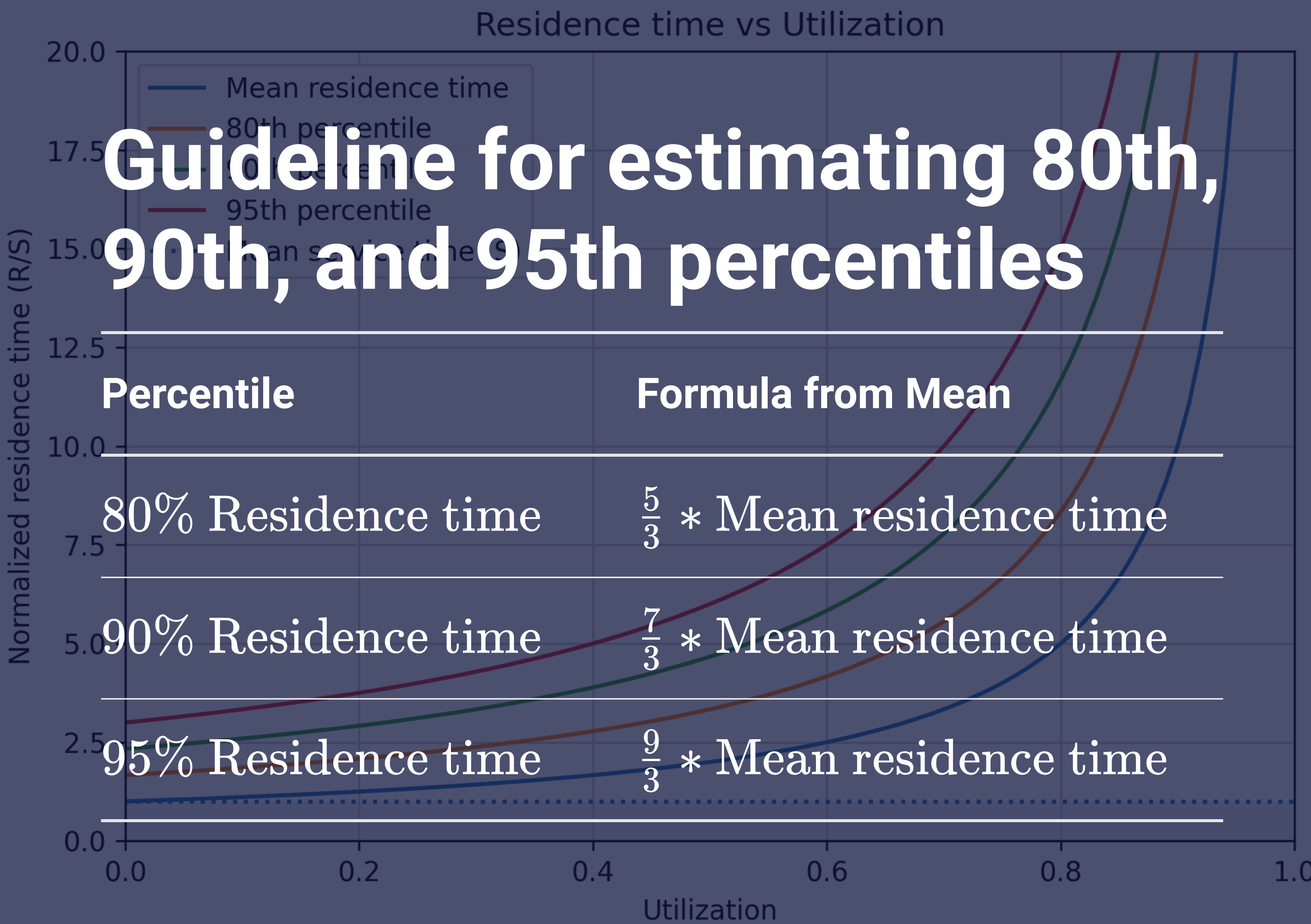
When the idle time went from 50% to 25% (utilization went from 50% to 75%), the expected residence time went from 2 times the service time to 4 times the service time.


Residence time vs Utilization
Mean residence time
Mean service time (S)
Mean residence time = 10 * S
at 90% utilization
Normalized residence time (R/S)
Utilization
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

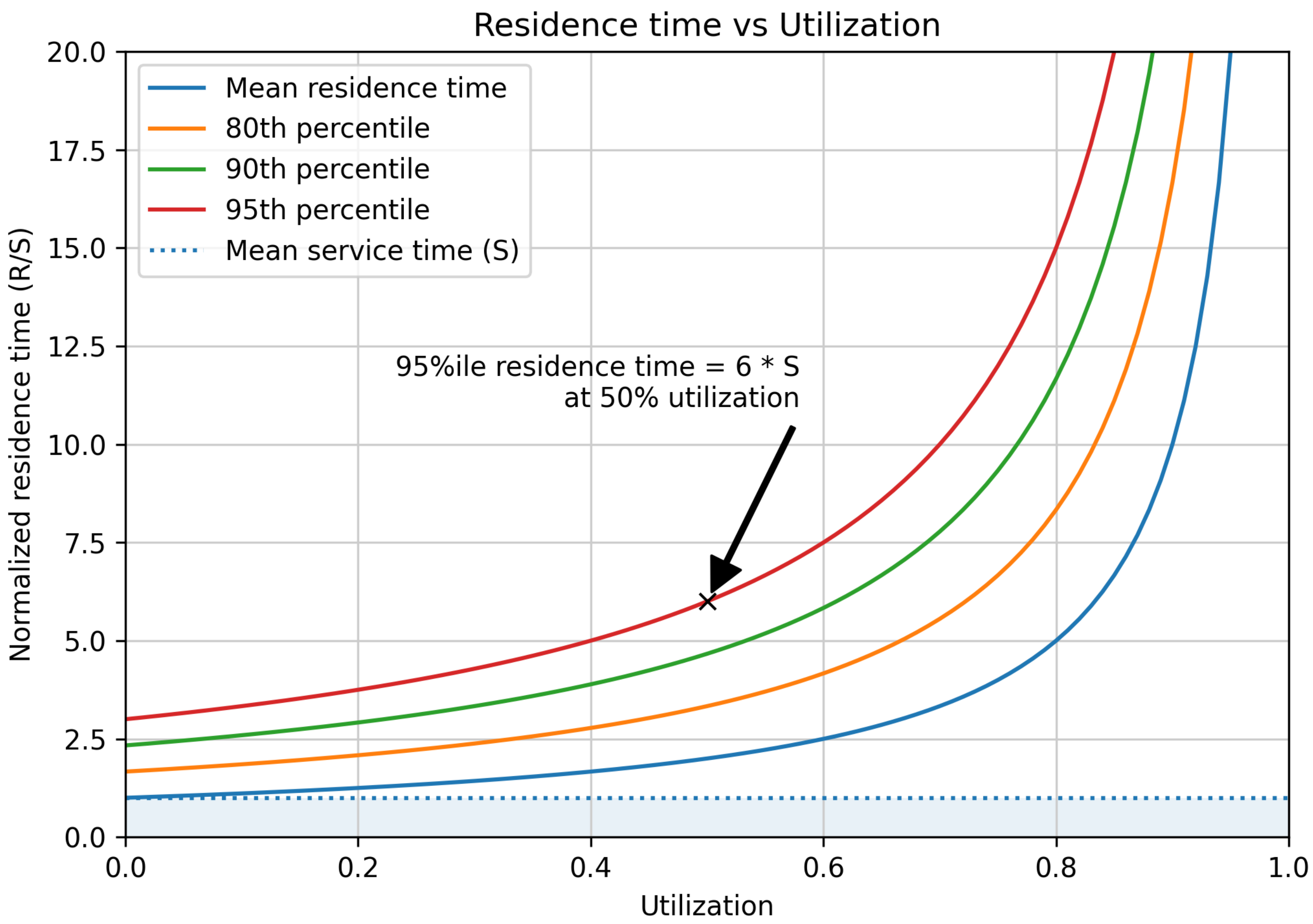


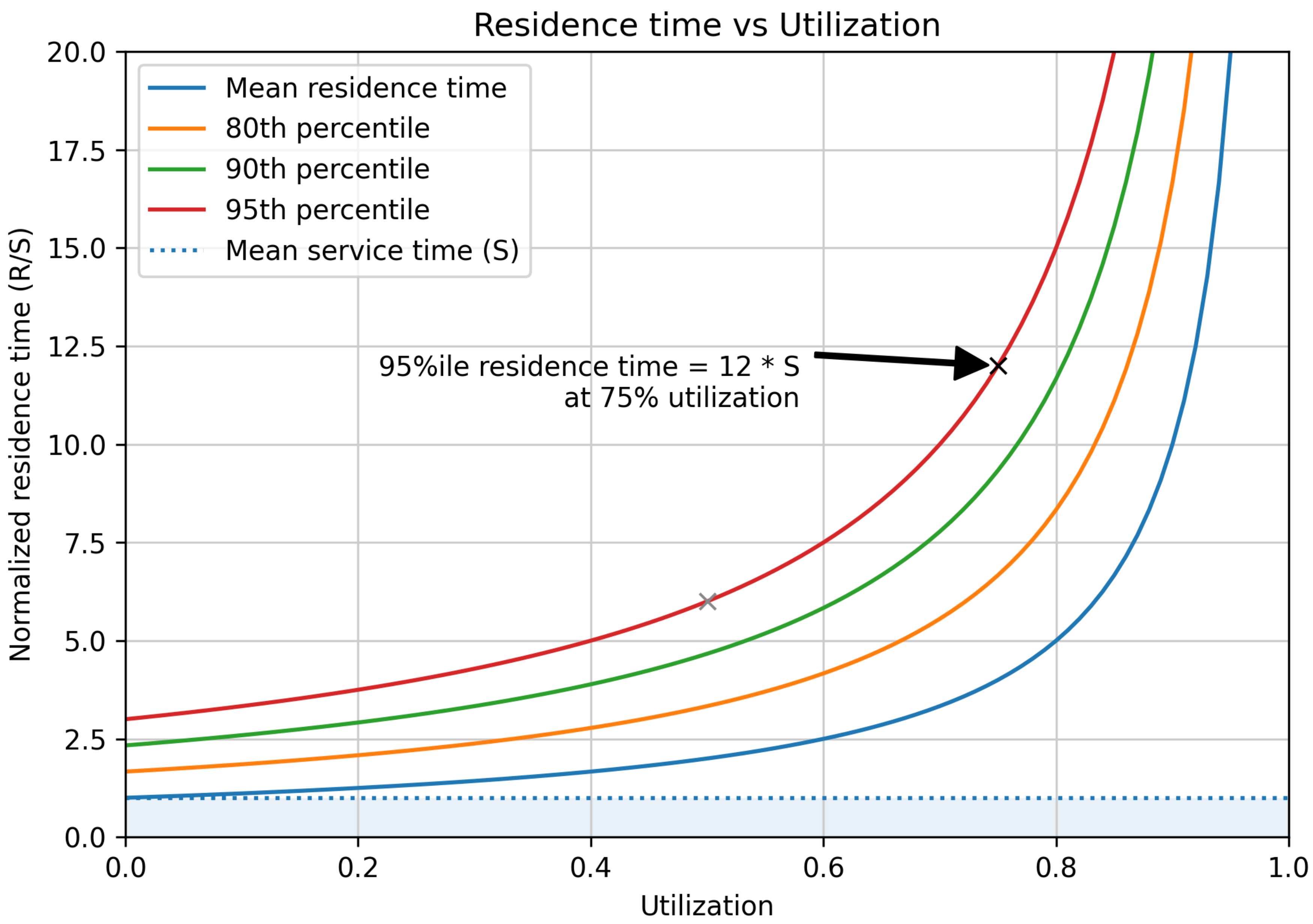
Residence time vs Utilization
Mean residence time
80th percentile
90th percentile
95th percentile
Mean service time (S)
Normalized residence time (R/S)
Utilization
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0
0.2
0.4
0.6
0.8
1.0

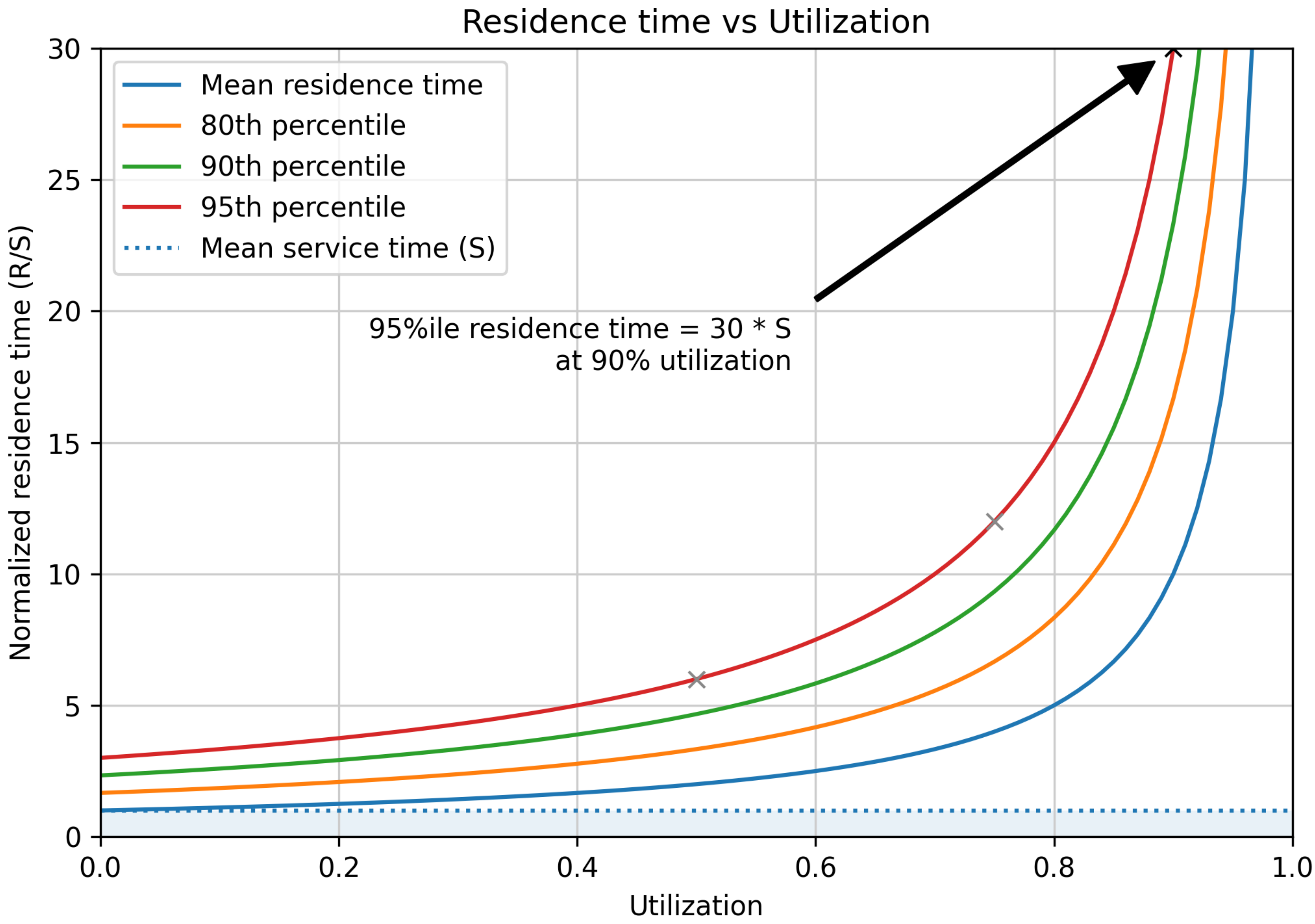
Guideline for estimating 80th, 90th, and 95th percentiles



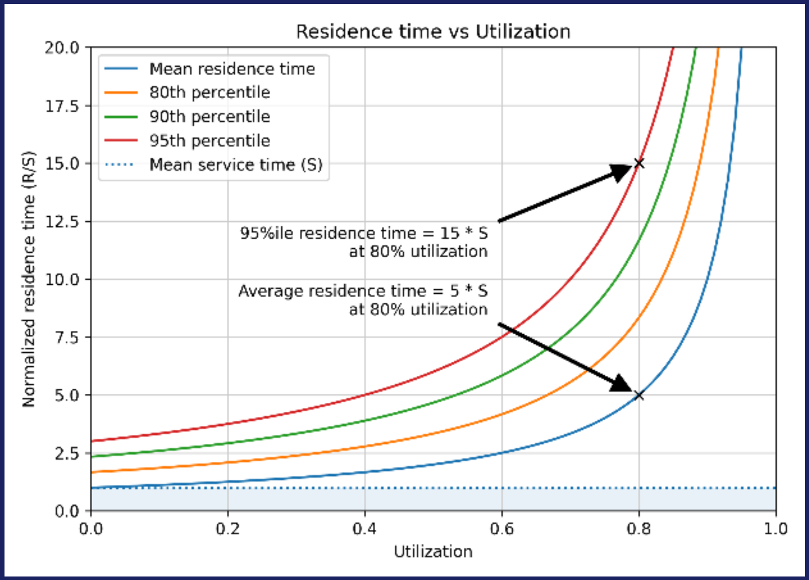
Percentile	Formula from Mean
80% Residence time	$\frac{5}{3} *$ Mean residence time
90% Residence time	$\frac{7}{3} *$ Mean residence time
95% Residence time	$\frac{9}{3} *$ Mean residence time


Residence time vs Utilization
Normalized residence time (R/S)
Utilization
Mean residence time
80th percentile
90th percentile
95th percentile
Mean service time (S)
95%ile residence time = 6 * S
at 50% utilization
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

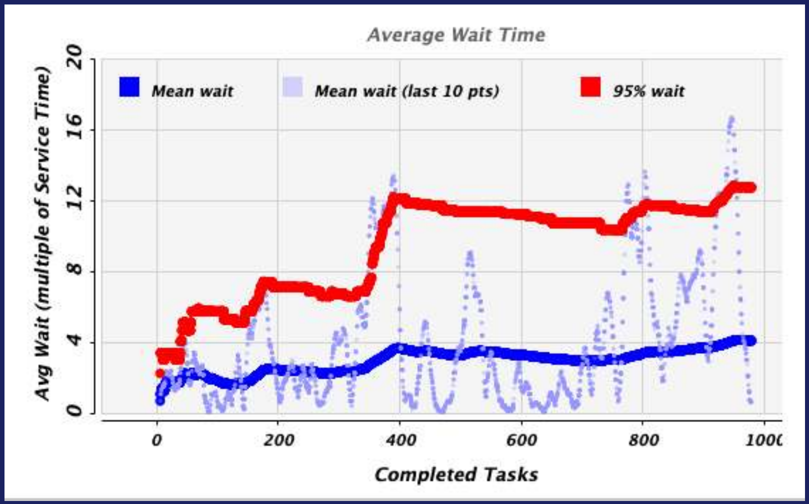

Residence time vs Utilization
Mean residence time
80th percentile
90th percentile
95th percentile
Mean service time (S)
95%ile residence time = 12 * S
at 75% utilization
Normalized residence time (R/S)
Utilization
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Residence time vs Utilization
Mean residence time
80th percentile
90th percentile
95th percentile
Mean service time (S)
95%ile residence time = 30 * S
at 90% utilization
Normalized residence time (R/S)
Utilization
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0

Looking at our earlier data for 80% utilization and variable arrival and service times



The chart on top predicts an average wait time of 4X service time and a 95% wait time of 14X service time



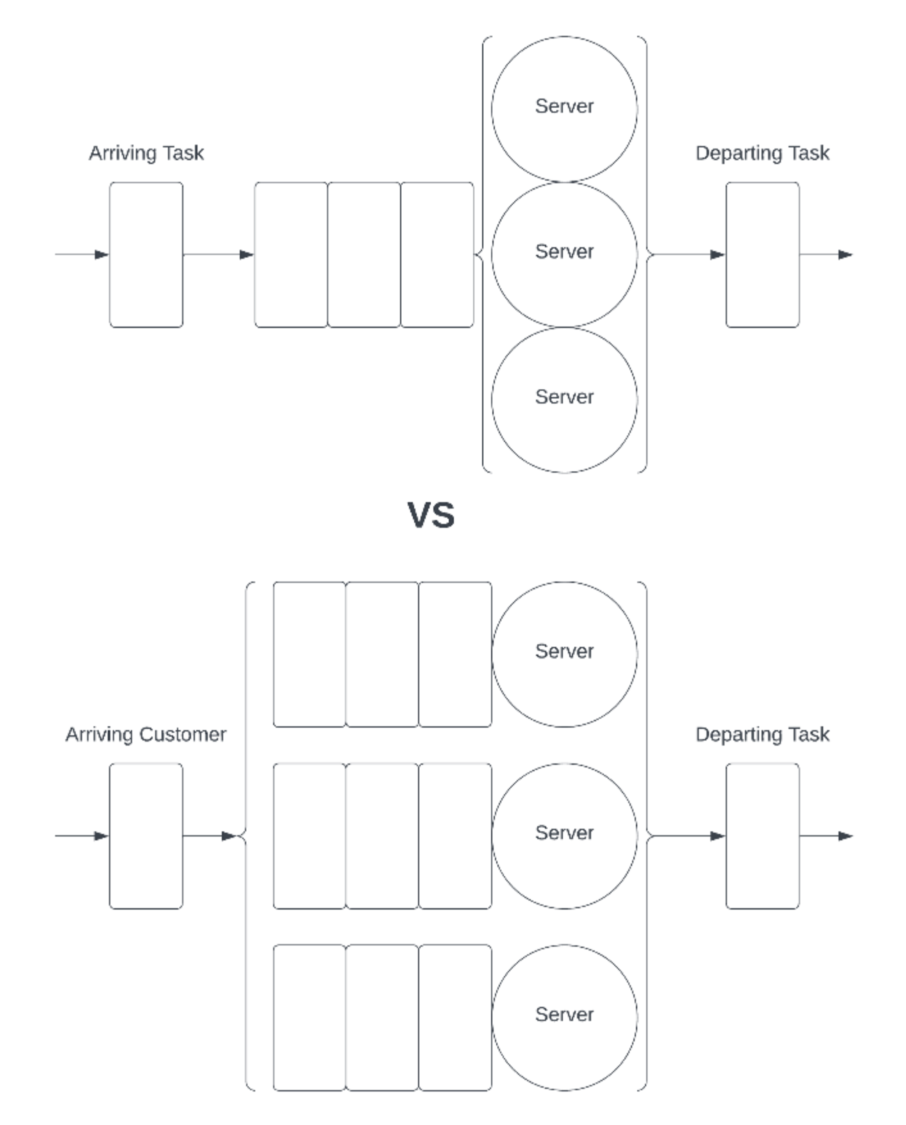
The chart on the bottom looks pretty close (~4X and ~13X)

Quick reference table

Utilization	Mean residence time	95% residence time
0%	$1S$	$1S$
50%	$2S$	$6S$
75%	$4S$	$12S$
90%	$10S$	$30S$

One queue per server or one queue for ALL servers?

Which is better?

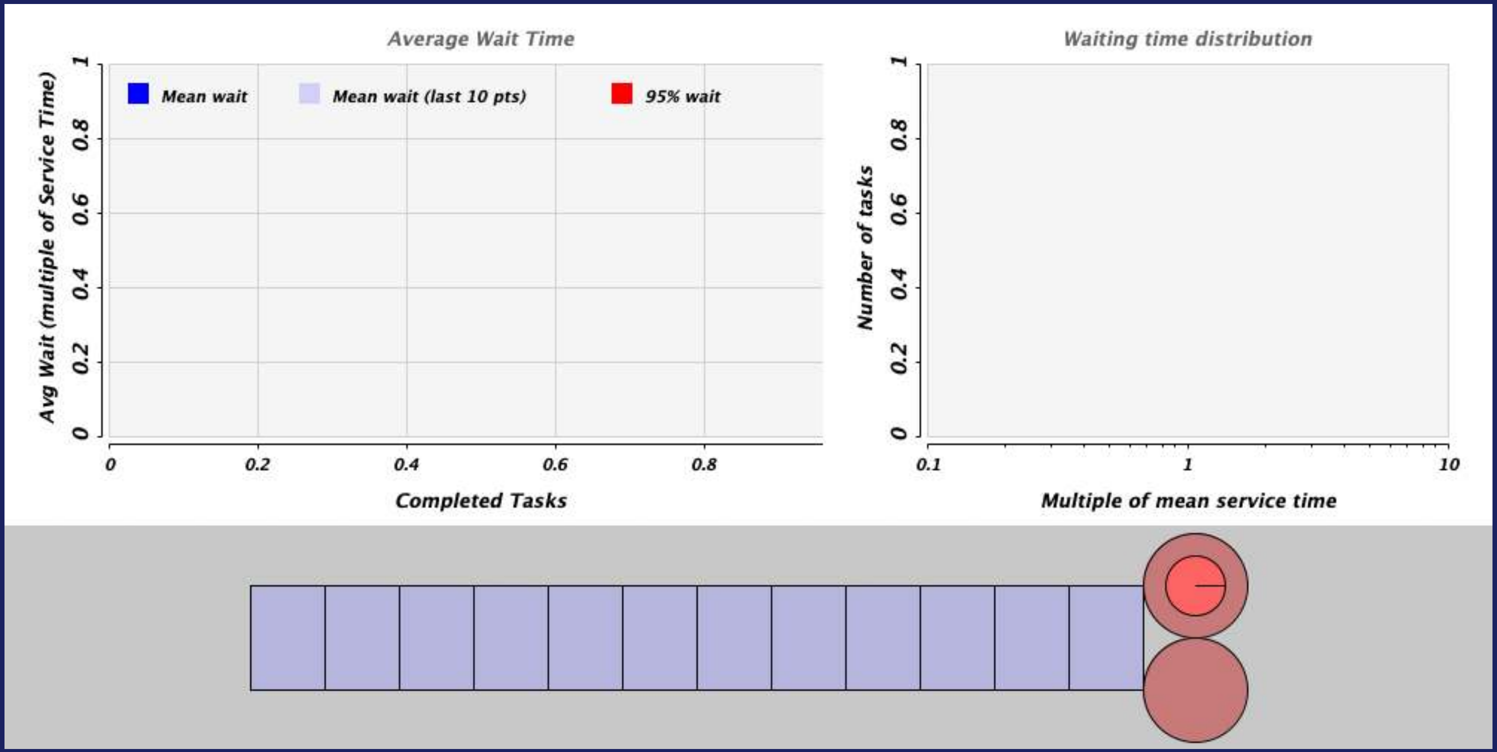


Two queues, two servers

There is no need to simulate two servers, each with their own queue.

We can just use our data from before for a single server/queue pair and pretend there were two of them

One queue, two servers



(link to gif)

Multiple servers (approximate formula)

$$W = \frac{S}{m} * \underbrace{\left(\frac{\rho^{\sqrt{2(m+1)}-1}}{1-\rho} \right)}_{\text{How busy}} * \underbrace{\left(\frac{CV_{arrival}^2 + CV_{service}^2}{2} \right)}_{\text{How variable}}$$

W - average waiting time

S - average service time

m - number of servers

ρ - utilization

Multiple servers (approximate formula)

For m servers:

$$W = \frac{S}{m} * \underbrace{\left(\frac{\rho^{\sqrt{2(m+1)}-1}}{1-\rho}\right)}_{\text{How busy}} * \underbrace{\left(\frac{CV_{arrival}^2 + CV_{service}^2}{2}\right)}_{\text{How variable}}$$

For one server:

$$W = S * \underbrace{\left(\frac{\rho}{1-\rho}\right)}_{\text{How busy}} * \underbrace{\left(\frac{CV_{arrival}^2 + CV_{service}^2}{2}\right)}_{\text{How variable}}$$

Multiple servers (approximate formula)

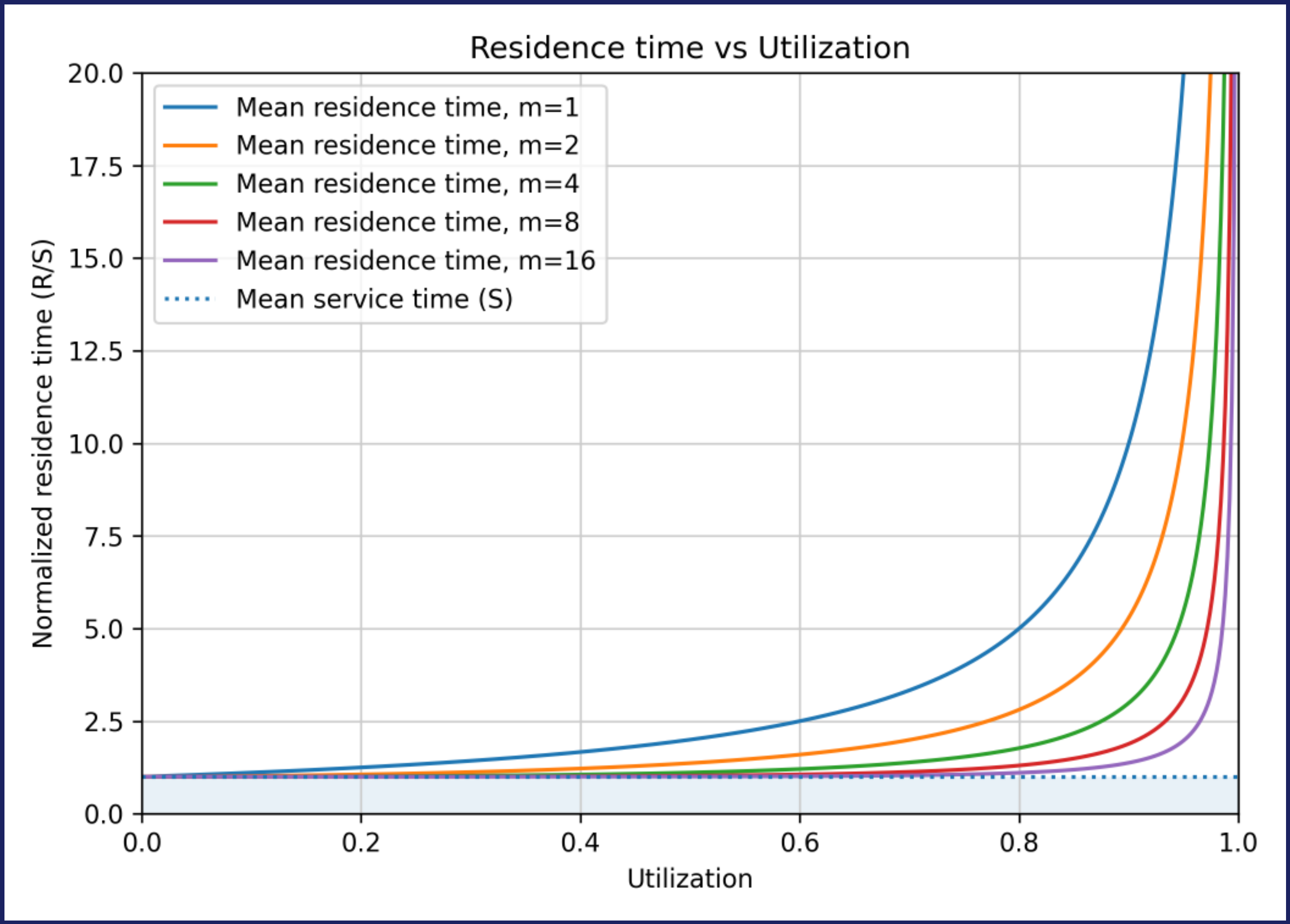
For m servers:

$$W = \frac{S}{m} * \underbrace{\left(\frac{\rho^{\sqrt{2(m+1)}-1}}{1-\rho} \right)}_{\text{How busy}}$$

For one server:

$$W = S * \underbrace{\left(\frac{\rho}{1-\rho} \right)}_{\text{How busy}}$$

Multiple servers



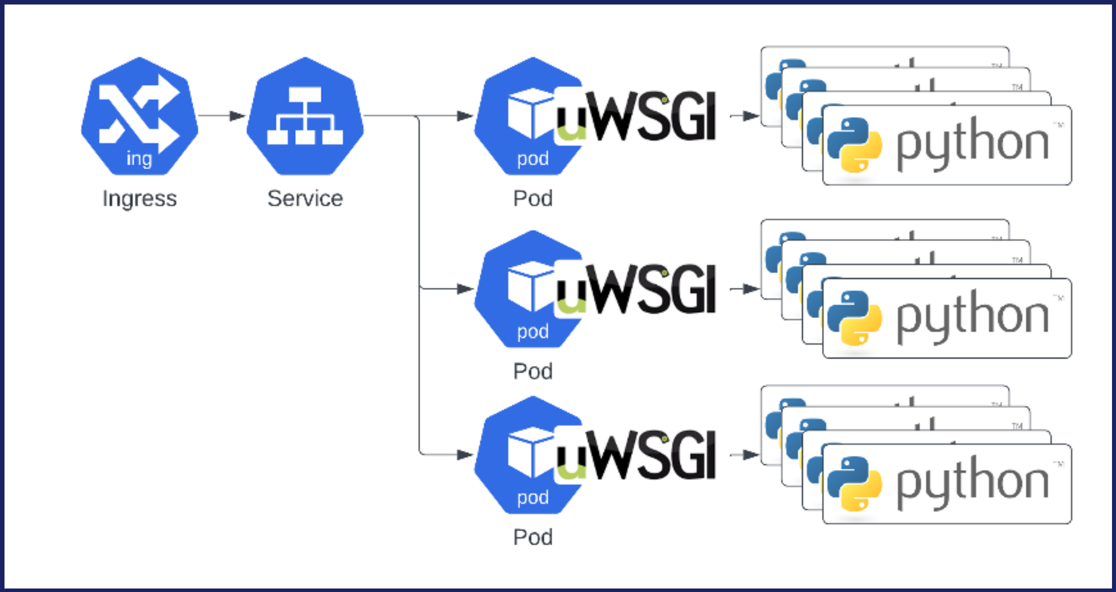
Practical Applications

Practical Applications

Disclaimer: Just about everything practical is an approximation.

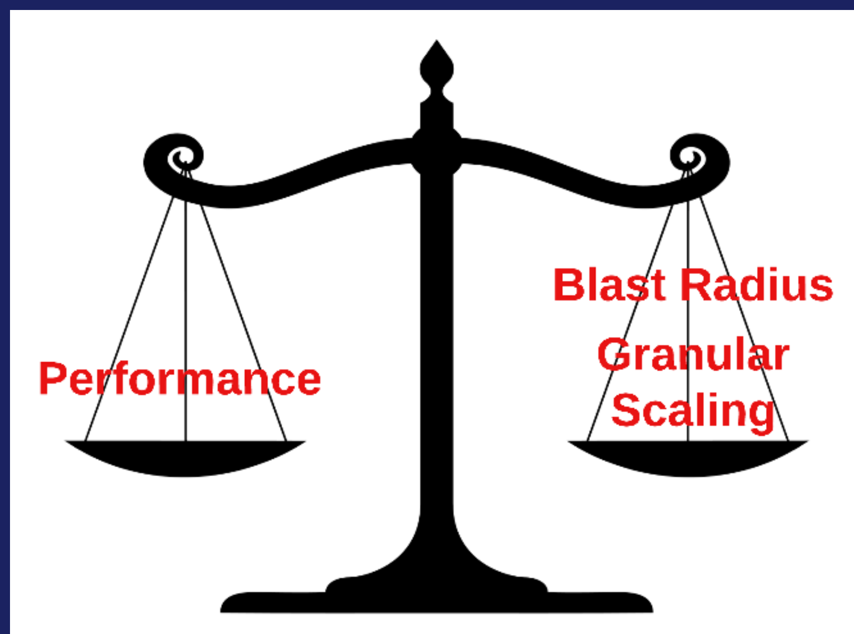
uWSGI Behind a Load Balancer

Each uWSGI process runs several Python processes and has it's own internal request queue



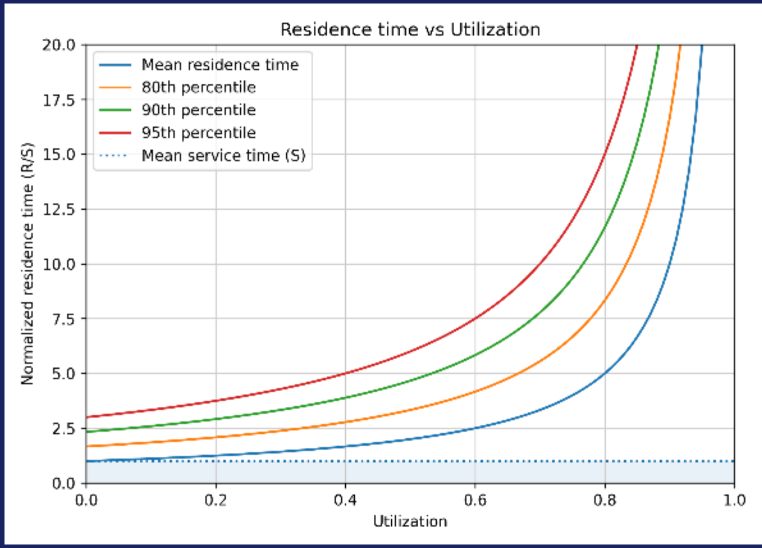
uWSGI Behind a Load Balancer

How many uWSGI instances should be run, versus how many processes run behind each?



Load averages, CPU usage, and latency

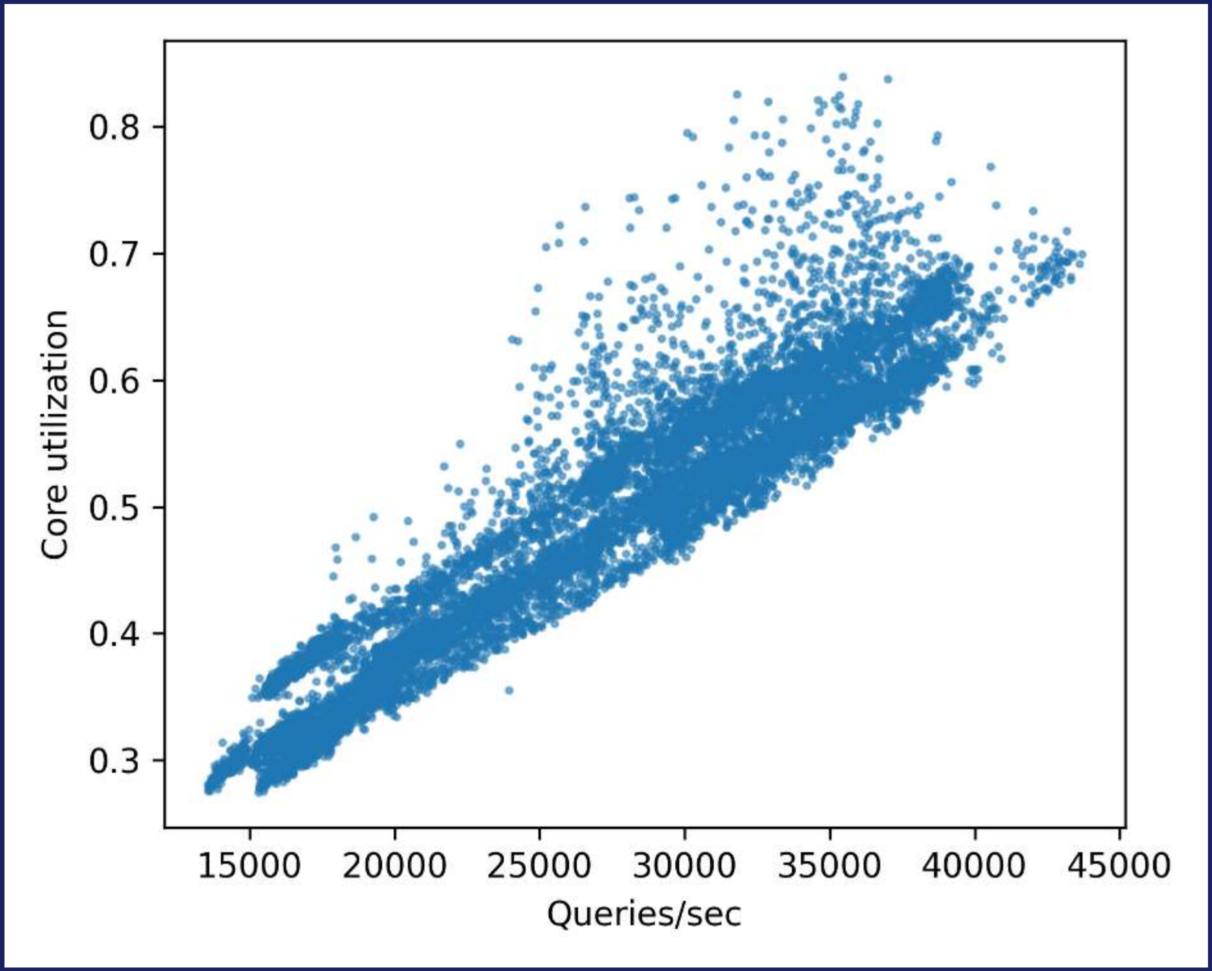
Remember this?



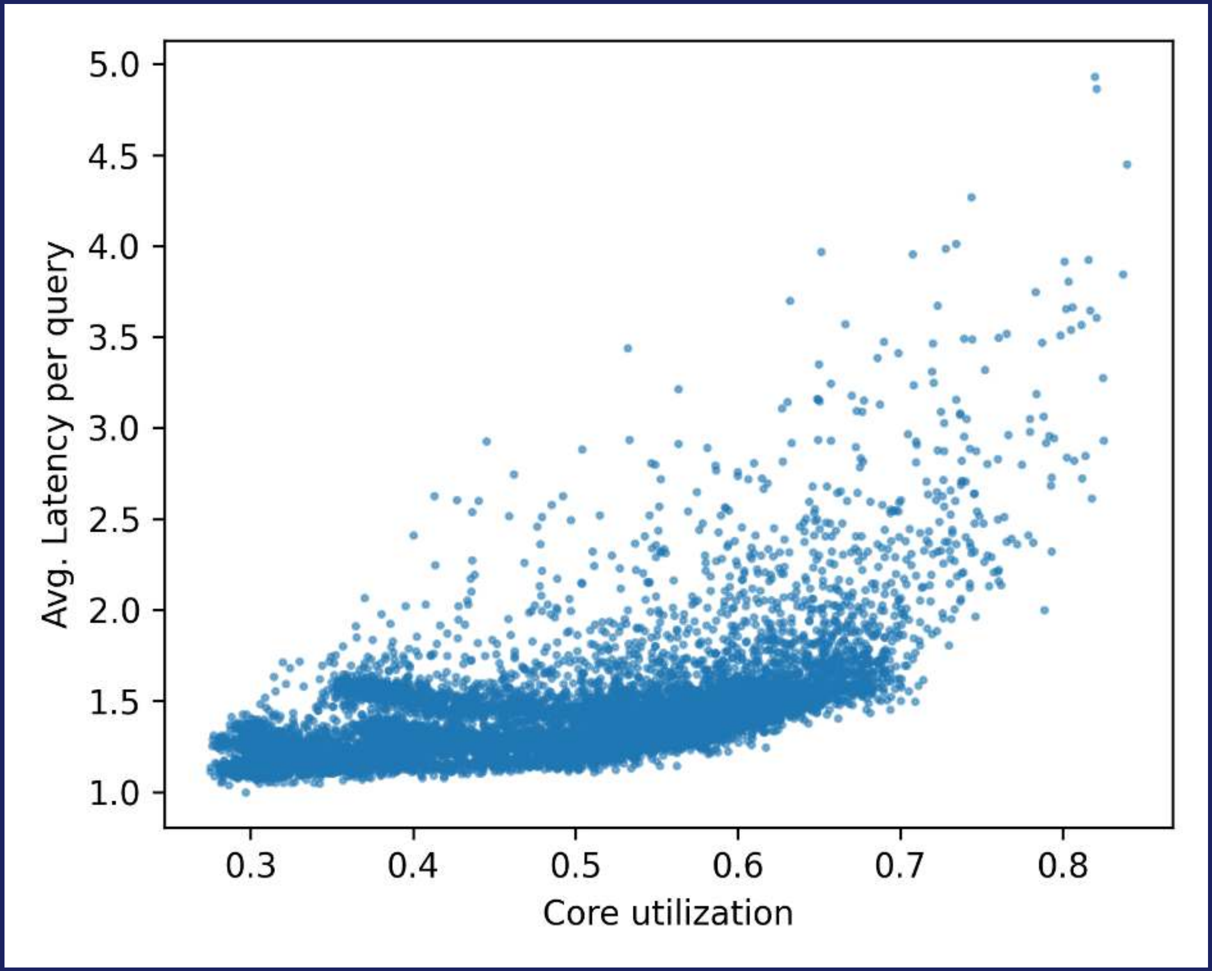
Think of the bottom axis as your % busy CPU. You can be "only" using 80% of your CPU, and still see latency climbing. If you approach 100%, latency goes to infinity (not good).

Guideline: Stay below 80% CPU usage. Lower if you are latency sensitive. <50% if you are VERY latency sensitive.

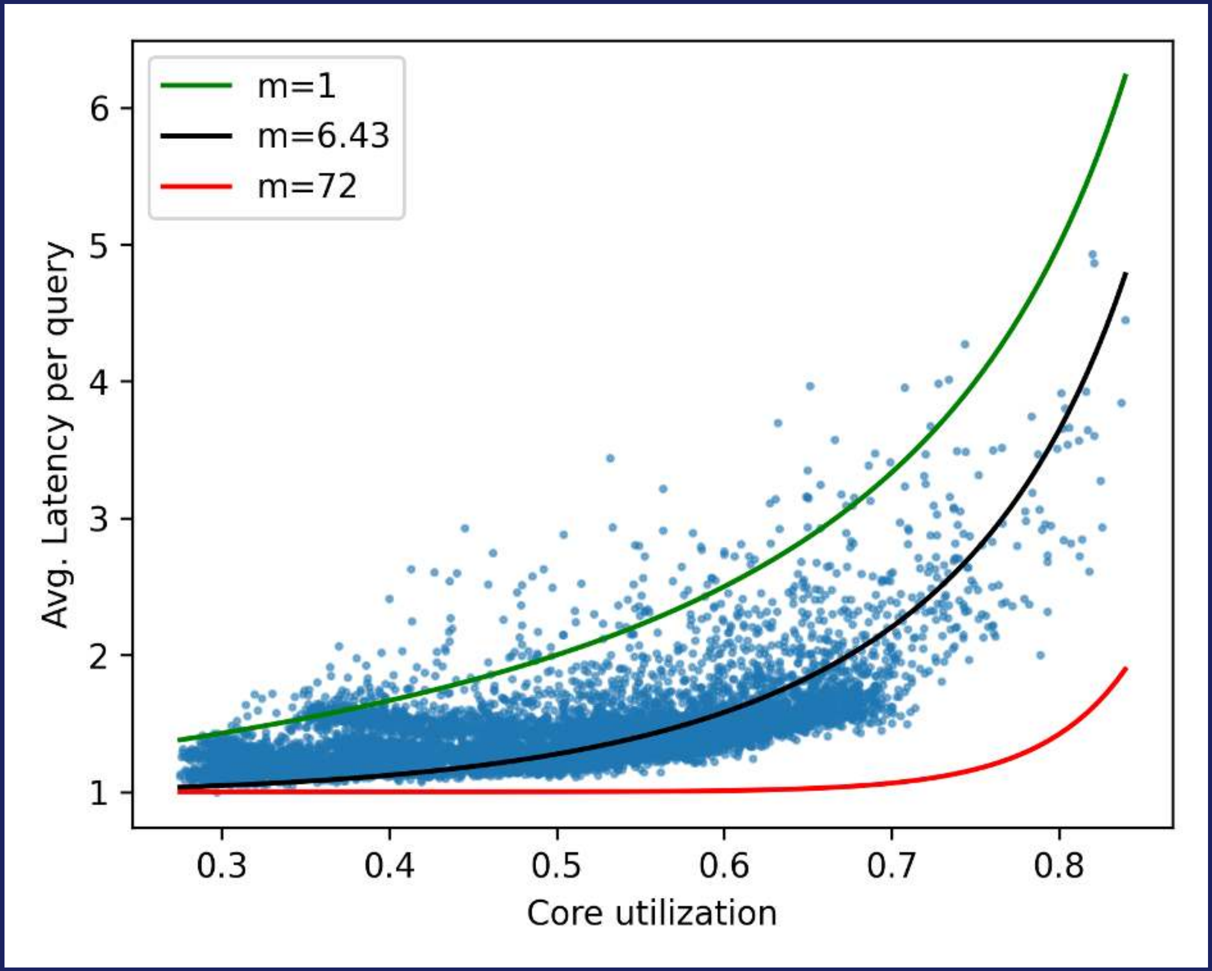
This database server looks neat



This database server looks neat



This database server looks neat



Prometheus queries used (MongoDB exporter)

Average query latency

```
sum(rate(mongodb_ss_opLatencies_latency{op_type="reads"}[1m])) by
(instance) /
sum(rate(mongodb_ss_opLatencies_ops{op_type="reads"}[1m])) by
(instance)
```

Query rate

```
sum(rate(mongodb_ss_opLatencies_ops{op_type="reads"}[1m])) by
(instance)
```

CPU Utilization

```
1-(sum(rate(mongodb_sys_cpu_idle_ms[1m])) by (instance) /
(1000*sum(mongodb_sys_cpu_num_cpus) by (instance)))
```

Other approaches

You can build a full queueing model of your system.

- This is what PDQ does
- Neil Gunther has a book on this

> "All models are wrong, but some are useful"

Other approaches

You can try to model your system with the Universal Scaling Law

- Originally developed by...Neil Gunther. He also has a book on this.
- Accounts for the fact that most systems have some non-parallelizable work that prevents them from scaling linearly (Amdahl's Law - α in the equation) and there is often some coordination penalty that can make the system get slower if you scale it past a certain point (β in the equation). $X(N)$ is the throughput at a given load, N.

$$X(N) = \frac{\gamma N}{1 + \alpha(N-1) + \beta N(N-1)}$$

Guideline Summary

Every time the idle time gets cut in half, the expected residence time doubles.

Stay below 80% CPU usage. Lower if you are latency sensitive. <50% if you are VERY latency sensitive.

Increasing variability makes the utilization/latency graph get worse faster. More servers off one queue makes it stay good longer.

Guidelines for residence time from utilization in a single-server system with exponential arrival and service time

Utilization	Mean residence time	95% residence time
0%	$1S$	$1S$
50%	$2S$	$6S$
75%	$4S$	$12S$
90%	$10S$	$30S$

Guideline for estimating 80th, 90th, and 95th percentiles

Percentile	Formula from Mean
80% Residence time	$\frac{5}{3} *$ Mean residence time
90% Residence time	$\frac{7}{3} *$ Mean residence time
95% Residence time	$\frac{9}{3} *$ Mean residence time

Other resources

Book: Analyzing Computer System Performance with Perl::PDQ - **Neil J. Gunther**

Book: Guerrilla Capacity Planning - **Neil J. Gunther**

Performance Dynamics - perfdynamics.com

Talk: Queueing Theory in Practice: Performance Modeling for the Working Engineer **- Eban Freeman - USENIX LISA17**

Talk: Scalability Is Quantifiable: The Universal Scalability Law **- Baron Schwartz - USENIX LISA17**